

AD-A151 833

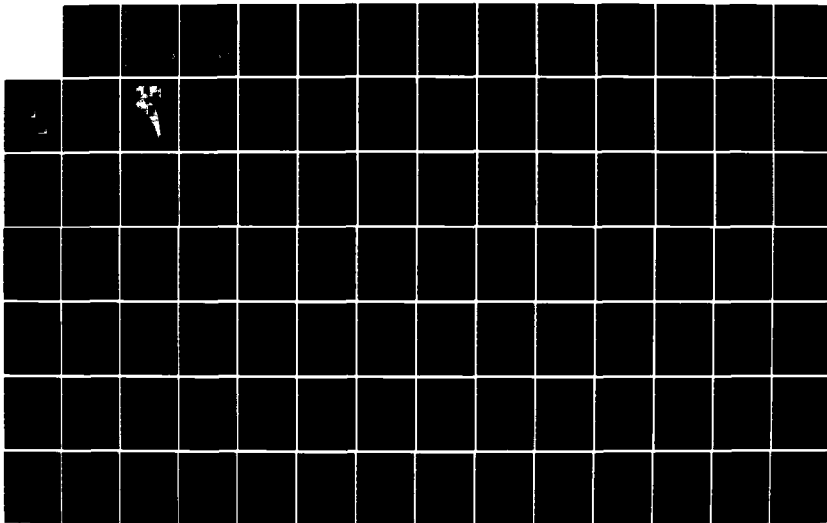
MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE
SEARCH PLANNING(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

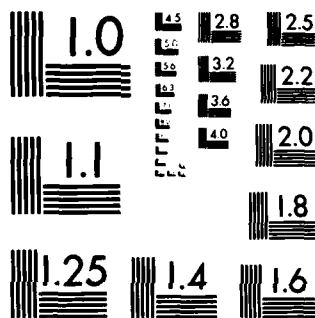
1/2

UNCLASSIFIED

D R DOUGLAS 14 DEC 84 AFIT/GSO/MATH/84D-1 F/G 6/7

NL





AD-A151 833

DTIC FILE COPY



MICROCOMPUTER APPLICATION OF
AEROSPACE ASSET SURFACE SEARCH PLANNING
THESIS

Don Richard Douglas
Captain, USAF

AFIT/GSO/MATH/84D-1

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

DTIC
ELECTE
MAR 29 1985

S

D

B

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

85 03 13 166

AFIT/GSO/MATH/84D-1

MICROCOMPUTER APPLICATION OF
AEROSPACE ASSET SURFACE SEARCH PLANNING
THESIS

Don Richard Douglas
Captain, USAF

AFIT/GSO/MATH/84D-1

DTIC
ELECTE
MAR 29 1985
S B D

Approved for public release; distribution unlimited

AFIT/GSO/MATH/84D-1

**MICROCOMPUTER APPLICATION OF
AEROSPACE ASSET SURFACE SEARCH PLANNING**

THESIS

**Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology**

Air University

**In Partial Fulfillment of
Requirements for the Degree of
Master of Science in Space Operations**

Don Richard Douglas

Captain, USAF

December 1984

Approved for public release; distribution unlimited

Acknowledgments

I wish to thank my thesis advisor, Dr. Panna B. Nagarsenker, Professor of Mathematics at the Air Force Institute of Technology (AFIT), for her guidance and support throughout this thesis effort. Our regularly-scheduled meetings provided me with ingenious insights into program design and kept me firmly goal-oriented. The assistance and recommendations of my reader, Lt. Colonel Joseph W. Coleman, USAF, Professor of Operations Research, were also of great merit.

Additionally, I am indebted to the faculty of the U. S. Coast Guard's (USCG) National Search and Rescue (SAR) School in New York City, especially Major Charles L. Vessey, USAF. He provided his SAR expertise/insight, recommendations to enhance the usefulness of the designed programs, and sample problems with solutions, enabling validation of the search programs generated for this project.

Equally noteworthy was the extensive computer assistance of LTJG Dale G. Streytle, USCG Atlantic Area Command (New York). He helped transfer, compile, link, and get the attached software running on the Coast Guard's standard mainframe computer system (C-3). This enabled the compiled/linked programs to be immediately used by National SAR School faculty, as well as allowing their +65 pages of source code to be electronically downloaded to interested SAR students having the microcomputers to run them on.

Finally, special appreciation and love goes to the former Paula J. Hartzfeld of Branaugh, Missouri. After foolishly marrying for love instead of money, she valiantly picked up my family responsibilities, in addition to her own, releasing me for full-time AFIT study and thesis research. I also acknowledge the unique contributions of my son, Craig, who ensured never a dull moment in our household during my course of study.

Don Richard "Rick" Douglas



Accession For	
NTIS	☒
DTIC	☐
Unannounced	☐
Availability Codes	
Avail and/or	
Dist	Special
A-1	

Contents

Acknowledgments.....	ii
List of Figures.....	vii
Abstract.....	viii
I. Introduction	
Background.....	1-1
Problem Description.....	1-3
Research Objective.....	1-6
Programming Language Selection.....	1-7
Scope of Project.....	1-9
II. Software Package Design	
Overview.....	2-1
Slicing Problem Into Manageable Pieces.....	2-1
Modular Program Development.....	2-3
Aerospace Drift Determination Program.....	2-4
Aerospace Drift Program Module Descriptions.....	2-8
Surface Drift Determination Program.....	2-10
Surface Drift Program Module Descriptions.....	2-12
Search Area Determination Program.....	2-14
Search Area Program Module Descriptions.....	2-15
III. Program Structure	
Overview.....	3-1
Software Engineering Guidelines.....	3-1
Testing & Error Checking.....	3-1
Calculation Accuracy.....	3-4
Summary Of Other Design Features.....	3-4
Validation/Verification.....	3-8
IV. Epilogue	
Conclusions.....	4-1
Recommendations For Further Research.....	4-2
Bibliography	
Appendix A: Definition of Terms	
Glossary.....	A-1
Acronyms.....	A-3

Appendix B: Search Planning Software User's Guide

Overview.....	B-1
Software Purpose.....	B-1
Data Required.....	B-2
Aerospace Drift Program Inputs.....	B-2
Surface Drift Program Inputs.....	B-3
Search Area Determination Program Inputs.....	B-5
Software Maintenance Responsibility.....	B-6

Appendix C: Sample Problem, Program(s) Run & Solutions

Overview.....	C-1
Sample Problem Introduction.....	C-1
Run First Program: Aerospace Drift.....	C-3
First Program Output.....	C-8
Run Second Program: Surface Drift.....	C-9
Second Program Output.....	C-18
Run Third Program: Search Area Determination....	C-19
Third Program Output.....	C-22
Conclusion.....	C-23

Appendix D: Search Planning Program(s) Listing (Source Code)

Program #1 of 3: Aerospace Drift Determination	
Description/License.....	D-1
Global Variable Declarations.....	D-2
Module #1: Procedure writeln.....	D-3
Module #2: Procedure VerifyDTG.....	D-4
Module #3: Procedure AeroPosition.....	D-5
Module #4: Procedure AddVectors.....	D-6
Module #5: Procedure GlideOrChute.....	D-7
Module #6: Procedure GlideThenChute.....	D-8
Module #7: Procedure Ejection.....	D-11
Module #8: Procedure WindDrift.....	D-12
Module #9: Procedure WindChart.....	D-14
Module #10: Procedure VerifyWinds.....	D-15
Module #11: Procedure InputWinds.....	D-16
Module #12: Procedure AeroGlide.....	D-19
Module #13: Procedure GlideData.....	D-21
Module #14: Procedure ChuteData.....	D-22
Module #15: Procedure WriteToDisk.....	D-24
Module #16: Procedure Warranty.....	D-26
Main Program.....	D-27
Variable & Operator Cross-Referenced Listing...	D-30

Program #2 of 3: Surface Drift Determination	
Description/License.....	D-37
Global Variable Declarations.....	D-38
Module #1: Procedure writelns.....	D-40
Module #2: Procedure VerifyDTG.....	D-41
Module #3: Procedure SurfacePosition.....	D-42
Module #4: Procedure WindPeriods.....	D-44
Module #5: Procedure DaysInMonth.....	D-45
Module #6: Procedure PeriodTimes.....	D-46
Module #7: Procedure InputSeaWinda.....	D-48
Module #8: Procedure WindChart.....	D-49
Module #9: Procedure VerifyWinda.....	D-49
Module #10: Procedure AddVectors.....	D-50
Module #11: Procedure WindLatCoeffs.....	D-51
Module #12: Procedure WindCurrent.....	D-52
Module #13: Procedure AvgSurfaceWind.....	D-54
Module #14: Procedure DriftDirUncertain.....	D-55
Module #15: Procedure LeewayDrift.....	D-56
Module #16: Procedure SeaCurrent.....	D-58
Module #17: Procedure Datum.....	D-60
Module #18: Procedure Warranty.....	D-62
Module #19: Procedure RecordCurrents.....	D-63
Module #20: Procedure WriteToDisk.....	D-66
Main Program.....	D-69
Variable & Operator Cross-Referenced Listing...	D-73

Program #3 of 3: Search Area Determination	
Description/License.....	D-82
Global Variable Declarations.....	D-83
Module #1: Procedure writelns.....	D-84
Module #2: Procedure Position.....	D-84
Module #3: Procedure AeroSurfVectors.....	D-85
Module #4: Procedure NewCoordinates.....	D-87
Module #5: Procedure FindXYaToCorner.....	D-88
Module #6: Procedure FindCoordinates.....	D-88
Module #7: Procedure AreaSearch.....	D-90
Module #8: Procedure WriteToDisk.....	D-92
Module #9: Procedure Warranty.....	D-95
Main Program.....	D-96
Variable & Operator Cross-Referenced Listing...	D-97

Vita

List Of Figures

<u>Figure</u> -----	<u>Page</u> ----
1 Survivor Recovery Time.....	1-2
2 Human Life Expectancy In Water.....	1-4
3 Search Stages.....	1-11
4 Final Search Planning Software Statistics.....	2-2
5 Average Winds Aloft Example.....	2-3
6 First Path: Drift Method = Glide Only.....	2-5
7 Second Path: Drift Method = Parachute Only....	2-6
8 Third Path: Drift Method = Glide & Parachute..	2-7
9 Surface Drift Determination Program Flow Chart	2-11
10 Search Area Determination Program Flow Chart..	2-14

Abstract

This paper tackles the most time-consuming and complicated type of search and recovery planning -- calculating the approximate surface position of an aerospace object which has been affected over time by glide or parachute winds aloft, as well as surface current winds, leeway drift, and sea current vectors. The three, highly-interactive, search applications programs herein are written in Standard Pascal using Borland International's "TURBO Pascal" (an inexpensive software package available for virtually every microcomputer on the market). They have been tested on a small, portable, 64K memory, Z-80A processor-based microcomputer (Osborne One), a Convergent Technologies C-3 Data System, and a Digital Equipment Corporation VAX 11-780 mainframe.

Search and Rescue/Recovery (SAR) in the United States is based on the humanitarian principle which compels people to render aid to those in distress. Search planning guidelines and formulae to help locate persons in distress or missing aerospace objects are described in the National Search and Rescue Manual (AFM 64-2). This methodology has not been implemented for microcomputers in a compiled, transportable programming language like Pascal. This research project does just that. It does not, however,

teach the guidelines or formulae. The reader MUST have a solid understanding of SAR methodology before using the attached software package to assist in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, the attached software package is recommended for training use only.

Appendices include: glossary, user's guide, sample problem with program runs and solutions, and source code listing.

Chapter 1

MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE SEARCH PLANNING

Background

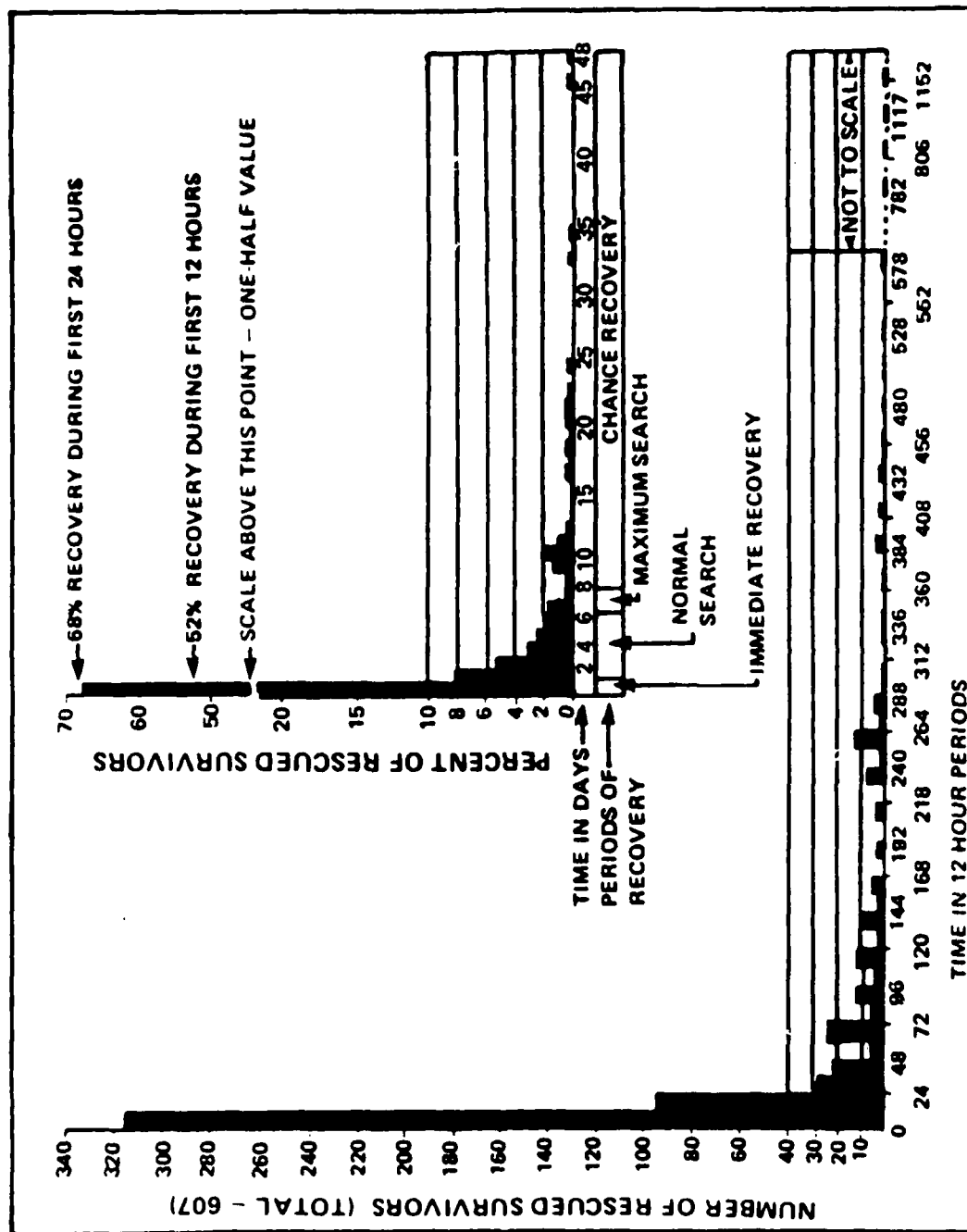
What is the value of a human life? Each year millions of dollars and countless manhours are expended searching for missing persons in distress. International, national, state, and local government search and rescue/recovery (SAR) agencies commit limited resources looking for the real or perceived victims of accidents or incidents such as sinking vessels or airliner or light plane crashes. And, as humans increasingly orbit, fly and sail over the Earth, the number of such missing/distressed persons can only multiply.

Likewise, as man launches and recovers increasing amounts of equipment to and from space, more of it will, undoubtedly, land somewhere other than intended. Remember when no one knew where the uncontrolled US Skylab or the Soviet nuclear-powered COSMOS satellite(s) would crash once they re-entered the Earth's atmosphere?:

There is no way to tell exactly where between 50 North and 50 South latitude Skylab debris will fall even as late as one orbit prior to entry. . . Skylab weighs about 175,000 lbs., and analysis indicates that 40,000-50,000 lbs. could survive reentry. (5:19)

No one knew where these spacecraft finally did land until several hours/days of searching located the missing pieces (1:33,7:1-6,9:1-5).

Survivor Recovery Time



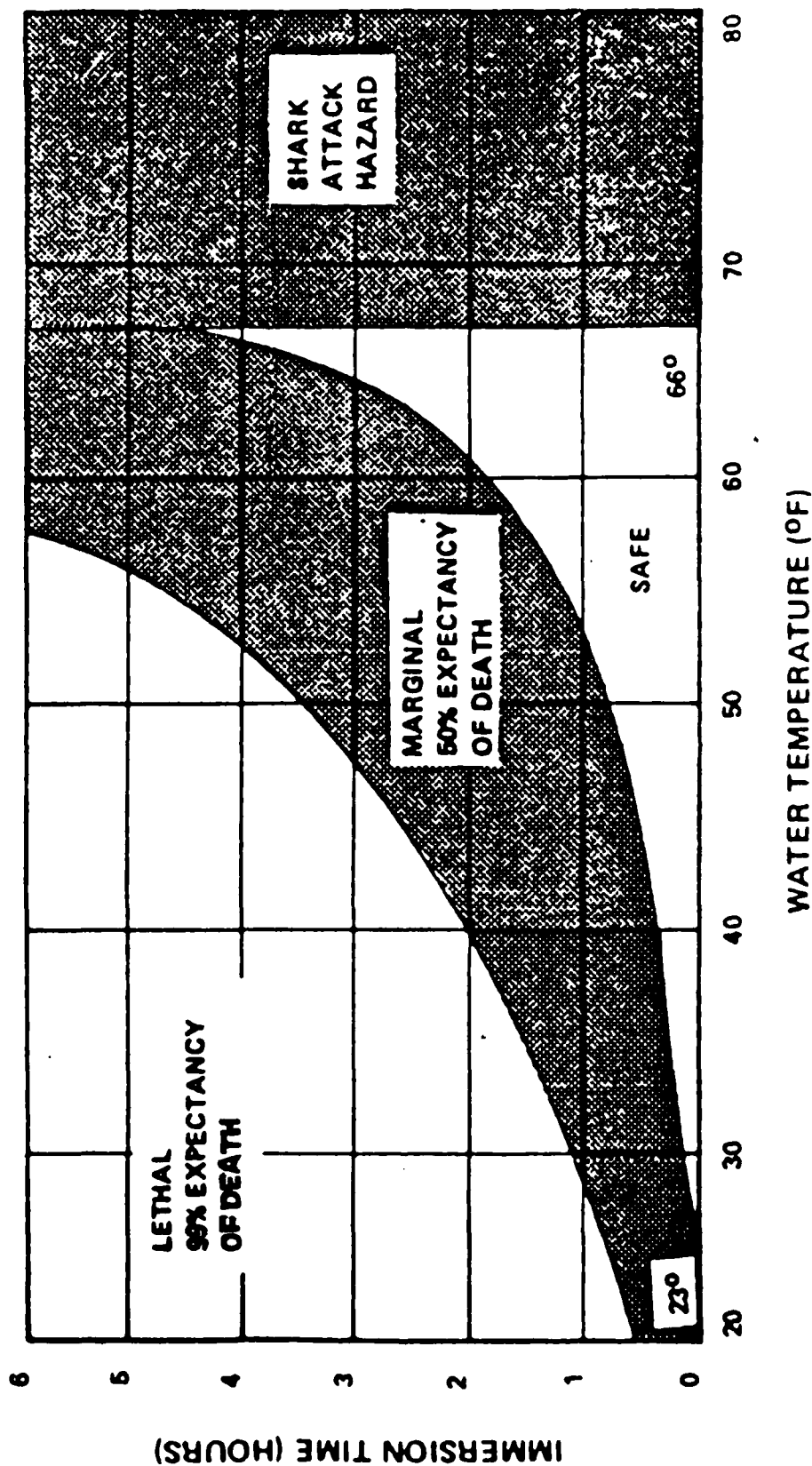
The United States Coast Guard (USCG) is tasked with training Federal military and civil agencies in search planning (10:2-7). For this purpose they maintain the National SAR School in New York City. After tabulating many years of search statistics, USCG developed a lengthy, yet comprehensive, manual method of search area determination using vector addition, algebra, tables, graphs, and nomographs. This technique is taught over a 3-week course, where the typical oceanic problem takes experienced planners, assisted by electronic calculators, several hours to complete (8).

Unfortunately, these same hours may be spent by injured survivors on land or in the water awaiting recovery (see Figure 1). Or, valuable equipment may receive environmental exposure damage (e.g., corrosive seawater) before search forces are directed where to look for it.

Problem Description

The National SAR School manual search planning technique is not available for microcomputer users in a widely-transferable, high-level, compiled programming language, like Pascal (8). Such a software package could save precious time (see Figure 2) by providing appropriate agencies with preliminary data on which to initiate search planning. This would result in search forces being organized and launched much sooner, saving more lives and critical resources. The search planning technique is

Water Chill Without Antilexposure Suit



available, however, on the USCG mainframe computer system at Governor's Island, New York, with some limitations on capability and accessibility.

Any discussion of these limitations must first describe current capabilities. An operations research firm in Paoli, PA, received a government contract to design a "Computer Assisted Search Planning" (CASP) program to run on the Coast Guard's Convergent Technologies C-3 Data System computer. The program allows weighted multi-criteria decision-making, draws search location probability maps, and has other features which only embellish the basic manual calculation method and would never fit in the limited memory space of the standard 48-64K RAM microcomputer. Yet, CASP only handles surface search object planning! It lacks the capability of calculating the initial surface/splash-down position of gliding, falling, and/or parachuting aerospace objects (ejecting pilots, satellite packages, etc.). For years, the National SAR School faculty has been forced to manually calculate this aerospace drift position before they could execute their commercially-procured, mainframe computer search planning software (8). Were this aerospace drift capability incorporated into the software created in this thesis effort, then it could be used on the USCG mainframe and microcomputers alike.

The second limitation involves accessibility. Outside agencies desiring rapid and accurate search planning

assistance do not have direct access to USCG's mainframe running the CASP program. They must verbally pass the required information by phone to the Coast Guard, who must key in their data and phone back the results. This method is slow and typographical errors can make it an even longer process. Adding to the problem are possible time-sharing delays, explained by Fraley and Kem:

Due to the increased use of computers in all aspects of management, the number of users attempting to access the computer is normally quite large. This aspect may then cause the response time of non-dedicated remote computer systems to be unacceptable in a time critical environment. (2:4)

These authors conclude with the valid suggestion that microcomputers offer a solution to this accessibility problem.

As a final advantage of developing search planning software for microcomputers, the Coast Guard has expressed interest in freely offering/downloading this software to interested students/graduates on an "optional basis" to maintain the skills they learned at the National SAR School (8).

Research Objective

The primary objective of this project is to design, implement, verify and validate a microcomputer-based aerospace asset search planning tool. The reason is to generate answers much faster, and with greater accuracy, than if manual calculations were used.

The other key objective is to create a search planning tool that is highly-transportable between different types of microcomputers. This entails sub-objectives of keeping the program(s) small and writing it (them) in a high-level, transportable programming language for microcomputers. Therefore, the tool is divided into three programs. This keeps their compiled versions small enough to implement on microcomputers having at least 64K random-access memory (RAM) and one disk drive. The selection of programming language is now described.

Programming Language Selection

The choice of a suitable microcomputer language offered these alternatives: BASIC, FORTRAN, or Pascal. Most microcomputers can be purchased today with their own generic brand of BASIC (Beginner's All-Purpose Symbolic Instruction Code) language. Unfortunately, most microcomputer BASICs are slower, interpreted or pseudo-compiled languages. Interpreted languages translate each program statement into a sequence of machine code instructions which are executed before the next statement is translated. This method takes much longer to run than does pre-translating the program into machine code (or "compiling it", as do Fortran and Pascal) enabling instant execution. Also, BASIC does not allow the use of mnemonic names to give the user a clue as to a variable's purpose in a program. Worst of all, BASICs are often incompatible between computer

types, limiting their portability.

The FORTRAN (FORMula TRANslation) story is even more bleak. Those packages available for microcomputers are non-standard subsets of FORTRAN 77. At time of writing only one software company (Digital Research) had announced development of a complete, standard version of FORTRAN 77 for micros. However, this version only runs on IBM personal computers upgraded (at user expense) to 512K RAM and is scheduled to cost over ten times as much as the language used in this project.

By this process of elimination, Pascal was ultimately selected. This language was designed by Professor Niklaus Wirth of the Eidgenossische Technische Hochschule in Zurich, Switzerland, and named in honor of Blaise Pascal (1623-1662), the famous 17th-century French philosopher and mathematician (6:2-3). Since first introduced in 1971, Pascal has been applied to almost any task on computers and has received increasing attention and use in business and academia worldwide. A recent Soviet journal reports:

Scientists of the Lithuanian Academy of Sciences' Institute of Mathematics and Cybernetics, Department of Systems Programming, selected the so-called Pascal language from several programming languages used by computers (for training programmers). This language reflects the main concepts of programming rather clearly and is highly-suitable for training purposes . . . helping (programmers) to master the fundamentals of programming more quickly and easily.
(11:2)

It is the only microcomputer language to meet the criteria

of being readily-available, highly-transportable, and:

Pascal is trim enough to run in the 48K- to 64K-byte memory limit that characterizes (most personal computers). It is small enough to be easily implemented, and its trimness makes it syntax and semantics easy to specify and relatively easy to grasp. (4:234)

Best of all, Borland International's "TURBO Pascal" used in this project comes complete with a 259-page User's Manual for only \$49.95 retail (\$35 mail order). The attached programs were written in TURBO on an Osborne personal computer and readily compiled and executed under Standard Pascal (Jensen and Wirth rules) on a DEC VAX 11-780 mainframe.

As such, this project's three programs for aerospace asset search planning were developed to run on a minimum system consisting of at least a 64K RAM-equipped microcomputer with one disk drive. The user need not purchase TURBO to compile and execute the programs, if a pre-compiled (compiled for a specific type of personal computer) copy of these programs is acquired. The only software needed to execute the compiled programs is the user's microcomputer operating system. However, if a pre-compiled version is unavailable, the Borland product must be acquired and the source code listed in Appendix D must be keyed in and compiled to run.

Scope of Project

The scope of this project limits validation of

National SAR Manual search planning methodology, which is applied herein to microcomputer programming to calculate the smallest, feasible search area with the highest probability of locating missing aerospace objects over land or in the water. The algorithms incorporate the National SAR School's manual search planning method, making no attempts to verify the accuracy of formulae, graphs, or nomographs the Coast Guard has relied on and updated over many years of successful search and rescue experience. As such, they address: aerospace drift; calculation of average winds aloft; sea, wind, and leeway current drifts; calculation of average surface winds; minimum, maximum, and DATUM_minimax position determination; and, calculation of search area. These areas fit into the overall search planning methodology structure as shown in Figure 3.

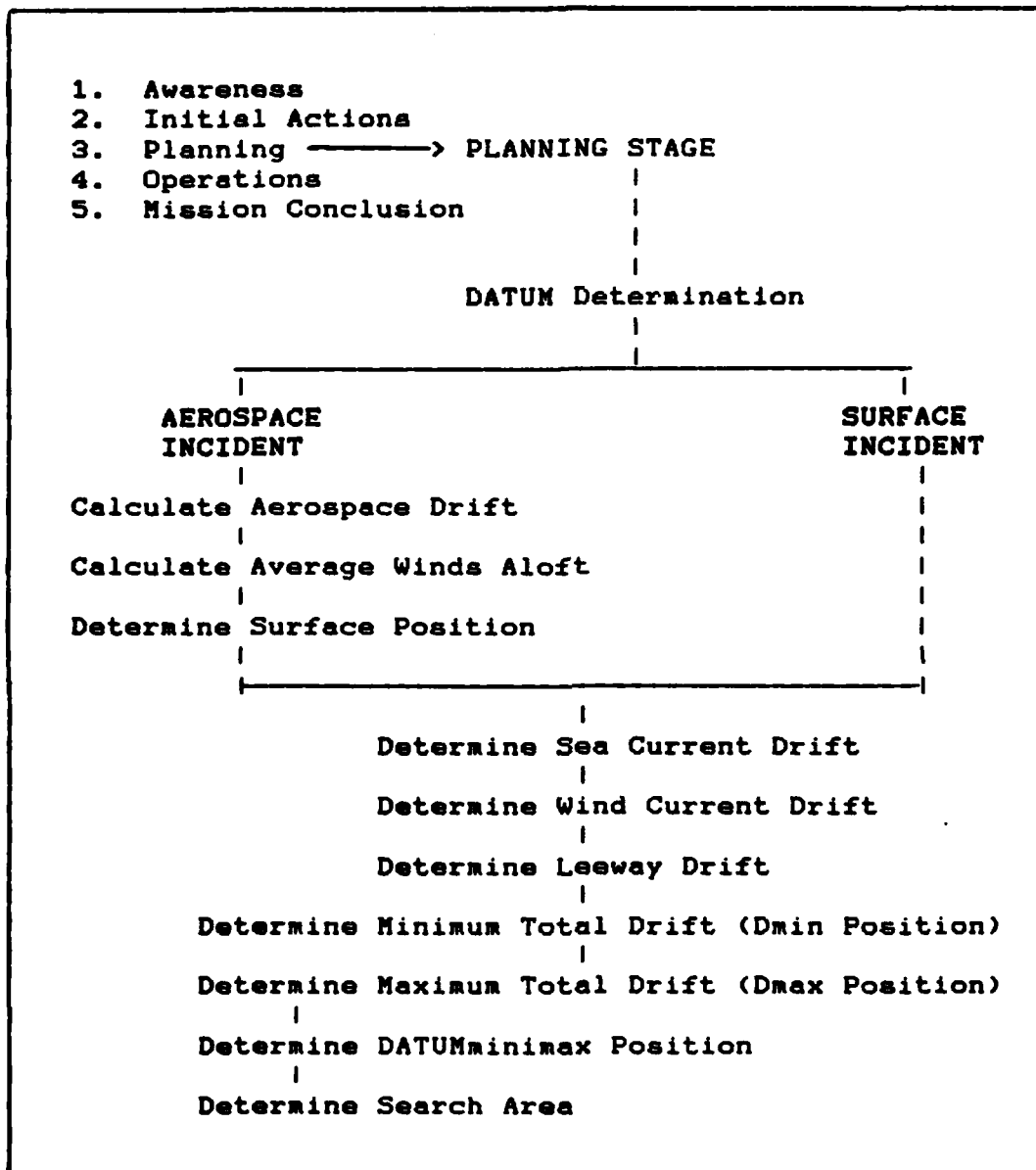


Figure 3. SEARCH STAGES

Chapter 2

Software Package Design

Overview

Software engineering practices (discussed in Chapter 3) were used throughout the development of the attached microcomputer package. Top-down design (breaking a larger problem into smaller pieces for solution and coding) led to the modular development of three programs. This research effort uses top-down program design methodology. Actual coding follows, emphasizing simplistic structures for program clarity to enable simplified modifications by individuals having limited Pascal programming experience. Then comes debugging, testing with USCG-supplied problems, and, finally, drafting of documentation (User's Guide).

Slicing Problem Into Manageable Pieces

The 64K RAM restriction set on available microcomputer memory and limited magnetic disk storage space forced the overall problem to be subdivided into smaller units of manageable size with the added advantage of faster individual modification and re-compilation times (see Figure 4). This search planning implemented in one comprehensive program would not fit most microcomputer memory or disk storage spaces. In particular, the ideal disk would have to store source (183K) and compiled (112K) code, as well as leaving approximately 10K space for input/output record

PROGRAM NAME	SOURCE CODE	COMPILED CODE	TOTAL LINES	TOTAL PAGES
AeroDrift	67K	38K	1468	27
SurfaceDrift	81K	50K	1839	34
SearchArea	35K	24K	777	15
TOTAL	183K	112K	4084	76

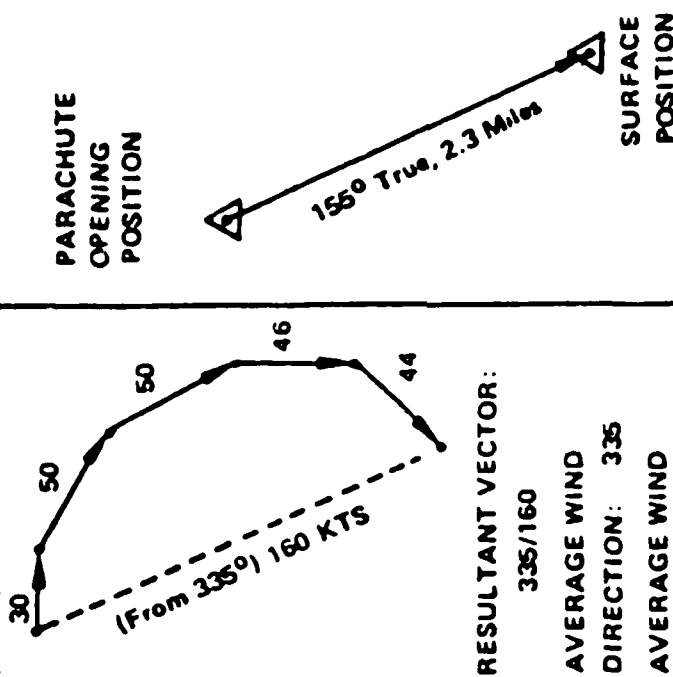
Figure 4. FINAL SEARCH PLANNING SOFTWARE STATISTICS

files created during program runtime. Additionally, the disk would have to reserve editing/ updating space equal to the size of the source code -- a total of 488K. For comparison, Osborne microcomputer single-sided, double-density (SSDD) disks are limited to 183K usable storage space, while single-sided, single-density formats are only 92K. The attached software fills most of two Osborne-formatted, SSDD disks, leaving approximately 65K for generated files and partial editing space.

Modular Program Development

This technique simplified coding and allowed easier verification, debugging, validation, and later modification. It also eliminated a design limitation in the Borland TURBO Pascal package. Program sizes always exceeded the maximum size the Borland product could compile in memory at one time (approximately 20K, depending on code and data sizes). By

Average Winds Aloft Example

A. WINDS ALOFT DATA			B. VECTOR SOLUTION	C. PARACHUTE DRIFT
SPECIFIED ALTITUDE	EFFECTIVE RANGE	DIRECTION/VELOCITY	VECTOR VALUE	PARACHUTE OPENING POSITION
8000	9000	270/30	090/30	 1550 True, 2.3 Miles SURFACE POSITION
6000	7000	300/25	120/50	
4000	5000	330/25	150/50	
2000	3000	000/23	180/46	
SEA LEVEL	1000	045/44	225/44	
NOTE: THE VECTOR VALUES FOR 6000, 4000, AND 2000 REPRESENT A RANGE OF 2000 FT. EACH. THEREFORE, THE TOTAL NUMBER OF VECTORS FOR 1000 FT. INTERVALS IS 8.				

(PARACHUTE OPENED AT 8000 FEET OVER OCEAN)

chaining modules and compiling to disk instead of memory (options available in the Borland product), no complications arose from the compilation of programs with source code larger than available RAM (64K). This is because modules were individually-compiled to disk instead of simultaneously compiling in memory before recording on disk.

The three programs that make up the search planning package are: Aerospace Drift Determination, Surface Drift Determination, and Search Area Determination. These programs and their modules will now be discussed in more detail (Reader understanding of National SAR Manual search planning methodology is assumed).

Aerospace Drift Determination Program

The Aerospace Drift Determination program has three possible paths of flow (SAR object glides only, parachutes only, or, glides then parachutes to the surface). No matter how it reaches the surface, the search object is displaced from its initial incident position by the vector sum of various wind directions and velocities and different altitudes (see example in Figure 5).

The Aerospace Drift Determination program calls sub-modules in the order shown in Figures 6, 7, and 8.

```

|-AeroSpaceWinds Program
|->  |-AeroPosition
      |
      |->  VerifyDTG
      |
      |->  ChuteData
      |      (Drift Method = PARACHUTE ONLY; SEE BELOW)
      |
      |->  GlideData
      |      (Drift Method = GLIDE & PARACHUTE; SEE BELOW)
      |
      |->  |-GlideData
            |
            |->  |-AeroGlide
                  |
                  |->  |-InputWinds
                        |
                        |->  |-VerifyWinds
                        |
                        |->  WindChart
                        |
                        |->  |-WindDrift
                        |
                        |->  GlideOrChute
                        |
                        |->  AddVectors
                  |
                  |->  AddVectors (Optional)
      |
|->  WriteToDisk

```

Figure 6. FIRST PATH: DRIFT METHOD = GLIDE ONLY

```

|-AeroSpaceWinds Program
|-> |-AeroPosition
    |-> VerifyDTG
    |-> GlideData
        (Drift Method = GLIDE ONLY; SEE ABOVE)
    |-> GlideData
        (Drift Method = GLIDE & PARACHUTE; SEE BELOW)
    |-> |-ChuteData
        |-> |-InputWinds
            |-> |-VerifyWinds
            |-> WindChart
            |-> |-WindDrift
                |-> GlideOrChute
                |-> AddVectors
                |-> Ejection
                |-> AddVectors (Optional)
|-> WriteToDisk

```

Figure 7. SECOND PATH: DRIFT METHOD = PARACHUTE ONLY

```

|-AeroSpaceWinds Program
|->  |-AeroPosition
      |->  VerifyDTG
      |->  ChuteData
          (Drift Method = PARACHUTE ONLY; SEE ABOVE)
      |->  GlideData
          (Drift Method = GLIDE ONLY; SEE ABOVE)
      |->  |-GlideData
          |->  |-AeroGlide
              |->  |-InputWinds
                  |->  |-VerifyWinds
                      |->  WindChart
                  |->  |-WindDrift
                      |->  GlideThenChute
                      |->  AddVectors
              |->  AddVectors (Optional)
              |->  Ejection (Optional)
              |->  AddVectors (Optional)
              |->  |-WindDrift
                  |->  GlideOrChute
                  |->  AddVectors
                  |->  AddVectors (Optional)
|->  WriteToDisk

```

Figure 8. THIRD PATH: DRIFT METHOD = BOTH GLIDE & PARACHUTE

Aerospace Drift Program Module Descriptions

Aerospace Drift sub-modules accomplish the following tasks:

PROCEDURE -----	TASK -----
AddVectors	When given the X and Y components from procedure VectorComponents, this procedure calculates resultant vector bearing (AvgWindFrom) and magnitude (ResultMagnitude).
AeroGlide	Asks user for aerospace object glide information, if applicable.
AeroPosition	Asks user for last known time and position of an aerospace object.
ChuteData	Gathers additional information concerning the aerospace object's parachute descent before commencing calculations.
Ejection	If aerospace object(s) of interest is a (are) pilot(s) or astronaut(s) ejecting from a crippled vehicle, this procedure asks the user for the (judgmental) performance-type of vehicle. Then it sets the ejection distance equal to the standards for that type of vehicle as prescribed in the <u>National SAR Manual</u> (10:8-8).
GlideData	Gathers additional information concerning the aerospace object's glide before commencing calculations.
GlideOrChute	This procedure is used if the aerospace object only glides (or falls), or parachutes (after, or in lieu of, gliding) to the surface. It's purpose is to find the altitudes halfway between each altitude for which a wind direction and velocity have been reported. It begins with the next higher altitude winds reported above the surface level winds, and ends with that mid-altitude that is greater than or equal to the altitude at which the

descent began. It then uses these mid-altitudes to determine the effective wind component velocities on the descending aerospace object (10:8-10).

GlideThenChute

This procedure is used if the aerospace object glides (or falls) to a lower altitude at which parachute(s) open to slow it's descent to the surface. It's purpose is to find the altitudes halfway between each altitude for which a wind direction and velocity have been reported. It begins with the mid-altitude that is lower than or equal to the parachute(s) opening altitude, and ends with the mid-altitude that is higher than or equal to the altitude at which the descent began. It then uses these mid-altitudes to determine effective wind component velocities on the gliding (or falling) portion of the aerospace object's descent (10:8-10).

InputWinds

Queries user for altitudes at which the aerospace-object begins to fall or glide, or to deploy parachute(s). Then it asks for wind directions and velocities at those altitudes through which the object descends. The user is allowed to verify/change any input data before it is used in determining the resultant drift vector bearing and magnitude.

VerifyDTG

Verifies legitimate date/time/group data input.

VerifyWinds

Allows user to change wind altitudes, directions, and velocities, entered below in procedure InputWinds.

WindChart

Prints keyboard-input wind altitudes, directions, and velocities, on the video screen for user verification.

WindDrift

Determines the resultant drift vector bearing and magnitude.

WriteToDisk

Prints out record of search planning inputs and calculations from this program.

Surface Drift Determination Program

Over time, the search object is displaced from its initial surface/splash-down position by the vector sum of leeway, sea, tidal, and/or wind current directions and velocities at different times and locations (similar to the example presented in Figure 5 above). The Surface Drift Determination program calls sub-modules in the order shown in Figure 9.

```

|-Surface Drift Program
|->  |-SurfacePosition
      |->  VerifyDTG
      |->  VerifyDTG
|->  |-WindPeriods
|->  |-PeriodTimes
      |->  DaysInMonth (Optional #1)
      |->  DaysInMonth (Optional #2)
      |->  DaysInMonth (Optional #3)
|->  |-InputSeaWinds
|->  |-WindLatCoeffs
      |->  |-VerifyWinds
            |->  WindChart
            |->  WindChart (Optional)
|->  |-WindCurrent
      |->  AddVectors
      |->  AddVectors (Optional)
|->  |-AvgSurfaceWind
      |->  AddVectors
|->  |-LeewayDrift
      |->  DriftDirUncertain (Optional)
|->  |-SeaCurrent
|->  |-Datum
|->  |-WriteToDisk (If program used)
      |->  RecordCurrents

```

Figure 9. SURFACE DRIFT DETERMINATION PROGRAM FLOW CHART

Surface Drift Program Module Descriptions

Surface Drift Determination sub-modules accomplish the following tasks:

PROCEDURE	TASK
-----	-----
AddVectors	When given X and Y components this procedure calculates resultant vector bearing (AvgWindFrom) and magnitude (ResultMagnitude).
AvgSurfaceWind	Determines the average surface (leeway) wind blowing on the search object's exposed area above the ocean's surface.
Datum	Determines the Datum Drift Vector (Min and Max, if applicable) from the vector sum of all previous surface drift vectors.
DaysInMonth	Given the month, determines the number of days in that month, including a check to see if that month is a leap-year-February with 29 days.
DriftDirUncertain	Used by procedure Leeway (if drift rate and time are known with certainty) to calculate minimum and maximum leeway drift direction.
InputSeaWinds	Queries user for ocean surface wind directions and velocities.
LeewayDrift	Calculates the total leeway drift direction and distance (10:8-13 through 8-15).
PeriodTimes	Determines the day and hour for which wind directions and velocities are required in the eight intervals of each period to calculate each period's ocean surface wind current component vector (10:8-16b).
RecordCurrents	Called by procedure WriteToDisk; Prints out record of Wind and Sea Current vectors or Observed Total Water Current vector, as applicable.

SeaCurrent	Determines the sea (or slope) current affecting search object drift, and, queries user for total observed water current vector, if known.
SurfacePosition	Asks user for the last known and the desired Datum times and positions of a search object on the ocean's surface.
VerifyDTG	Verifies legitimate date/time/group data input.
VerifyWinds	Allows user to change ocean surface wind directions and velocities entered in procedure InputSeaWinds.
WindChart	Prints keyboard-input wind coefficient directions and velocities on the video screen for user verification.
WindCurrent	Uses vector addition to calculate the resultant wind current vector direction and magnitude for each period and for the sum of all period vectors.
WindLatCoeffs	Queries user for Wind Latitude Coefficient Vector directions and velocities from <u>National SAR Manual</u> tables (10:8-16c through 8-16d).
WindPeriods	Given the last known surface position time, and the hours elapsed from then until the desired Datum time, this procedure determines the number of 48-hour wind-current-effect periods, and the number of hours (1 to 6) in each period, required to calculate the average ocean surface current caused by the wind.
writeln	Writes out a specified number of blank lines.
WriteToDisk	Prints out record of search planning inputs and calculations from this program.

Search Area Determination Program

The Search Area Determination program calls sub-modules in the order shown in Figure 10.

```
| -Program: SearchArea
|
| -> | -Position
|
| -> | -AeroSurfVectors
|
| -> | -AreaSearch
|
| -> | -FindCoordinates
|
|     | -> | -NewCoordinates (Aerospace Drift Vector Pos.)
|     |
|     | -> | -NewCoordinates (Dmax Surface Drift Position)
|     |
|     | -> | -NewCoordinates (Dmin Surface Drift Position)
|     |
|     | -> | -NewCoordinates (Search Area Center Point)
|     |
|     | -> | -FindXYsToCorner
|     |
|         | -> NewCoordinates (Upper Right Corner Pos.)
|         |
|         | -> NewCoordinates (Lower Left Corner Pos.)
|
| -> | -WriteToDisk
```

Figure 10. Search Area Determination Program Flow Chart

Search Area Program Module Descriptions

Search Area Determination sub-modules accomplish the following tasks:

PROCEDURE

TASK

----- AreaSearch

Given the search object's aerospace and/or surface drift vector(s), this procedure calculates the search area.

AeroSurfVectors

Queries user for previously-calculated Aerospace and (Max & Min) Surface Drift vectors.

FindCoordinates

Calculates X & Ycomponents of aerospace and surface drift vectors and sends them to procedure NewCoordinates for it to determine the search area center point latitude/longitude. Next, this procedure creates two vectors (1 & 2), 90-degrees apart, and of length equal to the search area's radius. It sends these two vectors to procedure FindXYsToCorner for it to calculate the vectors' X & Ycomponents enroute to procedure NewCoordinates, which finds the corner point latitudes/longitudes.

FindXYsToCorner

Finds the X & Ycomponents of the search-area-center-point-displacement-vectors created by the calling procedure, FindCoordinates. Then, it passes these components to procedure NewCoordinates for it to determine search area corner point latitudes/longitudes.

NewCoordinates

When given X & Ycomponents and a reference latitude/longitude by the calling procedures FindCoordinates and FindXYsToCorner, this procedure calculates the updated position's latitude/longitude.

Position

Queries user for last known position latitude and longitude.

writeln

Writes out a specified number of blank lines.

WriteToDisk

Prints out record of error adjustments to calculations of search radius and area.

Chapter 3

Program Structure

Overview

Chapter 2 declared the attached programs are designed using current software engineering guidelines. This chapter discusses these guidelines and how they were used to make this search planning software accurate, maintainable, reliable, and "user-friendly" (for users knowledgeable in search planning methodology).

Software Engineering Guidelines

According to Ledgard in Pascal With Style: Programming Proverbs, the most important guideline in software development is the use of top-down design using modules:

Top-down programming has two distinct advantages. First, a programmer is initially freed from the confines of a particular language and can deal with more natural data structures or actions. Second, it leads to a modular approach that allows the programmer to write statements relevant to the current structures or actions. (3:13)

The modules are then thoroughly-tested before their combination into the larger program. The final step involves program verification (testing) to pinpoint module interface errors.

Testing & Error Checking

Unfortunately, program testing only reveals the presence, not the absence, of errors:

Too often quantity of test data is accepted in place of quality. The mere fact that a program works on 50 test runs means nothing if those 50 sets were not chosen carefully. . . (The) objective must be to select values that cause the execution of all flow paths through the program. (6:436)

In addition, the testing sequence should include boundary, null, and illegal case trials.

Boundary case errors are uncovered by the input of data which falls at the boundaries or extreme legal range allowed by the program. For example, a calendar-date program should be tested for accuracy in determining the day after the last day of different months, last day of the year, and last day of a leap year February (as is done by Procedure DaysInMonth in the attached Surface Drift program).

In contrast, testing for null case error involves such tricks as the input of blanks or carriage returns where data is requested by the program, or directing the program to reference an empty data file. Such actions should not cause an abrupt abort of the program run, although wrong answers will be generated. To avoid null errors, the attached programs are initialized with valid values, and, they often lock the user into a "repeat-until" loop until valid new data is input. Also, there are no calls to external data files, empty or otherwise.

Finally, programs must be checked for their "robustness" in handling illegal data entry. A robust

program produces meaningful results (although not always correct answers) from any input data set, even if that set is illegal or improper. Since the three programs created for this project request numerous data inputs from the user, input errors are a certainty. The type of errors expected are:

- (1) Accidentally typing the wrong character.
- (2) The input of a real number where an integer is requested.
- (3) The input of wind vector data out of sequence.
- (4) Transposing the input of a date-time-group (e.g., 1200-25Z 1/86 vs. 251200Z JAN 1986).

Improper values accepted by these programs leads to the "Garbage-In, Garbage-Out" (GIGO) syndrome. To prevent such an occurrence, "defensive programs" have been designed with extensive error-checking for legal and plausible data values. When data is requested from the user, format examples are provided. Then, user input is filtered through "repeat-until" loops to ensure adherence to required format. For example, the programs will not accept compass headings outside of the 0 to 360 degree range, winds greater than 99 knots, or selection of option #4 when only three choices are offered. In some cases, the programs will print an appropriate error message, explaining how to correct and resubmit the data.

Unfortunately, the design of the Pascal programming language allows erroneous input of real numbers where integers are called for. This action leads to an extremely abrupt program abort. All data must then be re-input from the beginning. However, since the search planning package was divided into three separate programs, data re-input and execution time of any one is usually less than a minute for experienced users with the required data readily-available.

Calculation Accuracy

Another problem over which there is no control is the machine representation and manipulation of real numbers. This leads to incorrect fractional values and rounding or truncation errors. For example, a computer having eight binary place accuracy records the value "1/5" as 0.00110011, or 0.1992 instead of 0.20. In this case, five times a fifth equals 0.996! Therefore, checking for equality of real values or successively multiplying or dividing them introduces errors during program execution. Steps were taken in designing the attached search planning programs to minimize opportunities for the occurrence of these errors.

Summary Of Other Design Features

Other design features of attached programs include:

- (1) Cleverness of expression was always sacrificed for expression clarity and simplicity.
- (2) Mnemonic variable names correspond with those

used by the USCG National SAR School, even at the expense of having some names 18 characters in length.

(3) Variable prefixes and suffixes are standardized (e.g., different types of wind's directions, velocities, or distances covered, always use the terms "Brg" (Bearing to), "Dir" (Direction from), "Spd" (Speed), or "Dist" (Distance) in their mnemonic name).

(4) Comments to assist program readability and modification are used extensively:

- Comments just prior to procedure declaration describe their purpose.
- All variables names and functions are comment-defined in the appropriate "Var" and "Const" sections.
- All "Begin" and "End" pairs are matched using comments.
- Any statement whose intent is not obvious is described.

(5) To enhance program clarity:

- Each line is numbered (non-standard for Pascal).
- Only one statement per line is allowed (except for variable value initialization).
- Multiline statements are indented so that structurally-related clauses are aligned.

(6) Following each of the three source codes in Appendix D are tables of the mnemonic variables and Pascal functional operators used by each program. These tables

also include a complete, cross-referenced listing of line numbers on which each variable and operator appears. These "Variable & Operator Cross-Referenced Listings" are provided to further enhance software understanding and aid modification efforts.

(7) All input wind altitudes, directions, and velocities are reprinted back to the user for verification and update, if necessary, before being used in program calculations. The only exception are those wind current wind times, directions, and velocities requested by the largest of the three attached programs -- Surface Drift Determination. Unfortunately, there was only enough microcomputer memory space remaining in the Surface Drift program to verify this data, or to verify the requested Wind Latitude Coefficient Vector directions and magnitudes. Since the Latitude Coefficient data is more complex, it appears the likeliest candidate for typographical input error. For this reason, it was selected to fill the remaining verification space. Therefore, the user must exercise caution to carefully input wind current wind data the first time. Any errors will require the user to terminate that program run and try again.

(8) The attached search planning programs request some input from the user that is not fully tested for error. An example is the query for the search object's last known position date-time-group (DTG) in the first program --

Aerospace Drift Determination. Although the DTG format is checked to confirm the date falls between 1 and 31 and the hour between 0000 and 2300, the date is not correlated with the month. For example, "311400Z Feb 1984" (or "2:00 P.M. 31 Feb 1984" -- 31 days in February!) is entirely acceptable by the program. Why? The DTG is not used in any calculations in this particular program, but only for final output file reporting purposes. Therefore, scarce microcomputer memory space was conserved by relying on the user to avoid logic or typographical input errors, while maintaining the his/her ability to recognize such simple errors, should they occur in the final output disk file.

(9) "Goto" statements were not used to avoid unconditional transfers of control thereby enhancing program readability.

(10) The use of global vs. local variables was abused in designing the attached programs. Why? Calling variables "by location" uses much less of the scarce microcomputer memory than "call by value" routines. Also, after all user input and computations are completed, most program variables must be written to an external record/file to serve as a "calculations-audit trail." Making these variables global allowed for a simple, "WriteToDisk" procedure to create the required record.

(11) All three attached programs take advantage of a non-standard Pascal feature of Turbo Pascal -- a built-in

command/procedure called "CLRSCR" for clearing the video screen whenever desired. For these programs to run properly under standard Pascal rules, these "CLRSCR" commands should be globally replaced with the command "writelns(24)". This command calls on the predefined procedure Writelns included in all three programs. The unit "24" informs this procedure to print 24 blank lines on the output device (monitor or printer), essentially clearing the screen just as "CLRSCR" (although not as briskly).

Validation/Verification

The goal of verification and validation was to test the programs to ensure they perform as expected, providing valid and correct output from legitimate input data. Verification of the attached programs was accomplished in two steps.

Individual modules were first tested and re-tested during the design phase to guarantee they accurately performed only those functions they were created to do. Input and output test data came from two scenarios provided by the Coast Guard. Once the modules passed these preliminary inspections, they were combined to form the overall program. This program was then subjected to additional testing to verify it still worked as desired -- grinding out identical answers to those supplied by USCG.

Confirming program "correctness" under a variety of test conditions, or "validation," was generously

accomplished by the Coast Guard. The attached programs were downloaded to their Convergent Technologies C-3 Data System mainframe computer system in New York in September of 1984. Then, National SAR School faculty members ran numerous school and real-world problems to test program limits and accuracy. The result -- in each case the search planning software performed flawlessly, generating answers identical to those expected.

Chapter 4

Epilogue

Conclusions

The initial intent of this thesis effort was to create fast, reliable, and accurate, microcomputer-based, search planning software to be used to maintain the skills of those SAR planners/National SAR School graduates patient enough to retype (and compile) the source code from free copies provided by the SAR School. From this meager beginning, the three programs generated by this effort developed a life of their own.

The great breakthrough came with the creation of an electronic pathway between the C-3 mainframe computer (used at almost every Coast Guard base) and the Osborne microcomputer during the author's visit to USCG Governor's Island, New York. This allowed transfer and subsequent compilation of the search planning software, with the inherent advantages of:

- (1) National SAR School faculty members had around-the-clock access on their own office terminals to the search planning software created by this thesis effort. This access enabled them to verify and validate this software, as well as make valuable modification recommendations before the final version(s) of the three programs were released.
- (2) Search planners/National SAR School graduates would not be forced to retype the +65-pages of source code (which

would certainly discourage widespread dissemination of the software). Instead, they could now download the source code directly to disk (in seconds) or via modem telecommunications to their microcomputers (in minutes).

(3) The first of the three attached programs, Aerospace Drift Determination, suddenly achieved great significance upon the discovery that the Coast Guard's commercially-procured, surface search planning software (CASP) lacks the crucial capability to determine aerospace drift (glide, fall, and/or parachute winds) vectors. USCG National SAR School faculty members spoke of using this program for more than just refresher training purposes.

The only unavoidable complaint directed against the attached software was actually a criticism of the programming language used. SAR School faculty members disliked the abrupt way in which programs aborted when a fatal Pascal error was input (e.g., input of a character where an integer was requested -- this particular problem was corrected by module redesign and a change of input variable type).

Recommendations For Further Research

None. This is not to assert the design of the attached software could not be improved. However, such embellishment could easily exceed scarce microcomputer memory limits, thus violating the prime objective of this entire effort.

The only feature which could be added to the attached package is search area "Effort Allocation" -- dividing the search area into sub-areas suited to the capabilities and duration of each available search craft and having the highest probability of containing the search object. However, optimal effort allocation involves the capability of weighted multi-criteria decision making and geographic map generation, both of which are better suited for mainframe computer implementation due to the large amounts of memory and processing required. In fact, the Coast Guard's mainframe CASP program does both in an effort allocation program far beyond the reach of current microcomputer capability (8).

Bibliography

1. "Cosmos Reentry Spurs Nuclear Waste Debate." Aviation Week & Space Technology Magazine, January 30, 1978, pp. 33.
2. Fraley, Major Theodore R. E., USAF, and Captain Dale A. Ken, USA. Fortran Based Linear Programming For Microcomputers, MS Thesis GOR/OS/82D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982 (AD-A124 804).
3. Ledgard, Henry F., John F. Hueras and Paul A. Nagin. Pascal With Style: Programming Proverbs. Rochelle Park, NJ: Hayden Book Co., Inc., 1979
4. Masterson, Fred A. "Languages For Students." Byte Magazine, June, 1984, pp. 233-238.
5. "NASA Abandons Skylab Control Plans." Aviation Week & Space Technology Magazine, January 1, 1979, pp. 19-20.
6. Schneider, G. Michael, and Steven C. Bruell. Advanced Programming And Problem Solving With Pascal. New York: John C. Wiley & Sons, Inc., 1981.
7. "Soviet Cosmos 954 With Nuclear Reactor Aboard Reenters Earth's Atmosphere January 24 - Disintegrates Over Northwestern Canada." New York Times, January 25, 1978, 1:6.
8. Telephone conversations, course materials, and personal interviews with faculty members of the United States Coast Guard National Search and Rescue School, USCG Governor's Island, New York, during the period Feb 84 through Oct 84.
9. "US-Canadian Air-Search Teams Discover Powerful Radiation In Area Where Satellite Reentered Atmosphere." New York Times, January 27, 1978, 1:5.
10. U.S., Department of Defense, Department of the Air Force. Air Force Manual 64-2*, National Search and Rescue Manual (*Also: USA FM 20-150, USN NWP-37(B), and USCG COMDINST M16130.2). Washington, D.C.: Government Printing Office, July 1, 1973, Amendments #1 through #7.
11. "Use of Pascal Language in Training of Computer Programmers." Sovetskaya Litva, No. 167 (12480), July 20, 1984.

Appendix A

DEFINITION OF TERMS

Glossary

AEROSPACE ASSET - Any object of value which operates in or passes through the Earth's atmosphere, e.g. aircraft, spacecraft, debris of same, pilots, astronauts, etc.

AEROSPACE DRIFT - The ground distance an aerospace asset could possibly cover during its descent.

AEROSPACE POSITION - The initial position of an aerospace asset at the time of reentry, engine failure, or aircrew ejection/bailout

BAILOUT TRAJECTORY - Momentum imparted in the direction of travel by an aircraft movement upon an airman who has ejected or bailed out.

CONFIDENCE FACTOR - Values ranging from 0.125 to 0.375 used to obtain drift error. Confidence factor is the search planners estimate of how reliable his/her information is in terms of initial distress position, sea current, wind current, and leeway. If one or more of these factors are poorly defined, confidence is low and a value of 0.30 is used. If all four factors are known with high certainty, a value of 0.125 is used.

CRASH - A landing in which the vertical velocity is so great and the time spent in reducing it to zero is so brief that the acceleration and hence the forces acting become so great as to result in structural failure.

DATUM - The probable location of the search object corrected for drift at any particular moment during the mission.

DATUM MINIMAX - The datum point established midway between the resultant minimum drift position and the resultant maximum drift position.

DEAD RECKONING - Determination of one's position by monitoring course and adding the additional displacement since passing last position. This displacement is calculated by multiplying one's speed by the time elapsed since passing last position.

DRIFT - The vectorial movement (direction and distance) of the search object caused by momentum, drag, wind, water, or other external forces.

HEADING - The horizontal direction in which an aerospace or surface asset is pointing/moving.

INDIVIDUAL DRIFT ERROR - Individual drift multiplied by Confidence Factor.

INITIAL POSITION ERROR - The assumed error of the initially reported position of a search and rescue incident.

KNOT - A measurement unit of speed equivalent to one nautical mile per hour.

LEEWAY - The movement of a search object caused by being pushed through the water by local winds acting against the exposed surface of the object. This motion is distinct from the effect of the wind on the surface current (see **WIND CURRENT** on next page).

LAST KNOWN POSITION - The position last witnessed, reported, or computed as a dead-reckoning position from a previously-reported and reliable position of the search object.

PARACHUTE DRIFT - The combined drift of parachute glide ratio and its displacement due to winds aloft while the parachute is descending.

SAFETY FACTOR - A factor used to increase Total Probable Error to a length that insures greater than a 50% probability that the object is in the calculated search area.

SEA CURRENT - 1. Current present in the open sea caused by factors other than local winds (combined with wind current to form total water current in oceanic areas); 2. The permanent, large scale flow of ocean waters, also called "slope current." Not usually computed for an incident of less than four hours of drift, unless the target is in a relatively high speed current area such as the Gulf Stream.

SEARCH AND RESCUE/RECOVERY - The employment of available personnel and facilities in rendering aid to persons and property in distress.

SEARCH AREA - That area calculated to contain the search object and subject to intensive scrutiny. Because there are few search patterns which are easily adaptable to a circular search area, a square is circumscribed around the first search circle. Thus a square search area with sides equal to twice the search radius and centered at datum is established.

SEARCH CRAFT ERROR - Error introduced by the less than 100% navigational accuracy of equipment installed aboard specific types of search craft.

SEARCH RADIUS - The radius of a circle centered on a datum point. It has a length equal to the Total Probable Error plus an additional safety length to insure greater than 50% probability that the object is in the search area.

SET - The compass direction towards which a current flows, or the direction an object moves under the influence of wind and/or current.

SURFACE DRIFT - The vector sum of the average sea current, wind driven current, and leeway.

SURFACE POSITION - The surface position (in latitude and longitude) of the search object on the Earth's surface.

TIDAL CURRENT - Water currents influenced predominantly by reversing, coastal rotary, or river currents, or their combinations.

TOTAL DRIFT ERROR - The sum of all individual drift errors during a certain time interval.

TOTAL PROBABLE ERROR - That error in datum within which it is assumed there is a 50% chance that the search object is located. It is the square root of the sum of the squares of the Total Drift Error, the Initial Position Error, and the Search Craft Error.

VECTOR - A quantity having both direction and magnitude.

WIND BEARING - The compass direction toward which a wind is blowing.

WIND CURRENT - The current generated by the wind acting upon the surface of the water for a period of time, affected by the wind's velocity and duration.

WIND DIRECTION - The compass direction from which a wind is blowing.

Acronyms

AFIT - Air Force Institute of Technology

AGL - Above Ground Level (altitude)

AFM - Air Force Manual

CASP - Computer-Assisted Search Planning
 COMDINST - USCG Commandant's Instructions (regulations)
 Dmax - maximum Drift distance
 Dmin - minimum Drift distance
 DTG - Date-Time-Group (e.g. "291000Z Jul" is equivalent to
 July 29th at 1000-hours zulu time.)
 ETA - Estimated Time (of) Arrival
 FM - (USA) Field Manual
 GIGO - "Garbage-In, Garbage-Out" (computer proverb)
 KM - Kilometer
 LKP - Last Known Position
 MINIMAX - Point halfway between maximum and minimum
 MSL - Mean Sea Level (altitude)
 NAUT - Nautical mile
 RAM - (Computer) Random Access Memory (space)
 SAR - Search and Rescue/Recovery
 SSDD - Single-Sided, Double-Density (magnetic disk)
 TAC - (USAF's) Tactical Air Command
 USA - United States Army
 USAF - United States Air Force
 USCG - United States Coast Guard
 USN - United States Navy
 Xcomponent - The horizontal component of a vector used in
 vector addition
 Ycomponent - The vertical component of a vector used in
 vector addition

Appendix B

SOFTWARE USER'S GUIDE

Overview

The purpose of this guide is to provide all the information a user needs to employ the attached aerospace asset search planning programs properly. The user must have a microcomputer equipped with a minimum of 64K RAM and one disk drive. The three programs generate input/output data record files during run-time which easily fit any standard 80-column-width printer at ten characters-per-inch (no "line wrap-around"). These search planning programs are recommended for training use only, especially if human life is at risk. Also, due to the possibility of undetected errors, user's should be totally familiar with search planning methodology before attempting to use this software (completion of the USCG National SAR School search planning course is highly-encouraged).

Software Purpose

The attached programs provide the user with the necessary automated tools to complete search area planning for missing persons or aerospace objects in the Earth's most complex drift environment -- the ocean. The first program calculates drift forces on a falling, gliding, or parachuting aerospace object to determine the displacement vector from its last known coordinates. The second program

calculates wind current, sea current, and leeway drift forces on a surface object to determine the maximum and minimum displacement vector from its last known position. Finally, the third program combines previously input data and computed vectors to determine the center and four corner points of the search area with the highest probability of containing the missing person(s) or object.

Data Required

To calculate these drift vectors and the search area, the programs request specific data inputs from the user. Each program requests different data with some duplication for continuity. A detailed description of the input data required for each of the three attached programs follows.

Aerospace Drift Program Inputs

In the Aerospace Drift Program the user will be asked for:

- (1) Last known aerospace position and time (month, year, date-time-group, latitude, and longitude);
- (2) Surface altitude or terrain height;
- (3) Altitude (Incident altitude) at which the object began it's glide, fall, and/or parachute deployment, as applicable;
- (4) Altitude lost during glide, fall, and/or parachute deployment, as applicable;
- (5) Object's glide ratio, descent heading and rate,

as applicable;

(6) Wind directions and velocities at every reported level up to incident altitude (plus one or two more above it) with the opportunity to verify or change the input before committing it to internal calculations;

(7) Parachute drift distance and chute descent rate from National SAR Manual tables, if applicable (10:8-11 through 8-12); and,

(8) Whether the aerospace object was a medium- or high-performance jet aircraft (as defined by the National SAR Manual) to determine pilot/crew ejection distance, if applicable (10:8-8).

Once the run is complete, the Aerospace Drift program creates an external text file ("calculation audit trail") named "AERODATA" detailing user inputs and program outputs.

Surface Drift Program Inputs

In the Surface Drift Program the user will be asked for:

(1) Last known surface position and time (month, year, date-time-group, latitude, and longitude);

(2) Desired (Datum) surface search time (month, year, date-time-group);

(3) To confirm the search object is more than 20 miles (32 km) off shore in water greater than 100 feet (32 m) in depth as prescribed for this search planning method by the National SAR Manual (10:8-16a);

(4) Approximate hours elapsed from last known position time to search initialization;

(5) Sea wind directions and velocities (reported at 0000, 0600, 1200, and 1800 Zulu time) for at least a 48-hour period from last known position time to search initialization time (User must exercise caution to input this particular data correctly the first time. It is not echoed back for user verification and update due to scarce microcomputer memory limits. If an input error is made, followed by a carriage return, the user must manually abort and rerun the program to obtain reliable calculations.);

(6) Wind latitude coefficient directions and magnitudes for the appropriate latitude from the National SAR Manual (10:8-16c through 8-16d) with the opportunity to verify or change the input before committing it to internal calculations;

(7) Choose from drift rate, drift time, or directional drift uncertainties. Then input minimum and maximum leeway drift speeds, minimum and maximum hours of drift, and/or the maximum expected drift divergence angle (from the National SAR Manual), as applicable (10:8-15);

(8) Sea current drift direction, velocity, and published source of information (10:8-16i through 8-16j);

(9) Total observed water current, if applicable;

and,

(10) Tidal current direction and velocity, if applicable (10:8-22).

Once the run is complete, the Surface Drift program creates an external text file ("calculation audit trail") named "SEADATA" detailing user inputs and program outputs.

Search Area Determination Program Inputs

In the Search Area (Determination) Program the user will be asked for:

(1) Last known surface coordinates (latitude and longitude);

(2) Previously-calculated total aerospace drift direction and distance, if applicable;

(3) Aerospace drift error confidence factor from the National SAR Manual (10:8-27);

(4) Sum of previously-calculated drift errors, if applicable;

(5) Previously-calculated minimum and maximum total surface drift direction and distance;

(6) Plotted distance between these surface drift positions;

(7) Surface drift error confidence factor from the National SAR Manual (10:8-27);

(8) Search object and search craft navigational fix and dead-reckoning errors from the National SAR Manual (10:8-29); and,

(9) Whether this is the first through the fifth search effort (to determine total probable error factor) (10:8-32a through 8-32b).

Once the run is complete, the Search Area program creates an external text file ("calculation audit trail") named "AREADATA" detailing user inputs and program outputs.

Software Maintenance Responsibility

The author makes no express or implied warranty of any kind with regard to the attached programs. This includes, but is not limited to the implied warranty of fitness for any particular purpose. The author shall not be liable for incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of these programs.

Published changes to the National Search and Rescue Manual or errors detected by users should be brought to the attention of the author:

Captain D. R. Douglas, USAF
2065 East 3300 South
Salt Lake City, UT 84109 ,

who will make every effort to update or correct the attached software and re-release it to public domain under a different version number. The attached "Microcomputer Application of Aerospace Asset Search Planning" programs are released as versions 1.0 or 1.1.

Appendix C

SAMPLE PROBLEM / SOFTWARE DEMONSTRATION

Overview

This section introduces a sample search planning problem from the USCG National Search and Rescue School. The data presented is then input into the attached programs and solutions generated to demonstrate software execution. Appearing in brackets () beside some of these program solutions are answers provided by the Coast Guard, as well as a percentage error difference between the two solutions due to the previously-described internal rounding errors, etc. . For example:

Program solution	USCG solution
-----	-----
X = 0.950	(X = 0.974; 2.464% error)
Y = 0.892	(Y = 0.887; 0.0056% error)

Sample Problem Introduction

Situation: At 1000Z, 29 April 1984 a distress call was received from an ASAT-equipped (anti-satellite weapon), Air Force F-15 jet fighter. The pilot advised he was ejecting at 10,000 feet AGL (above ground level) in position 39-15.0 North 64-30.0 West. At time of bailout, the pilot's high-performance aircraft was on an satellite interception heading of 360-degrees Magnetic (348-degrees True, accounting for magnetic variation of 12-degrees West at that location). According to the parent command (TAC), F-15

ejection seats are equipped with self-deploying, single-man, orange life rafts with built-in drogue (sea anchors).

The first available search craft, an Air Force HC-130H from the Aerospace Rescue and Recovery Service, will arrive on scene at approximately 291900Z Apr 1984 to begin the search. A Russian trawler has also signalled intentions to divert and provide assistance with an ETA (estimated time of arrival) in the incident area of 300600Z.

The following meteorological data has been obtained for the vicinity of 39-15N 64-30W:

291000Z UPPER WINDS FORECAST FROM WSFO, JFK, NEW YORK

Surface	070/15	(070-degrees at 15 knots)	
3000	120/30	18000	210/25
6000	130/25	24000	210/20
9000	170/48	30000	210/30
12000	165/52	34000	215/65

SURFACE WIND HISTORY/FORECAST FROM USAF AIR WEATHER SERVICE

270000Z	100/15	290000Z	040/25
0600Z	070/18	0600Z	065/20
1200Z	065/10	1200Z	074/20
1800Z	080/20	1800Z	077/18
280000Z	040/30	300000Z	090/10
0600Z	035/40	0600Z	090/20
1200Z	040/40	1200Z	065/15
1800Z	045/20	1800Z	050/20

USN Publication 1400-NA 6 for the approximate position shows sea current set = 071-degrees True at 1.1 knots.

Directional drift uncertainty is assumed with a leeway drift divergence of plus-or-minus 35-degrees (from the National SAR Manual).

Use the attached programs to determine:

- (1) Average winds aloft
- (2) Aerospace drift vector
- (3) Average Surface winds
- (4) Wind current vector
- (5) Leeway vector
- (6) Maximum & minimum datum vectors
- (7) Search area size
- (8) Search area center point coordinates
- (9) Search area corner point coordinates

Sample Problem Program Runs

```
<< RUN FIRST PROGRAM: A E R O S P A C E   D R I F T >>
/\ /\ /\ /\ /\ /\   N E W   S C R E E N   /\ /\ /\ /\ /\ /\
```

```
(*****)
(*      SEARCH PLANNING SOFTWARE (PROGRAM #1 OF 3)      *)
(* TITLE:      AERODRIF.COM (Aerospace Drift Algorithm) *)
(* VERSION:    1.1 for CP/M Operating System           *)
(* DATE WRITTEN: September 1984                       *)
(* LICENSE:    COPYRIGHT 1984   D. RICK DOUGLAS        *)
(*****)
```

The author makes no express or implied warranty of any kind with regard to this program material, including, but not limited to, the implied warranty of fitness for a particular purpose. The author shall not be liable for incidental or consequential damages in connection with or arising out of furnishing, use, or performance of this program. The reader MUST HAVE a solid understanding of search and rescue methodology before using this software in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, this program is recommended for training use only. Prior attendance at the United States Coast Guard's National SAR School is highly-encouraged.

```
(***** WARNING! *****)
(*      THIS SOFTWARE MAY BE FREELY-DISTRIBUTED PROVIDED NO FEE      *)
(*      IS CHARGED AND THIS COPYRIGHT NOTICE IS RETAINED            *)
(*****)
```

PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE

\\ \\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\ \\

This program calculates the initial surface position coordinates (latitude/longitude) of a falling, gliding, or parachuting aerospace object or person. If these coordinates are all ready known, or the search object did not fall at least 500 feet, then enter N to the next question and proceed with other search planning as outlined in the National Search and Rescue Manual.

Do you wish to continue with this program? (y/n) => Y

\\ \\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\ \\

Please enter the number for the month the search object was at the last known position:

1 = Jan	4 = Apr	7 = Jul	10 = Oct
2 = Feb	5 = May	8 = Aug	11 = Nov
3 = Mar	6 = Jun	9 = Sep	12 = Dec

MONTH=> 4

Please enter the year the search object was at the last known position (e.g., 1985, 1986, etc.)

YEAR=> 1984

Please enter the day-hour-minute the search object was at the last known position. Enter it in Z DTG (Zulu Date-Time-Group) format. For example: 9:37 PM, August 4, would appear as 042137 Greenwich-Mean Time (Z=0).

TIME (Z DTG)=> 291000

Was search object's last known latitude north or south?
(Enter N or S) Answer=> N

Please enter the search object's last known latitude
(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)
LATITUDE => 39.15

Was search object's last known longitude east or west?
(Enter E or W) Answer=> W

Please enter the search object's last known longitude
(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)
LONGITUDE => 64.3

Please enter the surface level altitude or terrain height (in feet) above or below mean sea level. Enter a zero if altitude equals sea level, or, enter a negative altitude if below sea level (e.g. Death Valley, California.).
SURFACE LEVEL/TERRAIN HEIGHT => 0

After receiving report of the search object's last known coordinates (lat/long), did the object fall or glide to a lower altitude before parachute(s) opened or ground contact was made? (y/n) => N

After receiving report of the search object's last known coordinates (lat/long), did the object successfully use parachutes to descend? (y/n) => Y

Approximate (to the nearest 500 feet) at what altitude above mean sea level (MSL) did the parachute(s) open? (Enter a zero if descent began below 500 feet AGL)
ALTITUDE=> 10000

\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\\\\\

PLEASE ENTER REPORTED OR FORECAST WIND DIRECTION AND VELOCITY FOR EACH KNOWN ALTITUDE WITHIN THE FOLLOWING CONSTRAINTS:

1. You must input the altitude, wind direction (0 to 360), and velocity (0 to 99), from surface level up to the altitude at which the power-off descent began or parachute(s) opened, whichever is higher. You will then be asked to enter one higher altitude for which wind data is known. Finally, you will be able to verify your input data before any computations are made.
2. Altitudes must be entered in rounded thousands of feet, e.g., 3000, 7000, 12000, not 3500, 5200, etc.
3. An example follows of input altitude/winds data format:

ALTITUDE = 0 (Surface level); Enter lowest altitude first!
WIND DIRECTION = 330
WIND VELOCITY = 25

HIT RETURN (Once or twice, as necessary) TO CONTINUE

\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

Begin altitude/wind data input starting with surface level winds:

ALTITUDE = 0
WIND DIRECTION = 70
WIND VELOCITY = 15

ALTITUDE = 3000
WIND DIRECTION = 120
WIND VELOCITY = 30

ALTITUDE = 6000
WIND DIRECTION = 130
WIND VELOCITY = 25

ALTITUDE = 9000
WIND DIRECTION = 170
WIND VELOCITY = 48

ALTITUDE = 12000
WIND DIRECTION = 165
WIND VELOCITY = 52

Enter data for just one more, higher altitude.

ALTITUDE = 18000
WIND DIRECTION = 210
WIND VELOCITY = 25

\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

1. Winds at 0 feet: 70 degrees at 15 knots
2. Winds at 3000 feet: 120 degrees at 30 knots
3. Winds at 6000 feet: 130 degrees at 25 knots
4. Winds at 9000 feet: 170 degrees at 48 knots
5. Winds at 12000 feet: 165 degrees at 52 knots
6. Winds at 18000 feet: 210 degrees at 25 knots

Are these altitudes, wind directions, and wind
velocities all correct? (y/n) T (INPUT ERROR (INTENTIONAL))
(CAUSES QUESTION TO REPEAT)

Are these altitudes, wind directions, and wind
velocities all correct? (y/n) Y

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Wind Data (Glide or Parachute Opening Altitude to surface level AGL)
With Velocity Components Calculated Prior To Vector Addition:

Winds at	18000 feet:	210 degrees at	0 knots
Winds at	12000 feet:	165 degrees at	0 knots
Winds at	9000 feet:	170 degrees at	120 knots
Winds at	6000 feet:	130 degrees at	75 knots
Winds at	3000 feet:	120 degrees at	105 knots
Winds at	0 feet:	70 degrees at	15 knots

Xcomponent = 183.319 (183.319; 0% ERROR)

Ycomponent = -213.756 (-213.756; 0% ERROR)

The average winds aloft affecting the parachute drift of
the aerospace object are bearing to 319 degrees at 28.2 knots.

Refer to the "Parachute Drift Table" in the National
Search and Rescue Manual (approximately page 8-12) and
interpolate parachute drift distance (in nautical miles)
from the published chart. For example: For a parachute
opening altitude of 3000 feet (915 meters) and winds aloft
averaging 15 knots, the parachute drift distance
interpolates to 0.675 nautical miles).

PARACHUTE DRIFT DISTANCE => 3.94

Refer to the "Parachute Descent Data Table" in the National
Search and Rescue Manual (approximately page 8-11) and
determine the parachute descent rate (in feet-per-second)
from the published chart. For example: A 28-foot (C-9)
escape parachute at 7000 feet lowers a person at a rate of
21.4 ft/sec, while the three, 83-ft.-diameter parachutes
on the Apollo space capsule allowed for a 32.5 ft/sec
descent at this same altitude.

PARACHUTE DESCENT RATE => 21.4

Please enter descent or bailout heading, if known
(Enter 361 if not known). HEADING (0-361) => 348

If aerospace object is a pilot ejecting from a
crippled aerospace vehicle, was that vehicle:

1. A turboprop or medium performance jet aircraft
2. A high-performance jet aircraft
3. Neither of the above

ENTER A 1, 2, or 3 => 1

Xcomponent = -2.669 (-2.671; 0.0075% ERROR)

Ycomponent = 3.480 (3.481; 0.0029% ERROR)

////////// N E W S C R E E N \\\\\\\

A record of significant input and output data used during this program run is stored in an external file named "AERODATA". If you desire to keep this record permanently, please rename file AERODATA before running this program again!

<< F I R S T P R O G R A M O U T P U T >>
////////// N E W S C R E E N \\\\\\\

Missing aerospace object/pilot(s) last known position: 39-15 North 64-30 West
Time: 291000Z 4/1984

Bailout/Parachute opening altitude	=	10000 feet MSL
Surface level or terrain height	=	0 feet MSL
Wind bearing affecting para-descent	=	319 degrees True
Wind velocity affecting para-descent	=	28.2 knots
Parachute table descent rate	=	21.4 feet per second
Parachute drift table distance	=	3.9 nautical miles
Ejection displacement	=	0.5 nautical miles
Total aerospace drift bearing	=	323 degrees True
Total aerospace drift distance	=	4.4 nautical miles
Total time of parachute descent	=	7.8 minutes

Wind Data Used To Calculate Above Results:

1. Winds at 0 feet: 70 degrees at 15 knots
2. Winds at 3000 feet: 120 degrees at 30 knots
3. Winds at 6000 feet: 130 degrees at 25 knots
4. Winds at 9000 feet: 170 degrees at 48 knots
5. Winds at 12000 feet: 165 degrees at 52 knots
6. Winds at 18000 feet: 210 degrees at 25 knots

```
<< RUN SECOND PROGRAM:  S U R F A C E  D R I F T  >>
/\/\/\/\/\/\/\  N E W  S C R E E N  \\/\/\/\/\/\/\
```

```
(*****
(*          SEARCH PLANNING SOFTWARE (PROGRAM #2 OF 3)          *)
(*  TITLE:          SURFDRIF.COM (Surface Drift Algorithm)      *)
(*  VERSION:        1.1 for CP/M Operating System              *)
(*  DATE WRITTEN:    September 1984                             *)
(*  LICENSE:        COPYRIGHT 1984      D. RICK DOUGLAS         *)
(*****
```

The author makes no express or implied warranty of any kind with regard to this program material, including, but not limited to, the implied warranty of fitness for a particular purpose. The author shall not be liable for incidental or consequential damages in connection with or arising out of furnishing, use, or performance of this program. The reader MUST HAVE a solid understanding of search and rescue methodology before using this software in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, this program is recommended for training use only. Prior attendance at the United States Coast Guard's National SAR School is highly-encouraged.

```
(***** WARNING! *****
(*  THIS SOFTWARE MAY BE FREELY-DISTRIBUTED PROVIDED NO FEE    *)
(*  IS CHARGED AND THIS COPYRIGHT NOTICE IS RETAINED         *)
(*****
```

PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE

```
/\/\/\/\/\/\/\  N E W  S C R E E N  \\/\/\/\/\/\/\
```

Given the initial surface position coordinates (latitude and longitude) of an object on the ocean's surface and starting date/time, this program calculates an updated surface position (Datum point) or search area for a specified (Datum) time. If the updated surface position coordinates are all ready known, then enter N to the next question and proceed with other recovery planning as outlined in the National Search and Rescue Manual.

P.S. Run this program TWICE if the number of hours of search object drift is uncertain. On the first run, enter data only for the SHORTER drift period (later drift start time). After the program calculates the Average Surface Wind and Wind Current vectors, record them, ABORT the program run, and CLEAR your microcomputer's memory. Then rerun the entire program, making sure to enter data for the LONGER drift period (earlier drift start time). Enter the vectors you recorded above when asked by the program.

Do you wish to continue with this program? (y/n) => Y

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Please enter the number for the month the search object was at the last known position:

1 = Jan	4 = Apr	7 = Jul	10 = Oct
2 = Feb	5 = May	8 = Aug	11 = Nov
3 = Mar	6 = Jun	9 = Sep	12 = Dec

LAST KNOWN POSITION MONTH=> 4

Please enter the year the search object was at the last known position (e.g., 1985, 1986, etc.)

LAST KNOWN POSITION YEAR=> 1984

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Please enter the day-hour-minute (TO THE NEAREST HOUR) the search object was at the last known position. Enter it in Z DTG (Zulu Date-Time-Group) format. For example: 9:37 PM, August 4, should appear as 042200 Greenwich-Mean Time (Z=0).

If the number of hours of search object drift is uncertain, you must run this program twice (as mentioned earlier). On the first run enter here the time the SHORTER drift period started (LATER than the last known position time). On the second run, or if two runs are not applicable, enter here the the LKP time as requested above:

LAST KNOWN POSITION TIME (Z DTG)=> 291000

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Was search object's last known latitude north or south?
(Enter N or S) Answer=> N

Please enter the search object's last known latitude
(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)
LATITUDE => 39.1848

Was search object's last known longitude east or west?
(Enter E or W) Answer=> W

Please enter the search object's last known longitude
(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)
LONGITUDE => 64.3345

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Now you will be asked to enter the year, month, and date/time for
the desired Datum position, which MUST be later than the time you
just entered for the last known position!

Please enter the desired Datum month:

1 = Jan	4 = Apr	7 = Jul	10 = Oct
2 = Feb	5 = May	8 = Aug	11 = Nov
3 = Mar	6 = Jun	9 = Sep	12 = Dec

DATUM MONTH=> 4

Please enter the desired Datum year (e.g., 1985, 1986, etc.)
DATUM YEAR=> 1984

Please enter the desired Datum day-hour-minute (TO THE NEAREST HOUR).
Enter it in Z DTG (Zulu Date-Time-Group) format. For example:
9:37 PM, August 4, should appear as 042200 Greenwich-Mean Time (Z=0).
DATUM TIME (Z DTG)=> 291800

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

This program includes wind current calculations to determine
Datum. However, according to the National SAR Manual, wind
currents are usually ignored in coastal, lake, river, and
harbor areas due to the many variable effects from the water-
land interface. This program is based on the assumption of
open-sea search where land masses do not interfere with the
action of the wind on the water or on the currents generated
by them. A rule of thumb is to calculate wind currents when
water depths are greater than 100 feet (32 meters) and at
distances of 20 miles (32 kilometers) or greater from shore.
Wind currents are not usually used inside these limits
except where local knowledge makes it possible to estimate
one. This is especially true where, close to shore, the
water's depth increases rapidly.

Do you wish to continue with this program? (y/n) => Y

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Please enter the number of hours elapsed (to the NEAREST HOUR) from the last known surface position (LKP) time to the desired Datum time.

If the number of hours of search object drift is uncertain, you must run this program twice (as mentioned earlier). On the first run enter here the SHORTER number of hours of search object drift. On the second run, or if two runs are not applicable, enter here the difference between the Datum and LKP times as requested above:

ELAPSED HOURS => 8

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Please enter the ocean surface wind direction and velocity at 271800Z:

WIND DIRECTION = 80

WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at 280000Z:

WIND DIRECTION = 40

WIND VELOCITY = 30

Please enter the ocean surface wind direction and velocity at 280600Z:

WIND DIRECTION = 35

WIND VELOCITY = 40

Please enter the ocean surface wind direction and velocity at 281200Z:

WIND DIRECTION = 40

WIND VELOCITY = 40

Please enter the ocean surface wind direction and velocity at 28 1800Z:

WIND DIRECTION = 45

WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at 290000Z:

WIND DIRECTION = 40

WIND VELOCITY = 25

Please enter the ocean surface wind direction and velocity at 290600Z:

WIND DIRECTION = 65

WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at 291200Z:
WIND DIRECTION = 74
WIND VELOCITY = 20

Please enter the ocean surface wind direction and velocity at 291800Z:
WIND DIRECTION = 77
WIND VELOCITY = 18

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Refer to the latitude coefficient-vectors table for the Northern latitudes (approximately page 8-16c) in the National SAR Manual. Find the degrees-latitude column nearest to 39-degrees-North. You will now be asked to enter the directions and magnitudes for each of the eight periods appearing under that column.

PERIOD #1 DIRECTION = 217
PERIOD #1 MAGNITUDE = .024

PERIOD #2 DIRECTION = 350
PERIOD #2 MAGNITUDE = .01

PERIOD #3 DIRECTION = 107
PERIOD #3 MAGNITUDE = .008

PERIOD #4 DIRECTION = 223
PERIOD #4 MAGNITUDE = .006

PERIOD #5 DIRECTION = 339
PERIOD #5 MAGNITUDE = .006

PERIOD #6 DIRECTION = 95
PERIOD #6 MAGNITUDE = .005

PERIOD #7 DIRECTION = 211
PERIOD #7 MAGNITUDE = .004

PERIOD #8 DIRECTION = 327
PERIOD #8 MAGNITUDE = .004

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Wind Current Latitude Coefficients for 39-degrees North are:

1.	Period #1	217 / 0.024
2.	Period #2	350 / 0.010
3.	Period #3	107 / 0.008
4.	Period #4	223 / 0.006
5.	Period #5	339 / 0.006
6.	Period #6	95 / 0.005
7.	Period #7	211 / 0.004
8.	Period #8	327 / 0.004

Are these altitudes, wind directions, and wind velocities all correct? (y/n) Y

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Period #1

Xcomponent = -0.119 (-0.119; 0% ERROR)

Ycomponent = 0.229 (0.229; 0% ERROR)

Period #2

Xcomponent = -0.283 (-0.283; 0% ERROR)

Ycomponent = 0.109 (0.109; 0% ERROR)

Total Wind Current:

Xcomponent = -1.443 (-1.454; 0.69% ERROR)

Ycomponent = 1.470 (1.466; 0.27% ERROR)

HIT RETURN (Once or twice, as necessary) TO CONTINUE

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Total Wind Current Direction is 316-degrees for 2.06 nautical miles.

If the number of hours of search object drift is uncertain, you must run this program twice (as mentioned earlier). On the first run, RECORD this Total Wind Current vector direction and distance for input during the second run.

If this is the second run, please now ENTER the previously-recorded Wind Current vector over the shorter drift period (If not applicable, enter a heading of 361):

WIND CURRENT DIRECTION (from Run #1) => 361

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

Average Surface Wind:

Xcomponent = 148.742 (148.742; 0x)

Ycomponent = 39.711 (39.711; 0x)

Average Surface Wind Direction 75-degrees True at 19.24 knots.

If the number of hours of search object drift is uncertain, you must run this program twice (as mentioned earlier). On the first run, RECORD this Average Surface Wind vector direction and speed for use in calculating the minimum leeway speed during the second program run. Then, ABORT this program run, CLEAR your microcomputer's memory, and RERUN the entire program

HIT RETURN (Once or twice, as necessary) TO CONTINUE

\\//\\//\\//\\// N E W S C R E E N \\//\\//\\//\\//

This program will now calculate the LEEWAY vector, or, that drift caused by average surface winds pushing on the exposed area of the search object. There are three leeway vector calculation options:

- (1) DRIFT RATE UNCERTAINTY - Used when the search object is unknown, or, when it is not known whether the object has deployed an anti-drift device (e.g., sea drogue). All drift is computed as a minimum and maximum distance downwind;
- (2) DRIFT TIME UNCERTAINTY - Used if the number of hours the search object has been drifting is not known for certain. All drift is computed as a minimum and maximum distance downwind;
- (3) DIRECTIONAL DRIFT UNCERTAINTY - Used if above options do not apply. However, drift is computed for a divergent bearing left and right of the downwind vector.

Refer to the National SAR Manual for additional information.

PLEASE SELECT OPTION #1, #2, or #3 => 3

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Refer to the Leeway Speed Graph in the National SAR Manual (approximately page 8-15). Please enter the search object's maximum expected degrees-divergence from the downwind vector:
MAX EXPECTED DIVERGENCE => 35

Refer to the Leeway Speed Formulae in the National SAR Manual (approximately page 8-13). Please enter the leeway speed at which the search object drifts:
LEEWAY SPEED => 0.842

Please enter the number of hours the search object drifted:
HOURS OF DRIFT => 8

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Please enter the sea current vector as described in the National SAR Manual (approximately pages 8-16i and 8-16j):
SEA CURRENT DIRECTION (SET) => 71

SEA CURRENT VELOCITY => 1.1

Which publication did you use as the source of your sea current data?:

- (1) Naval Oceanographic Office Spec. Pub. Series 4000, Surface Currents
- (2) Publication No. 700
- (3) Oceanographic Atlas
- (4) Atlas of Surface Currents
- (5) Pilot Charts
- (6) Other

SELECT ONLY ONE => 1

\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

Please enter the observed total water current, if known. This current is determined by observing the drift positions and times of surface debris, oil slicks, or, by inserting an electronic Datum Marker Buoy in the search area. This total water current vector replaces the surface wind and sea current vectors previously calculated by this program. (If unknown, enter a heading of 361):
TOTAL WATER CURRENT DIRECTION => 361

\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

If you have calculated a Tidal Current vector, please enter it here
(If not applicable, enter a heading of 361):
TIDAL CURRENT DIRECTION => 361

\\\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\\\\\

(Max) Datum Vector:
Xcomponent = 0.550
Ycomponent = 6.645

(Min) Datum Vector:
Xcomponent = 2.543
Ycomponent = -0.821

Total Surface Drift Dir. :	108-degrees	5-degrees
Total Surface Drift Dist.:	2.67 naut. mi.	6.67 naut. mi.

HIT RETURN (Once or twice, as necessary) TO CONTINUE

\\\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\\\\\

Calculating . . . Please stand by

\\\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\\\\\

A record of significant input and output data used during this
program run is stored in an external file named "SEADATA".
If you desire to keep this record permanently, please rename
file SEADATA before running this program again!

<< SECOND PROGRAM OUTPUT >>
 /\ /\ /\ /\ /\ /\ N E W S C R E E N /\ /\ /\ /\ /\ /\

Missing aerospace object/pilot(s) last known position: 39-18 North 64- 33 West
 Time: 291000Z 4/1984

=====

AVERAGE SURFACE WINDS:

Date-Time-Group	Hours	Wind	Contributions
291200Z	5	74 / 20	74 / 100
291800Z	3	77 / 18	77 / 54

Total Average Surface Wind Direction 075-degrees True at 19.24 knots

=====

LEEWAY DRIFT VECTOR CALCULATIONS: Directional Drift Uncertainty

Maximum Expected Divergence 35-degrees
 Drift Rate 0.842 knots
 Hours Of Drift 8
 Leeway Direction Min = 220-degrees Max = 290-degrees True
 Leeway Distance Min = 6.74 naut.mi. Max = 6.74 naut.mi.

=====

WIND CURRENT PERIOD #1

Interval	Date-Time-Group	Wind	Coefficients	Contributions
1	291200Z	74 / 20	217 / 0.024	291 / 0.480
2	290600Z	65 / 20	350 / 0.010	415 / 0.200
3	290000Z	40 / 25	107 / 0.008	147 / 0.200
4	281800Z	45 / 20	223 / 0.006	268 / 0.120
5	281200Z	40 / 40	339 / 0.006	379 / 0.240
6	280600Z	35 / 40	95 / 0.005	130 / 0.200
7	280000Z	40 / 30	211 / 0.004	251 / 0.120
8	271800Z	80 / 20	327 / 0.004	407 / 0.080

Local Wind Current This Period 333 / 0.258
 Number Of Hours In This Period 5
 Wind Current Vector This Period 333 / 1.289 nautical miles

WIND CURRENT PERIOD #2

Interval	Date-Time-Group	Wind	Coefficients	Contributions
1	291800Z	77 / 18	217 / 0.024	294 / 0.432
2	291200Z	74 / 20	350 / 0.010	424 / 0.200
3	290600Z	65 / 20	107 / 0.008	172 / 0.160
4	290000Z	40 / 25	223 / 0.006	263 / 0.150
5	281800Z	45 / 20	339 / 0.006	384 / 0.120
6	281200Z	40 / 40	95 / 0.005	135 / 0.200
7	280600Z	35 / 40	211 / 0.004	246 / 0.160
8	280000Z	40 / 30	327 / 0.004	367 / 0.120

Local Wind Current This Period 291 / 0.303
Number Of Hours In This Period 3
Wind Current Vector This Period 291 / 0.909 nautical miles

Total Wind Current Direction 316-degrees True
Total Wind Current Distance 2.06 nautical miles
=====

SEA CURRENT VECTOR:

Used Naval Oceanographic Office Spec. Pub. Series 4000, Surface Currents
Sea Current Direction (Set) 071-degrees True
Sea Current Drift Rate 1.10 knots
Sea Current Drift Distance: 8.80 knots
=====

TOTAL SURFACE DRIFT (Dmin & Dmax) VECTORS:

Tot.Surf.Drift Direction : 108-degrees 005-degrees True
Tot.Surf.Drift Distance : 2.67 naut. mi. 6.67 naut. mi.
=====

<< RUN THIRD PROGRAM: SEARCH AREA DETERMINATION>>
\\ \\ \\ \\ \\ \\ \\ N E W S C R E E N \\ \\ \\ \\ \\ \\ \\

(*****
(* SEARCH PLANNING SOFTWARE (PROGRAM #3 OF 3) *)
(* TITLE: AREA.COM (Search Area Determination Algorithm) *)
(* VERSION: 1.0 for CP/M Operating System *)
(* DATE WRITTEN: August 1984 *)
(* LICENSE: COPYRIGHT 1984 D. RICK DOUGLAS *)
(*****

The author makes no express or implied warranty of any kind with regard to this program material, including, but not limited to, the implied warranty of fitness for a particular purpose. The author shall not be liable for incidental or consequential damages in connection with or arising out of furnishing, use, or performance of this program. The reader MUST HAVE a solid understanding of search and rescue methodology before using this software in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, this program is recommended for training use only. Prior attendance at the United States Coast Guard's National SAR School is highly-encouraged.

(***** WARNING! *****
(* THIS SOFTWARE MAY BE FREELY-DISTRIBUTED PROVIDED NO FEE *)
(* IS CHARGED AND THIS COPYRIGHT NOTICE IS RETAINED *)
(*****

PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE

\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

Was search object's last known latitude north or south?
(Enter N or S) Answer=> N

Please enter the search object's last known latitude
(For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)
LATITUDE => 39.15

Was search object's last known longitude east or west?
(Enter E or W) Answer=> W

Please enter the search object's last known longitude
(For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)
LONGITUDE => 64.3

\\\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\\\\\

Did you previously calculate an aerospace drift vector
for this problem? (y/n): Y

Please enter the previously-calculated,
TOTAL AEROSPACE DRIFT DIRECTION => 323

Please enter the previously-calculated
TOTAL AEROSPACE DRIFT DISTANCE => 4.39 (4.39; 0% ERROR)

Refer to the "Individual Drift Error" section in the National SAR
Manual (approximately page 8-27). Please enter the desired
Aerospace Drift Error Confidence Factor (e.g, 0.125, 0.3, etc.):
AEROSPACE DRIFT ERROR CONFIDENCE FACTOR => 0.3

\\\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\\\\\

Please enter the sum of previous drift errors
(Enter a zero if not applicable):
SUM OF DRIFT ERRORS => 0

Please enter the previously-calculated,
MINIMUM TOTAL SURFACE DRIFT DIRECTION => 109

Please enter the previously-calculated,
MINIMUM TOTAL SURFACE DRIFT DISTANCE => 2.69 (2.67; 0.74% ERROR)

Please enter the previously-calculated,
MAXIMUM TOTAL SURFACE DRIFT DIRECTION => 005

Please enter the previously-calculated,
MAXIMUM TOTAL SURFACE DRIFT DISTANCE => 6.63 (6.67; 0.6% ERROR)

Please enter the measured distance between the two, previously-
calculated, Total Surface Drift Distance positions:
DISTANCE BETWEEN SURFACE DRIFT POSITIONS => 7.73

Refer to the "Individual Drift Error" section in the National SAR
Manual (approximately page 8-27). Please enter the desired
Surface Drift Error Confidence Factor (e.g, 0.125, 0.3, etc.):
SURFACE DRIFT ERROR CONFIDENCE FACTOR => 0.3

\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

Refer to the "Initial Position Error" section in the National SAR
Manual (approximately page 8-29). Please enter the search object's
Navigational Fix Error (e.g, 0, 5, 10, or 15 naut. mi. radius),
and, if applicable, the Navigational Dead-Reckoning (DR) Error
(e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):
SEARCH OBJECT NAVIGATIONAL FIX ERROR => 10

SEARCH OBJECT NAVIGATIONAL DR ERROR => 0

\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

Refer to the "Search Craft Error" section in the National SAR
Manual (approximately page 8-29). Please enter the search craft's
Navigational Fix Error (e.g, 0, 5, 10, or 15 naut. mi. radius),
and, if applicable, the Navigational Dead-Reckoning (DR) Error
(e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):
SEARCH CRAFT NAVIGATIONAL FIX ERROR => 5

SEARCH CRAFT NAVIGATIONAL DR ERROR => 0

\\\\\\\\\\\\\\\\\\\\ N E W S C R E E N \\\\\\\\\\\\\\\\\\\

You are calculating the search area for:

- 1 The first search
- 2 The second search
- 3 The third search
- 4 The fourth search
- 5 The fifth search

Input SEARCH NUMBER => 1

PRELIMINARY INFORMATION:

Search Area Radius = 15 naut. mi.
Search Area = 900 sq-naut.mi. (900; 0% ERROR)

HIT RETURN (Once or twice, as necessary) TO CONTINUE

\/\/\/\/\/\/\/\/ N E W S C R E E N \\/\/\/\/\/\/\/\/

Calculating . . . Please stand by

\/\/\/\/\/\/\/\/ N E W S C R E E N \\/\/\/\/\/\/\/\/

A record of significant input and output data used during this program run is stored in an external file named "AREADATA". If you desire to keep this record permanently, please rename file AREADATA before running this program again!

<< T H I R D P R O G R A M O U T P U T >>
\/\/\/\/\/\/\/\/ N E W S C R E E N \\/\/\/\/\/\/\/\/

DRIFT COORDINATES : LATITUDE LONGITUDE
Last Known Position : 39.1500 64.3000
Aerospace Drift Position : 39.1830 64.3325
Surface Dmin Position : 39.1738 64.3008
Surface Dmax Position : 39.2507 64.3240

=====

SEARCH AREA:

Aerospace Drift Distance : 4.39 naut. mi.
Drift Error Confidence : 0.300
Aerospace Drift Error : 1.32 naut. mi.

Sum of Previous Drift Errors : 0.00 naut. mi.
Surface Drift Distance : 2.69 naut. mi. 6.63 naut. mi.
Drift Error Confidence : 0.300
Drift Error Min and Max : 0.81 naut. mi. 1.99 naut. mi.
Distance Between Dmin/max : 7.73 naut. mi.
Surface Drift Error Minimax : 5.26 naut. mi.

Total Drift Error	:	6.58 naut. mi.

Object Navigation Fix Error	:	10 naut. mi.
Object Dead-Reckn Error	:	0.00 naut. mi.
Object Total Position Error	:	10.00 naut. mi.

Searcher Navigation Fix Error	:	5 naut. mi.
Searcher Dead-Reckn Error	:	0.00 naut. mi.
Searcher Total Position Error	:	5.00 naut. mi.

Total Probable Error	:	12.97 naut. mi.
Safety Factor	:	1.1

Search Radius	:	15 naut. mi.
Search Area	:	900 naut. mi.-squared.

SEARCH AREA COORDINATES:	:	LATITUDE LONGITUDE
Center Point	:	39.2414 64.2923
Corner Point (Upper Left)	:	39.3908 64.4834
Corner Point (Lower Left)	:	39.0908 64.4834
Corner Point (Upper Right)	:	39.3908 64.0945
Corner Point (Lower Right)	:	39.0908 64.0945
=====		

Conclusion

The only significant hardware calculation errors introduced in the sample USCG problem are those in determining the search area center and corner point coordinates. Inaccuracies here were as large as 6 minutes of latitude (6 nautical miles) and 2 minutes of longitude. However, drift vector error discrepancies were miniscule to non-existent. As such, the attached programs can be used to generate these vectors, which can then be plotted on navigation charts to more accurately determine drift position coordinates during search planning.

Appendix D

SEARCH PLANNING PROGRAMS LISTING (SOURCE CODE)

```

0001 (*****
0002 (*)
0003 (*)      SEARCH PLANNING SOFTWARE (PROGRAM #1 OF 3)
0004 (*)
0005 (*)  TITLE:      AERODRIF.COM (Aerospace Drift Algorithm)
0006 (*)  VERSION:    1.1 for CP/M Operating System
0007 (*)  DATE WRITTEN: September 1984
0008 (*)
0009 (*)  DESCRIPTION:
0010 (*)    - User asked for last known position (coordinates) and
0011 (*)      date-time-group of search object
0012 (*)    - User queried as to whether aerospace object glided,
0013 (*)      fell, parachuted, or some combination, to get to the
0014 (*)      surface
0015 (*)    - Depending upon the descent method, user is then asked
0016 (*)      additional information as specified in the National
0017 (*)      Search and Rescue (SAR) Manual
0018 (*)    - User asked to input reported winds aloft for every
0019 (*)      known altitude from surface to the search object's
0020 (*)      incident level, plus one or two more above that
0021 (*)    - Wind data is printed back to the user for verification
0022 (*)    - User may make changes to input data as required
0023 (*)    - Program calculates applicable, resultant drift vector
0024 (*)      direction(s) and magnitude(s) and creates an "audit
0025 (*)      trail"/record file of program input, significant
0026 (*)      calculations, and output, named "AERODATA"
0027 (*)
0028 (*)  LICENSE:  COPYRIGHT 1984      D. RICK DOUGLAS
0029 (*)
0030 (*)  The author makes no express or implied warranty of any
0031 (*)  kind with regard to this program material, including, but
0032 (*)  not limited to, the implied warranty of fitness for a
0033 (*)  particular purpose. The author shall not be liable for
0034 (*)  incidental or consequential damages in connection with or
0035 (*)  arising out of furnishing, use, or performance of this
0036 (*)  program. The reader MUST HAVE a solid understanding of
0037 (*)  search and rescue methodology before using this software
0038 (*)  in making decisions where human life is at risk. In fact,
0039 (*)  since no amount of testing can uncover 100% of program
0040 (*)  errors, this program is recommended for training use or
0041 (*)  Prior attendance at the United States Coast Guard's
0042 (*)  National SAR School is highly-encouraged.
0043 (*)
0044 (*)      THIS SOFTWARE MAY BE FREELY-DISTRIBUTED
0045 (*)      PROVIDED NO FEE IS CHARGED AND
0046 (*)      THIS COPYRIGHT NOTICE IS RETAINED.
0047 (*)

```

```

0048 (* LANGUAGE: PASCAL *)
0049 (* USED : Borland International, TURBO.PAS, Version 2.0 *)
0050 (* ----- *)
0051 (* MODULES CALLED (Sequentially listed); (OPT) = "Optional": *)
0052 (* ----- *)
0053 (* AeroPosition *)
0054 (* VerifyDTG *)
0055 (* ----- *)
0056 (* PATH #1 PATH #2 PATH #3 *)
0057 (* GLIDE ONLY PARACHUTE ONLY BOTH METHODS *)
0058 (* | | | *)
0059 (* GlideData ChuteData GlideData *)
0060 (* AeroGlide InputWinds AeroGlide *)
0061 (* InputWinds VerifyWinds InputWinds *)
0062 (* VerifyWinds WindChart VerifyWinds *)
0063 (* WindChart WindDrift WindChart *)
0064 (* WindDrift GlideOrChute WindDrift *)
0065 (* GlideOrChute AddVectors GlideThenChute *)
0066 (* AddVectors Ejection AddVectors *)
0067 (* AddVectors (OPT) AddVectors (OPT) AddVectors (OPT) *)
0068 (* WriteToDisk WriteToDisk Ejection *)
0069 (* AddVectors (OPT) *)
0070 (* WindDrift *)
0071 (* GlideOrChute *)
0072 (* AddVectors *)
0073 (* AddVectors (OPT) *)
0074 (* WriteToDisk *)
0075 (* ----- *)
0076 (*****
0077
0078 program AeroSpaceWinds (input,output);
0079
0080 const radians = 57.2957795; {Standard radian conversion factor}
0081
0082 type Winds = array[1..20,1..3] of real;
0083
0084 var OnePassCompleted : boolean;
0085     {Flags second pass through procedure WindDrift}
0086
0087 chute,           {Determines if Drift procedure should be used}
0088 continue,       {Determines if this program should be used}
0089 descent,        {Determines if AeroGlide procedure should be used}
0090 LatNS,          {Indicates whether latitude is North or South}
0091 LongEW,         {Indicates whether longitude is East or West}
0092 MethodDrift : char; {Indicates parachute(P), glide(G), or both(B) }
0093
0094 GlideRatio,     {Horizont./Vert. displacement (from flight manual) }
0095 I,J,            {Used as counters}
0096 N,              {Tracks number of wind altitudes used by program}
0097 LastMonth,      {Month object at last known position}
0098 LastYear,       {Year object at last known position}
0099 TypeVehicle : integer; {Determines if vehicle is fast or slow}

```

```

0100
0101 AssignedAlt, {Incident altitude above mean sea level (MSL)}
0102 AvgWindFrom, {Average winds aloft direction from (in degrees) }
0103 AvgWindTo, {AvgWindFrom plus-or-minus 180}
0104 BrgGlideWinds, {Average winds aloft glide bearing in degrees}
0105 BrgMaxGlide, {Maximum glide course in degrees}
0106 BrgParaDrift, {Average winds aloft bearing on parachute drift}
0107 BrgRads, {Individual bearings converted to radians}
0108 ChuteAlt, {Determines parachute(s) opening altitude}
0109 DescentHeading, {Aerospace object's descent heading in decimal}
0110 DescentRate, {Aerospace object's descent rate in feet per minute}
0111 DescentTime, {Object's descent/glide time through GlideAltLost}
0112 DistanceParaDrift, {Parachute drift distance}
0113 EjectDistance, {Distance ejecting pilot is tossed horizontally}
0114 GlideAltLost, {Altitude lost falling from AssignedAlt}
0115 GlideDistance, {Glide ratio distance covered excluding winds}
0116 GlideWindSpeed, {Average winds aloft velocity affecting glide}
0117 IncidentAlt, {Parachute-opening altitude}
0118 LastLatitudeKnown, {Last known latitude of aerospace object}
0119 LastLongitudeKnown, {Last known longitude of aerospace object}
0120 LastDateTime, {Time aerospace object was at last known position}
0121 MaxDistanceGlide, {In nautical miles}
0122 MaxRadiusGlide, {Maximum radius of glide if BrgMaxGlide uncertain}
0123 ParaDriftSpeed, {Average winds aloft parachute drift velocity}
0124 RateParaDescent, {Object's parachute descent rate in feet per second}
0125 ResultMagnitude, {Resultant distance computed from vector addition}
0126 Temp, {Confirms WindDrift IncidentAlt/500 remainder = 0}
0127 TerrainHeight, {Height of surface at or above mean sea level}
0128 TimeParaDescent, {Object's parachute descent time in minutes}
0129 TotalBrg, {Total aerospace drift vector direction}
0130 TotalDistance, {Total aerospace drift distance}
0131 TotalRadius, {Total possible area radius if direction unknown}
0132 TrajectBrg, {Aerospace trajectory direction}
0133 TrajectDistance, {Aerospace trajectory distance}
0134 TrajectRadius, {Aerospace trajectory radius (TrajectBrg uncertain)}
0135 WindDisplacement : real; {Distance covered due to winds}
0136
0137 No, { 'N,n' = "No" characters}
0138 Yes, { 'Y,y' = "Yes" characters}
0139 ValidAnswers : set of char; { 'Y,y,N,n' = valid "Yes" and "No" characters}
0140
0141 AeroWinds, {Duplicate array of WindsAloft for computation use}
0142 WindsAloft : Winds; {Individual-altitude winds affecting parachute drift}
0143
0144 {*****}
0145 {Writes out a specified number of blank lines.}
0146
0147 procedure writelns (lines : integer);
0148
0149 var count : integer;
0150
0151 begin

```

```

0152     count := 0;
0153     while lines > count do
0154         begin
0155             writeln;
0156             count := count + 1
0157         end
0158     end;
0159
0160 {*****}
0161 {Verifies legitimate date/time/group data input.}
0162
0163 procedure VerifyDTG;
0164
0165 var Temp,           {Used as a temporary computation variable}
0166     TempDate,       {Used as temporary date variable}
0167     TempHour,        {Used as temporary hour variable}
0168     TempMinutes : real; {Used as temporary minutes variable}
0169
0170 begin {VerifyDTG}
0171     repeat {until correct date/time format input}
0172
0173         {Verify the day is between 1 and 31}
0174         TempDate := trunc( LastDateTime/10000 );
0175         if (TempDate < 1) or (TempDate > 31) then
0176             begin {TempDate not between 1 and 31}
0177                 writeln;
0178                 writeln('You have incorrectly entered the date. Try again!');
0179                 write('Re-enter Z DTG= ');
0180                 readln(LastDateTime)
0181             end; {TempDate not between 1 and 31}
0182
0183         {Verify the hour is between 0000 and 2400}
0184         Temp := (LastDateTime/10000 - (trunc(LastDateTime/10000)) ) * 100;
0185         TempHour := trunc(Temp);
0186         if (TempHour < 0) or (TempHour > 23) then
0187             begin {TempHour not between 0 and 23}
0188                 writeln;
0189                 writeln('You have incorrectly entered the hour. Try again!');
0190                 write('Re-enter Z DTG= ');
0191                 readln(LastDateTime)
0192             end; {TempHour not between 0 and 23}
0193
0194         {Verify the minute is between 0 and 60}
0195         Temp := (LastDateTime/100 - (trunc(LastDateTime/100)) ) * 100;
0196         TempMinutes := trunc(Temp);
0197         if (TempMinutes < 0) or (TempMinutes > 60) then
0198             begin {TempMinutes not between 0 and 60}
0199                 writeln;
0200                 writeln('You have incorrectly entered the minutes. Try again!');
0201                 write('Re-enter Z DTG= ');
0202                 readln(LastDateTime)
0203             end {TempMinutes not between 0 and 60}

```

```

0204
0205     until (TempDate >= 1) and (TempDate <= 31) and (TempHour >= 0) and
0206         (TempHour <= 23) and (TempMinutes >= 0) and (TempMinutes <= 60)
0207
0208 end; (VerifyDTG)
0209
0210 {*****}
0211 {Asks user for last known time and position of an aerospace object.}
0212
0213 procedure AeroPosition;
0214
0215 begin (AeroPosition)
0216
0217     writeln;
0218     repeat {until valid response}
0219         writeln('Please enter the number for the month the search object was');
0220         writeln('at the last known position:');
0221         writeln;
0222         writeln(' 1 = Jan          4 = Apr          7 = Jul          10 = Oct');
0223         writeln(' 2 = Feb          5 = May          8 = Aug          11 = Nov');
0224         writeln(' 3 = Mar          6 = Jun          9 = Sep          12 = Dec');
0225         writeln;
0226         write('MONTH= ');
0227         readln(LastMonth);
0228         if (LastMonth < 1) or (LastMonth > 12) then
0229             writeln('Incorrect number entered. Please try again!');
0230         writeln
0231     until (LastMonth >= 1) and (LastMonth <= 12);
0232
0233     writeln('Please enter the year the search object was at the last known');
0234     writeln('position (e.g., 1985, 1986, etc.)');
0235     write('YEAR= ');
0236     readln(LastYear);
0237     writeln;
0238
0239     writeln;
0240     writeln('Please enter the day-hour-minute the search object was at the');
0241     writeln('last known position. Enter it in Z DTG (Zulu Date-Time-Group)');
0242     writeln('format. For example: 9:37 PM, August 4, would appear as');
0243     writeln('042137 Greenwich-Mean Time (Z=0).');
0244     write('TIME (Z DTG)= ');
0245     readln(LastDateTime);
0246     VerifyDTG;
0247     writeln;
0248
0249     repeat {until valid latitude}
0250         writeln('Was search object's last known latitude north or south?');
0251         write(' (Enter N or S) Answer= ');
0252         readln(LatNS);
0253         writeln
0254     until (LatNS = 'N') or (LatNS = 'n') or (LatNS = 'S') or (LatNS = 's');
0255

```

```

0256 repeat (until valid latitude)
0257     writeln('Please enter the search object''s last known latitude ');
0258     writeln(' (For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)');
0259     write('LATITUDE = ');
0260     readln(LastLatitudeKnown);
0261     if (LastLatitudeKnown < 0) or (LastLatitudeKnown > 90) then
0262         writeln('Input latitude must be between 0-90. Try again!');
0263     writeln
0264     until (LastLatitudeKnown >= 0) and (LastLatitudeKnown <= 90);
0265
0266 repeat (until valid longitude)
0267     writeln('Was search object''s last known longitude east or west?');
0268     write(' (Enter E or W)      Answer= ');
0269     readln(LongEW);
0270     writeln
0271     until (LongEW = 'E') or (LongEW = 'e') or (LongEW = 'W') or (LongEW = 'w');
0272
0273 repeat (until valid longitude)
0274     writeln('Please enter the search object''s last known longitude ');
0275     writeln(' (For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)');
0276     write('LONGITUDE = ');
0277     readln(LastLongitudeKnown);
0278     if (LastLongitudeKnown < 0) or (LastLongitudeKnown > 180) then
0279         writeln('Input longitude must be between 0-180. Try again!');
0280     writeln
0281     until (LastLongitudeKnown >= 0) and (LastLongitudeKnown <= 180);
0282     writeln
0283
0284 end; {AeroPosition}
0285
0286 {*****}
0287 {When given the X and Ycomponents from procedure VectorComponents, this}
0288 {procedure calculates resultant vector bearing (AvgWindFrom) and }
0289 {magnitude (ResultMagnitude). }
0290
0291 procedure AddVectors (var Xcomponent, Ycomponent : real);
0292
0293 var TempAngle : real; {Temporary calculation variable}
0294
0295 begin {AddVectors}
0296
0297     {Find resultant angle (in degrees) uncorrected for compass position.}
0298     TempAngle := arctan(Xcomponent/Ycomponent) * radians;
0299
0300     {Find bearing resulting from TempAngle corrected for compass position.}
0301     if (Xcomponent > 0) and (Ycomponent > 0) then AvgWindFrom := TempAngle;
0302     if ((Xcomponent < 0) and (Ycomponent < 0)) or
0303         ((Xcomponent < 0) and (Ycomponent > 0)) then
0304         if (TempAngle + 180) > 360 then
0305             AvgWindFrom := TempAngle - 180
0306         else AvgWindFrom := TempAngle + 180;
0307     if (Xcomponent < 0) and (Ycomponent > 0) then

```

```

0308     if (TempAngle + 360) > 360 then
0309         AvgWindFrom := TempAngle - 360
0310     else AvgWindFrom := TempAngle + 360;
0311
0312     {Convert AvgWindFrom to AvgWindTo bearing.}
0313     if (AvgWindFrom + 180) > 360 then
0314         AvgWindTo := AvgWindFrom - 180
0315     else AvgWindTo := AvgWindFrom + 180;
0316
0317     {Resultant bearing magnitude is the square root of the sum of}
0318     {the squared components.}
0319     ResultMagnitude := sqrt(Xcomponent * Xcomponent + Ycomponent * Ycomponent);
0320
0321     writeln('Xcomponent = ', Xcomponent:5:3);
0322     writeln('Ycomponent = ', Ycomponent:5:3);
0323     writeln
0324 end; {AddVectors}
0325
0326 {*****}
0327 {This procedure is used if the aerospace object only glides (or falls), or }
0328 {parachutes (after, or in lieu of, gliding) to the surface. Its purpose is}
0329 {to find the altitudes halfway between each altitude for which a wind }
0330 {direction and velocity have been reported. It begins with the next higher }
0331 {altitude winds reported above the surface level winds, and ends with that }
0332 {mid-altitude that is greater than or equal to the altitude at which the }
0333 {descent began. It then uses these mid-altitudes to determine the affective }
0334 {wind component velocities on the descending aerospace object. }
0335
0336 procedure GlideOrChute (var IncidentAlt: real);
0337
0338 var I          : integer; {Used as counters}
0339     Temp1,      {Used in wind component velocity calculations}
0340     Temp2       : real;   {Used in wind component velocity calculations}
0341     LowAltitude, {Determines lower range altitude for wind component}
0342     HighAltitude {Determines upper range altitude for wind component}
0343                : array[1..20] of real;
0344
0345 begin {GlideOrChute}
0346     {Initialize variables}
0347     for I := 1 to 20 do HighAltitude[I] := 0.0; Temp1 := 0.0;
0348     for I := 1 to 20 do LowAltitude[I] := 0.0; Temp2 := 0.0;
0349
0350     {Calculate wind components for each applicable altitude.}
0351     {If IncidentAlt = 500ft. then halve the wind velocity component.}
0352     if IncidentAlt = 500.0 then AeroWinds[1,3] := AeroWinds[1,3] * 0.5;
0353     {If IncidentAlt = 1000ft., then wind velocity component remains unchanged.}
0354
0355     if N > 1 then {Confirms parachute opened at or above 1500 feet}
0356     begin {N > 1 loop}
0357         HighAltitude[1] := 1000.0;
0358
0359         {Start calculations for altitudes above 1000-ft}

```

```

0360      I := 2;
0361      repeat
0362
0363          {HighAltitude = altitude halfway between present one and next}
0364          {higher one, unless HighAltitude is greater than IncidentAlt.}
0365          HighAltitude[I] := (AeroWinds[I+1,1] - AeroWinds[I,1])
0366                               /2 + AeroWinds[I,1];
0367          {LowAltitude = previous HighAltitude.}
0368          LowAltitude[I] := HighAltitude[I-1];
0369
0370          if IncidentAlt >= HighAltitude[I] then
0371              begin {IncidentAlt >= HighAltitude[I]}
0372                  Temp1 := (HighAltitude[I] - LowAltitude[I]) / 1000.0;
0373                  AeroWinds[I,3] := AeroWinds[I,3] * Temp1;
0374                  I := I + 1;
0375
0376                  if IncidentAlt = HighAltitude[I] then
0377                      begin {IncidentAlt = HighAltitude[I]}
0378                          {Make next two altitude's wind components = 0}
0379                          AeroWinds[I+1,3] := 0.0; AeroWinds[I+2,3] := 0.0;
0380                          I := N {Exit I = 2 to N loop}
0381                      end {IncidentAlt = HighAltitude[I]}
0382                  end {IncidentAlt >= HighAltitude[I]}
0383              else
0384                  begin {IncidentAlt < HighAltitude[I]}
0385                      Temp2 := (IncidentAlt - LowAltitude[I]) / 1000.0;
0386                      AeroWinds[I,3] := AeroWinds[I,3] * Temp2;
0387
0388                      {Make next two altitude's wind components = 0}
0389                      AeroWinds[I+1,3] := 0.0; AeroWinds[I+2,3] := 0.0;
0390                      I := N {Exit I = 2 to N loop}
0391                  end {IncidentAlt < HighAltitude[I]}
0392          until I = N;
0393
0394          write('Wind Data (Glide or Parachute Opening Altitude to');
0395          writeln(' surface level AGL)');
0396          writeln('With Velocity Components Calculated Prior To Vector Addition:');
0397          writeln;
0398          for I := N downto 1 do
0399              begin {for I = 1 to number of individual wind vectors}
0400                  write(' Winds at ', AeroWinds[I,1]:6:0, ' feet: ');
0401                  write(AeroWinds[I,2]:4:0, ' degrees at ', AeroWinds[I,3]:3:0);
0402                  writeln(' knots')
0403              end; {for I = 1 to number of individual wind vectors}
0404          writeln
0405          end {N } 1 loop}
0406      end; {GlideOrChute}
0407
0408      {*****}
0409      {This procedure is used if the aerospace object glides (or falls) to a lower }
0410      {altitude at which parachute(s) open to slow it's descent to the surface. }
0411      {It's purpose is to find the altitudes halfway between each altitude for }

```

```

0412 {which a wind direction and velocity have been reported. It begins with the }
0413 {mid-altitude that is lower than or equal to the parachute(s) opening }
0414 {altitude, and ends with the mid-altitude that is higher than or equal to the}
0415 {altitude at which the descent began. It then uses these mid-altitudes to }
0416 {determine affective wind component velocities on the gliding (or falling) }
0417 {portion of the aerospace object's descent. }
0418
0419 procedure GlideThenChute (var IncidentAlt: real);
0420
0421 var I,H,L : integer; {Used as counters}
0422     Temp1, {Used in wind component velocity calculations}
0423     Temp2 : real; {Used in wind component velocity calculations}
0424     LowAltitude, {Determines lower range altitude for wind component}
0425     HighAltitude {Determines upper range altitude for wind component}
0426             : array[1..20] of real;
0427
0428 begin {GlideThenChute}
0429     {Initialize variables}
0430     for I := 1 to 20 do HighAltitude[I] := 0.0; Temp1 := 0.0;
0431     for I := 1 to 20 do LowAltitude[I] := 0.0; Temp2 := 0.0;
0432
0433     {Searches for where the ChuteAlt fits in between AeroWinds array}
0434     {altitudes, starting at surface level.}
0435     I := 1; {Surface level}
0436
0437     repeat
0438         if ChuteAlt > AeroWinds[I,1] then I := I + 1
0439     until ChuteAlt <= AeroWinds[I,1];
0440
0441     {Determines midpoints between AeroWinds array wind altitudes. }
0442     {Then inserts ChuteAlt and IncidentAlt to calculate wind component}
0443     {for each reported altitude's actual velocity range of altitudes }
0444     {between midpoints. }
0445
0446     {First, find the midpoint between AeroWinds array altitudes }
0447     {between which ChuteAlt is located.}
0448     LowAltitude[I] := ( AeroWinds[I,1]-AeroWinds[I-1,1] ) / 2 ) +
0449                     AeroWinds[I-1,1];
0450
0451     {Next, find the midpoint between the next given AeroWinds array }
0452     {altitude (above ChuteAlt) and the next altitude above that one. }
0453     HighAltitude[I] := ( AeroWinds[I+1,1]-AeroWinds[I,1] ) / 2 ) +
0454                     AeroWinds[I,1];
0455
0456     {If ChuteAlt is at or above the midpoint (LowAltitude), but below }
0457     {the next given AeroWinds array altitude, then the velocity }
0458     {component = the given altitude - ChuteAlt all divided by 1000. }
0459     if ChuteAlt >= LowAltitude[I] then
0460     begin
0461         N := I; {Initialize N at AeroWinds array altitude above midpoint}
0462         Temp1 := (AeroWinds[I,1] - ChuteAlt) / 1000
0463     end

```

```

0464 else
0465 {If ChuteAlt is below the midpoint (LowAltitude), but above the }
0466 {last given AeroWinds array altitude, then the velocity component }
0467 {equals the midpoint - ChuteAlt all divided by 1000. }
0468 begin
0469     N := I-1; {Initialize N at AeroWinds array altitude below midpoint}
0470     Temp1 := 0; {Ensures Temp1 has no effect on below calculations}
0471     AeroWinds[I-1,3] := ( (LowAltitude[I] - ChuteAlt) / 1000) *
0472                                     AeroWinds[I-1,3]
0473 end;
0474 L := N;
0475
0476 {If IncidentAlt is at or above the midpoint between the AeroWinds }
0477 {array altitude and the next higher one, then the velocity }
0478 {component equals the midpoint - the given altitude, all divided }
0479 {by 1000. }
0480 if IncidentAlt >= HighAltitude[I] then
0481     Temp2 := (HighAltitude[I] - AeroWinds[I,1]) / 1000
0482 else
0483 {If IncidentAlt is below the midpoint between the AeroWinds array }
0484 {altitude and the next higher one, then the velocity component }
0485 {equals the IncidentAlt - the given altitude all divided by 1000. }
0486 Temp2 := (IncidentAlt - AeroWinds[I,1]) / 1000;
0487
0488 {Total wind component effect equals the sum of component effects }
0489 {above and below the midpoint between AeroWinds array wind }
0490 {altitudes times the given altitudes velocity. }
0491 AeroWinds[I,3] := (Temp1 + Temp2) * AeroWinds[I,3];
0492
0493 repeat {until valid response}
0494     if IncidentAlt > HighAltitude[I] then
0495         begin {IncidentAlt > HighAltitude[I]}
0496             I := I + 1;
0497             {HighAltitude = altitude halfway between present one and next}
0498             {higher one, unless HighAltitude is greater than IncidentAlt.}
0499             HighAltitude[I] := (AeroWinds[I+1,1] - AeroWinds[I,1])
0500                                     /2 + AeroWinds[I,1];
0501             if IncidentAlt < HighAltitude[I] then HighAltitude[I] := IncidentAlt;
0502
0503             {LowAltitude = previous HighAltitude.}
0504             LowAltitude[I] := HighAltitude[I-1];
0505
0506             if IncidentAlt > HighAltitude[I] then
0507                 begin {IncidentAlt > to present altitude limit.}
0508                     Temp1 := (HighAltitude[I] - LowAltitude[I]) / 1000.0;
0509                     AeroWinds[I,3] := AeroWinds[I,3] * Temp1
0510                 end {IncidentAlt > to present altitude limit.}
0511             else
0512                 begin {IncidentAlt < to present altitude limit.}
0513                     Temp2 := (IncidentAlt - LowAltitude[I]) / 1000.0;
0514                     AeroWinds[I,3] := AeroWinds[I,3] * Temp2;
0515                     H := I;

```

```

0516
0517         {Zero out final altitude/wind components if next-to-last}
0518         {altitude = IncidentAlt.}
0519         if AeroWinds[N-1,3] = IncidentAlt then AeroWinds[I+1,3] := 0
0520         end {IncidentAlt < to present altitude limit.}
0521         end {IncidentAlt > HighAltitude[I]}
0522     until IncidentAlt <= HighAltitude[I];
0523
0524     writeln('Wind Data (Glide to Parachute Opening Altitude AGL) With');
0525     writeln('Velocity Components Calculated Prior To Vector Addition');
0526     writeln;
0527     for I := H downto L do
0528     begin {for I = number of individual wind vectors}
0529         write(' Winds at ',AeroWinds[I,1]:6:0,' feet: ');
0530         write(AeroWinds[I,2]:4:0,' degrees at ',AeroWinds[I,3]:3:0);
0531         writeln(' knots')
0532     end; {for I = number of individual wind vectors}
0533     writeln
0534
0535 end; {GlideThenChute}
0536
0537 {*****}
0538 {If aerospace object(s) of interest is a (are) pilot(s) or astronaut(s) }
0539 {ejecting from a crippled vehicle, this procedure asks the user for }
0540 {(judgmental) performance-type of vehicle. Then it sets the ejection }
0541 {distance equal to the standards for that type of vehicle as prescribed in }
0542 {the National SAR Manual. }
0543
0544 procedure Ejection;
0545
0546 begin {Ejection}
0547
0548     repeat {until valid response}
0549     writeln('If aerospace object is a pilot ejecting from a');
0550     writeln('crippled aerospace vehicle, was that vehicle:');
0551     writeln;
0552     writeln('1. A turboprop or medium performance jet aircraft');
0553     writeln('2. A high-performance jet aircraft');
0554     writeln('3. Neither of the above');
0555     writeln;
0556     write('ENTER A 1, 2, or 3 = ');
0557     readln(TypeVehicle);
0558     writeln
0559     until (TypeVehicle < 0) or (TypeVehicle > 4);
0560
0561     if TypeVehicle = 1 then EjectDistance := 0.5;
0562     if TypeVehicle = 2 then EjectDistance := 0.8;
0563     if TypeVehicle = 3 then EjectDistance := 0.0
0564
0565 end; {Ejection}
0566
0567 {*****}

```

```

0568 {Determines the resultant drift vector bearing and magnitude.}
0569
0570 procedure WindDrift (var IncidentAlt: real);
0571
0572 var I,J      : integer; {Used as counters}
0573     BrgRads,  {Individual bearings converted to radians}
0574     TempAlt,  {Used for interim IncidentAlt calculations}
0575     Xcomponent, {Horizontal component of individual vectors}
0576     Ycomponent : real; {Vertical component of individual vectors}
0577
0578 begin {WindDrift}
0579     {Make duplicate array of WindsAloft for computation purposes}
0580     for I := 1 to N do for J := 1 to 3 do AeroWinds[I,J] := WindsAloft[I,J];
0581
0582     {Call procedure GlideOrChute if aerospace object uses no parachute(s) to }
0583     {descend to surface level. }
0584     if ( (AssignedAlt-GlideAltLost) = TerrainHeight) and (MethodDrift = 'G') then
0585         GlideOrChute(IncidentAlt);
0586
0587     {Call procedure GlideOrChute if aerospace object uses only parachute(s) to}
0588     {descend to surface level (no glide or free-fall involved). }
0589     if MethodDrift = 'P' then GlideOrChute(IncidentAlt);
0590
0591     {Ensures the following instructions are used only for the program's }
0592     {initial pass through procedure WindDrift. }
0593     if OnePassCompleted = false then
0594
0595         {If aerospace object glides or falls to a lower altitude before using }
0596         {parachutes to descend to surface level, then call procedure }
0597         {GlideThenChute for the "fall or glide portion" of the descent. }
0598         if (AssignedAlt - GlideAltLost) > 500.0 then GlideThenChute(IncidentAlt);
0599
0600     {At this point, individual wind-component-contributions exist in the}
0601     {cells of AeroWinds array; We now calculate the resultant magnitude }
0602     {and velocity of these vectors using procedure AddVectors. }
0603     Xcomponent := 0.0; Ycomponent := 0.0; {re-initialize}
0604     for I := 1 to (N - 1) do
0605     begin {calculate Xcomponent and Ycomponent sums}
0606         {Convert all vector's individual bearings into radian units.}
0607         BrgRads := AeroWinds[I,2] / radians;
0608
0609         {Total all AeroWinds individual X and Y vector components.}
0610         Xcomponent := Xcomponent + sin(BrgRads) * AeroWinds[I,3];
0611         Ycomponent := Ycomponent + cos(BrgRads) * AeroWinds[I,3]
0612     end; {calculate Xcomponent and Ycomponent sums}
0613
0614     AddVectors(Xcomponent,Ycomponent);
0615
0616     if MethodDrift = 'G' then
0617     begin {Gliding or falling object}
0618         BrgGlideWinds := AvgWindTo;
0619         TempAlt := GlideAltLost / 1000;

```

AD-A151 833

MICROCOMPUTER APPLICATION OF AEROSPACE ASSET SURFACE

2/2

SEARCH PLANNING(U) AIR FORCE INST OF TECH

WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

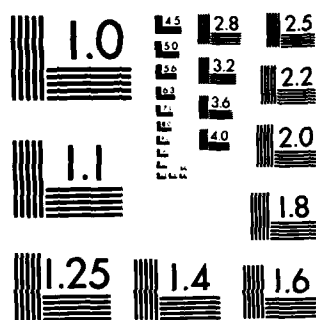
UNCLASSIFIED

D R DOUGLAS 14 DEC 84 AFIT/GSO/MATH/84D-1

F/G 6/7

NL

									END				



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

0620     GlideWindSpeed := ResultMagnitude /TempAlt;
0621 end; {Gliding or falling object}
0622
0623 if MethodDrift = 'P' then
0624 begin {Aerospace object uses parachute(s) descent}
0625     BrgParaDrift := AvgWindTo;
0626     TempAlt := IncidentAlt / 1000;
0627     ParaDriftSpeed := ResultMagnitude /TempAlt;
0628     writeln('The average winds aloft affecting the parachute drift of');
0629     write('the aerospace object are bearing to ',BrgParaDrift:3:0);
0630     writeln(' degrees at ',ParaDriftSpeed:3:1,' knots. ');
0631     writeln;
0632     writeln('Refer to the "Parachute Drift Table" in the National');
0633     writeln('Search and Rescue Manual (approximately page 8-12) and');
0634     writeln('interpolate parachute drift distance (in nautical miles)');
0635     writeln('from the published chart. For example: For a parachute');
0636     writeln('opening altitude of 3000 feet (915 meters) and winds aloft');
0637     writeln('averaging 15 knots, the parachute drift distance');
0638     writeln('interpolates to 0.675 nautical miles. ');
0639     write('PARACHUTE DRIFT DISTANCE = ');
0640     readln(DistanceParaDrift);
0641     writeln;
0642
0643     writeln('Refer to the "Parachute Descent Data Table" in the National');
0644     writeln('Search and Rescue Manual (approximately page 8-11) and');
0645     writeln('determine the parachute descent rate (in feet-per-second)');
0646     writeln('from the published chart. For example: A 28-foot (C-9)');
0647     writeln('escape parachute at 7000 feet lowers a person at a rate of');
0648     writeln('21.4 ft/sec, while the three, 83-ft.-diameter parachutes');
0649     writeln('on the Apollo space capsule allowed for a 32.5 ft/sec');
0650     writeln('descent at this same altitude. ');
0651     write('PARACHUTE DESCENT RATE = ');
0652     readln(RateParaDescent);
0653     writeln;
0654
0655     {Calculate parachute descent rate in minutes}
0656     TimeParaDescent := IncidentAlt / (RateParaDescent*60);
0657
0658     if OnePassCompleted = false then
0659     begin {procedure AeroSlide has not been used by program}
0660         repeat {until valid compass heading}
0661             writeln('Please enter descent or bailout heading, if known');
0662             write(' (Enter 361 if not known). HEADING (0-361) = ');
0663             readln(DescentHeading);
0664             if (DescentHeading < 0) or (DescentHeading > 361) then
0665                 writeln('This heading must be between 0-361. Try again!');
0666             writeln;
0667             until (DescentHeading >= 0) and (DescentHeading <= 361);
0668             Ejection;
0669
0670             {Finds maximum aerospace displacement vector}
0671             BrgRads := BrgParaDrift / radians;

```

```

0672      Xcomponent := sin(BrgRads) * DistanceParaDrift;
0673      Ycomponent := cos(BrgRads) * DistanceParaDrift;
0674
0675      if (DescentHeading < 361) and (EjectDistance > 0.0) then
0676      begin {descent heading known and EjectDistance significant}
0677          BrgRads := DescentHeading / radians;
0678          Xcomponent := Xcomponent + sin(BrgRads) * EjectDistance;
0679          Ycomponent := Ycomponent + cos(BrgRads) * EjectDistance;
0680          AddVectors(Xcomponent, Ycomponent);
0681          TotalBrg := AvgWindFrom;
0682          TotalDistance := ResultMagnitude
0683      end {descent heading known and EjectDistance significant}
0684      else
0685      begin {descent heading and EjectDistance not both significant}
0686          TotalBrg := BrgParaDrift;
0687          TotalDistance := DistanceParaDrift;
0688          if DescentHeading = 361 then
0689              if EjectDistance > 0.0 then
0690                  TotalRadius := EjectDistance
0691          end {descent heading and EjectDistance not both significant}
0692      end; {procedure AeroGlide has not been used by program}
0693
0694      if OnePassCompleted = true then
0695      begin {Adding parachute data to glide descent data for total vector.}
0696          {Finds total aerospace displacement vector}
0697          BrgRads := BrgParaDrift / radians;
0698          Xcomponent := sin(BrgRads) * DistanceParaDrift;
0699          Ycomponent := cos(BrgRads) * DistanceParaDrift;
0700          BrgRads := TrajectBrg / radians;
0701          Xcomponent := Xcomponent + sin(BrgRads) * TrajectDistance;
0702          Ycomponent := Ycomponent + cos(BrgRads) * TrajectDistance;
0703          AddVectors(Xcomponent, Ycomponent);
0704          TotalBrg := AvgWindFrom;
0705          TotalDistance := ResultMagnitude;
0706          TotalRadius := TrajectRadius
0707      end {Adding parachute data to glide descent data for total vector.}
0708      end; {Aerospace object uses parachute(s) descent}
0709
0710      OnePassCompleted := true
0711      end; {WindDrift}
0712
0713      {*****}
0714      {Prints keyboard-input wind altitudes, directions, and velocities, on the}
0715      {video screen for user verification.}
0716
0717      procedure WindChart;
0718
0719      var I : integer; {Used as a counter}
0720
0721      begin {WindChart}
0722          for I := 1 to N do
0723              begin {for I = 1 to number of individual wind vectors}

```

```

0724     write(1:2,'. Winds at ',WindsAloft[I,1]:6:0,' feet: ');
0725     write(WindsAloft[I,2]:4:0,' degrees at ',WindsAloft[I,3]:3:0);
0726     writeln(' knots')
0727     end {for I = 1 to number of individual wind vectors}
0728 end; {WindChart}
0729
0730 {*****}
0731 {Allows user to change wind altitudes, directions, and velocities, entered}
0732 {below in procedure InputWinds.}
0733
0734 procedure VerifyWinds;
0735
0736 var WindCheck : char; {Used to verify input wind data}
0737     J : integer; {Used as a counter}
0738     WindError : integer; {Used to indicate which input wind data is in error}
0739
0740 begin {VerifyWinds}
0741     WindCheck := 'Y'; WindError := -1; {initialize}
0742
0743 {Verify input wind directions and velocities.}
0744     WindChart; {Prints current wind data on video screen}
0745
0746     repeat {until WindCheck = valid response}
0747         writeln;
0748         writeln('Are these altitudes, wind directions, and wind');
0749         write('velocities all correct? (y/n) ');
0750         readln(WindCheck);
0751         writeln
0752     until (WindCheck in ValidAnswers);
0753
0754     while (WindCheck in No) do
0755     begin {while WindCheck = NO}
0756         repeat {until WindError = 0 to 20}
0757             writeln('Which line is in error?');
0758             write('Enter number, or zero for none) = ');
0759             readln(WindError);
0760             writeln
0761         until (WindError = 0.0) and (WindError < 21.0);
0762
0763         if WindError > 0.0 then
0764         begin {if an input line is identified as being in error}
0765             write('Please enter altitude, wind direction ');
0766             writeln('and velocity to replace:');
0767             write(WindsAloft[WindError,1]:4:0,' ');
0768             write(WindsAloft[WindError,2]:4:0);
0769             write(' ',WindsAloft[WindError,3]:4:0,' = ');
0770             for J := 1 to 3 do read(WindsAloft[WindError,J]);
0771             writeln(2)
0772         end; {if an input line is identified as being in error}
0773
0774         WindChart; {Prints current wind data on video screen}
0775

```

```

0776      writeln;
0777      repeat {until valid WindCheck response}
0778          write('Are all other lines correct? (y/n) ');
0779          readln(WindCheck);
0780          writeln
0781      until (WindCheck in ValidAnswers)
0782
0783      end; {while WindCheck = NO}
0784      CLRSCR
0785  end; {VerifyWinds}
0786
0787  {*****}
0788  {Queries user for altitudes at which the aerospace-object begins to fall or }
0789  {glide, or to deploy parachute(s). Then it asks for wind directions and }
0790  {velocities at those altitudes through which the object descends. The user }
0791  {is allowed to verify/change any input data before it is used in determining }
0792  {the resultant drift vector bearing and magnitude by procedure ParaDrift. }
0793
0794  procedure InputWinds (var IncidentAlt : real);
0795
0796  var I,                {Used as counter}
0797      P      : integer; {Tracks # of input wind altitudes above IncidentAlt}
0798      Temp2   : real;   {Confirms input wind altitudes are multiples of 1000}
0799
0800  begin {InputWinds}
0801      P := 0; {Initialize}
0802      CLRSCR;
0803
0804      {Starts bulk of winds aloft computations if IncidentAlt >= 500ft.}
0805      if IncidentAlt >= 500.0 then
0806          begin {winds aloft computations}
0807
0808              {Begin entry of individual wind components for applicable altitudes.}
0809              writeln;
0810              writeln('PLEASE ENTER REPORTED OR FORECAST WIND DIRECTION AND VELOCITY');
0811              writeln('FOR EACH KNOWN ALTITUDE WITHIN THE FOLLOWING CONSTRAINTS:');
0812              writeln;
0813              writeln('1. You must input the altitude, wind direction (0 to 360), and');
0814              writeln('    velocity (0 to 99), from surface level up to the altitude at');
0815              writeln('    which the power-off descent began or parachute(s) opened,');
0816              writeln('    whichever is higher. You will then be asked to enter one');
0817              writeln('    higher altitude for which wind data is known. Finally, you');
0818              writeln('    will be able to verify your input data before any');
0819              writeln('    computations are made.');

```

```

0828      writeln(' WIND DIRECTION = 330');
0829      writeln(' WIND VELOCITY = 25');
0830      writeln;
0831      write('HIT RETURN (Once or twice, as necessary) TO CONTINUE');
0832      readln(continue); {Pauses program before next clear-screen command}
0833      CLRSCL;
0834
0835      {Input surface winds}
0836      write('Begin altitude/wind data input starting with surface');
0837      writeln(' level winds:');
0838      writeln('ALTITUDE      = 0');
0839
0840      repeat {until wind direction = 0-360 degrees}
0841      write('WIND DIRECTION = ');
0842      readln(WindsAloft[1,2]);
0843      if (WindsAloft[1,2] < 0.0) or (WindsAloft[1,2] > 360.0) then
0844      begin {Wind direction not in 0-360 degree range}
0845          writeln('Wind direction must be between 0 and 360. Try again!');
0846          writeln
0847      end {Wind direction not in 0-360 degree range}
0848      until (WindsAloft[1,2] >= 0.0) and (WindsAloft[1,2] <= 360.0);
0849
0850      repeat {until wind velocity = 0-99 (knots, or whatever) }
0851      write('WIND VELOCITY = ');
0852      readln(WindsAloft[1,3]);
0853      if (WindsAloft[1,3] < 0.0) or (WindsAloft[1,3] > 99.0) then
0854      writeln('Wind velocity must be between 0 and 99. Try again!');
0855      writeln
0856      until (WindsAloft[1,3] >= 0.0) and (WindsAloft[1,3] <= 99.0);
0857
0858      {Input winds at other altitudes}
0859      I := 1; {Used to repeat loop until all valid winds are input}
0860      N := 1; {Used to note number of altitudes used}
0861
0862      if (IncidentAlt - TerrainHeight) >= 1500 then
0863      begin {for altitudes >= 1500 feet}
0864
0865          repeat {until I = 21}
0866          I := I + 1; {I = 2 to 20 individual altitudes/winds data input loop}
0867          N := N + 1; {Notes number of altitudes used}
0868
0869          {Repeat altitude input process until the next altitude input is at}
0870          {least 1000 feet higher than the previous one, greater than 1000, }
0871          {1000 feet, and less than or equal to 42000 feet.}
0872          repeat
0873              write('ALTITUDE      = ');
0874              readln(WindsAloft[I,1]);
0875
0876              if WindsAloft[I,1] < WindsAloft[I-1,1] then
0877              begin {Input altitude must be greater than previous one input}
0878                  write('This altitude must be higher than the last altitude');
0879                  writeln(' input. Try again!');

```

```

0880         writeln
0881     end;    {Input altitude must be greater than previous one input}
0882
0883     if WindsAloft[I,1] < 1000.0 then
0884     begin {Input altitude must be >= than 1000}
0885         writeln('This altitude must be >= 1000. Try again!');
0886         writeln
0887     end;    {Input altitude must be >= than 1000}
0888
0889     Temp2 := (WindsAloft[I,1] / 1000.0) - trunc(WindsAloft[I,1] / 1000.0);
0890     if Temp2 < 0.0 then
0891     begin {WindsAloft must be a multiple of 1000}
0892         write('This altitude must be a multiple of 1000 feet. ');
0893         writeln(' Try again!');
0894         writeln
0895     end;    {WindsAloft must be a multiple of 1000}
0896
0897     if WindsAloft[I,1] > 42000.0 then
0898     begin {Input altitude must be less than 42000}
0899         writeln('This altitude must be <= 42000. Try again!');
0900         writeln
0901     end    {Input altitude must be less than 42000}
0902
0903     until (WindsAloft[I,1] > WindsAloft[I-1,1]) and
0904           (WindsAloft[I,1] = 1000.0) and (WindsAloft[I,1] <= 42000.0);
0905
0906     repeat {until wind direction = 0-360 degrees}
0907     write('WIND DIRECTION = ');
0908     readln(WindsAloft[I,2]);
0909     if (WindsAloft[I,2] < 0.0) or (WindsAloft[I,2] > 360.0) then
0910     begin {Wind direction not in 0-360 degree range}
0911         write('Wind direction must be between 0 and 360. Try again!');
0912         writeln;
0913         writeln
0914     end    {Wind direction not in 0-360 degree range}
0915     until (WindsAloft[I,2] >= 0.0) and (WindsAloft[I,2] <= 360.0);
0916
0917     repeat {until wind velocity = 0-99 (knots, or whatever) }
0918     write('WIND VELOCITY = ');
0919     readln(WindsAloft[I,3]);
0920     if (WindsAloft[I,3] < 0.0) or (WindsAloft[I,3] > 99.0) then
0921         writeln('Wind velocity must be between 0 and 99. Try again!');
0922         writeln
0923     until (WindsAloft[I,3] >= 0.0) and (WindsAloft[I,3] <= 99.0);
0924
0925     {P notes first altitude >= IncidentAlt (AGL) and requests one more}
0926     if (WindsAloft[I,1] >= IncidentAlt) then
0927     begin {Input altitude >= IncidentAlt (AGL)}
0928         P := P + 1;
0929         if P = 1 then
0930             writeln('Enter data for just one more, higher altitude. ');
0931         writeln

```

```

0932         end; {Input altitude }= IncidentAlt (AGL)}
0933
0934     {P notes second altitude}IncidentAlt (AGL) then terminates the input loop}
0935     {by setting I equal to 21.}
0936     if (WindsAloft[I,1] )= IncidentAlt) and (P = 2) then I := 21
0937
0938         until I = 21
0939     end; {for altitudes }= 1500 feet}
0940
0941     CLRSCR;
0942     VerifyWinds;
0943     WindDrift(IncidentAlt)
0944
0945     end {winds aloft computations}
0946
0947 end; {InputWinds}
0948
0949 {*****}
0950 {Asks user for aerospace object glide information, if applicable.}
0951
0952 procedure AeroGlide;
0953
0954 const FtNm = 6076;
0955
0956 var Xcomponent,      {Horizontal component of individual vectors}
0957     Ycomponent : real; {Vertical component of individual vectors}
0958
0959 begin {AeroGlide}
0960     Xcomponent := 0.0; Ycomponent := 0.0; {initialize}
0961
0962     repeat {until valid Glide Ratio}
0963         writeln('Please enter the aerospace object's power-off glide ratio. ');
0964         writeln('Refer to object's operations manual, if one exists. ');
0965         writeln('Otherwise, enter best estimate. For example: If the object');
0966         writeln('glides 3 feet horizontally for every foot of vertical descent');
0967         write('(3/1), enter a 3. GLIDE RATIO = ');
0968         readln(GlideRatio);
0969         if GlideRatio < 0 then
0970             writeln('Glide Ratio must not be less than zero. Try again!');
0971         writeln
0972     until GlideRatio >= 0;
0973
0974     GlideDistance := (GlideRatio * GlideAltLost) / FtNm; {Converts ft to nm}
0975
0976     repeat {until valid compass heading}
0977         writeln('Please enter descent or bailout heading, if known');
0978         write('Enter 361 if not known. HEADING (0-361) = ');
0979         readln(DescentHeading);
0980         if (DescentHeading < 0) or (DescentHeading > 361) then
0981             writeln('This heading must be between 0-361. Try again!');
0982         writeln
0983     until (DescentHeading >= 0) and (DescentHeading <= 361);

```

```

0984
0985     repeat {until valid descent rate}
0986         writeln('Please enter aerospace object''s rate of descent in');
0987         write('feet-per-minute. DESCENT RATE = ');
0988         readln(DescentRate);
0989         if DescentRate < 0 then
0990             writeln('Descent rate must not be less than zero. Try again!');
0991             writeln
0992         until DescentRate >= 0;
0993         DescentTime := GlideAltLost / DescentRate;
0994
0995     {Returns average wind affecting object from incident altitude to parachute}
0996     {opening altitude or surface level.}
0997     InputWinds(AssignedAlt);
0998     WindDisplacement := (GlideWindSpeed / 60) * DescentTime;
0999
1000     if DescentHeading < 361 then {Descent or bailout heading known}
1001     begin {Calculate resultant vector of Glide + Wind vectors.}
1002
1003         {Convert winds bearing on gliding object into radian units.}
1004         BrgRads := BrgGlideWinds / radians;
1005         Xcomponent := sin(BrgRads) * WindDisplacement;
1006         Ycomponent := cos(BrgRads) * WindDisplacement;
1007
1008         BrgRads := DescentHeading / radians;
1009         Xcomponent := Xcomponent + sin(BrgRads) * GlideDistance;
1010         Ycomponent := Ycomponent + cos(BrgRads) * GlideDistance;
1011         AddVectors(Xcomponent, Ycomponent);
1012         BrgMaxGlide := AvgWindTo;
1013         MaxDistanceGlide := ResultMagnitude
1014     end {Calculate resultant vector of Glide + Wind vectors.}
1015     else
1016     begin {descent or bailout heading unknown}
1017         BrgMaxGlide := BrgGlideWinds;
1018         MaxDistanceGlide := WindDisplacement;
1019         MaxRadiusGlide := GlideDistance
1020     end; {descent or bailout heading unknown}
1021
1022     if (AssignedAlt - GlideAltLost) >= 500.0 then
1023     begin {Parachute(s) were used}
1024         Ejection;
1025
1026         {Calculate resultant vector of MaxGlide + Bailout vectors.}
1027         BrgRads := BrgMaxGlide / radians; {Convert bearing into radian units.}
1028         Xcomponent := sin(BrgRads) * MaxDistanceGlide;
1029         Ycomponent := cos(BrgRads) * MaxDistanceGlide;
1030
1031         if (DescentHeading < 361) and (EjectDistance > 0.0) then
1032         begin {Descent heading known AND EjectDistance > 0}
1033             {Convert bearing into radian units.}
1034             BrgRads := DescentHeading / radians;
1035

```

```

1036      {Total all vector's individual X and Y components.}
1037      Xcomponent := Xcomponent + sin(BrgRads) * EjectDistance;
1038      Ycomponent := Ycomponent + cos(BrgRads) * EjectDistance;
1039      AddVectors(Xcomponent, Ycomponent);
1040      TrajectBrg := AvgWindTo;
1041      TrajectDistance := ResultMagnitude
1042    end {Descent heading known AND EjectDistance > 0}
1043  else
1044    begin {Descent heading unknown OR EjectDistance = 0}
1045      TrajectBrg := BrgMaxGlide;
1046      TrajectDistance := MaxDistanceGlide;
1047      if DescentHeading = 361 then
1048        if EjectDistance > 0.0 then
1049          TrajectRadius := MaxRadiusGlide + EjectDistance
1050        else TrajectRadius := MaxRadiusGlide
1051      end; {Descent heading unknown OR EjectDistance = 0}
1052
1053      MethodDrift := 'P'; {Switches from "B" to "P" mode as new MethodDrift}
1054      WindDrift(ChuteAlt)
1055    end {Parachute(s) were used}
1056  end; {AeroGlide}
1057
1058  {*****}
1059  {Gathers additional information concerning the aerospace object's glide }
1060  {before commencing calculations. }
1061
1062  procedure GlideData;
1063
1064  begin {GlideData}
1065    MethodDrift := 'G'; {Selects "Glide" mode as the method of drift.}
1066
1067    repeat {Until valid AssignedAlt response}
1068      write('Approximate (to the nearest 500 feet) at what altitude ');
1069      writeln('above');
1070      write('mean sea level (MSL) did the aerospace object's ');
1071      writeln('power-off');
1072      write('descent begin? (If unknown, enter the MSL height of the ');
1073      writeln('last');
1074      write('assigned flight altitude.) ALTITUDE= ');
1075      readln(AssignedAlt);
1076
1077      {Used to ensure AssignedAlt is a multiple of 500}
1078      Temp := ( AssignedAlt/500.0 ) - trunc( AssignedAlt/500.0 );
1079      if Temp < 0 then
1080        writeln('Given altitude is not a multiple of 500 ft. ');
1081        writeln
1082      until AssignedAlt > TerrainHeight;
1083
1084      AssignedAlt := AssignedAlt - TerrainHeight; {Convert to AGL altitude}
1085
1086    repeat {until valid altitude}
1087      writeln('Please enter the altitude lost (to the nearest 500 feet)');

```

```

1088     writeln('between the above altitude and the bailout/parachute');
1089     writeln('deployment altitude. Altitude lost must be at least 500');
1090     writeln('feet. (If parachutes were not used, enter the altitude');
1091     writeln('lost before surface contact.));
1092     write('ALTITUDE LOST => ');
1093     readln(GlideAltLost);
1094
1095     {Used to ensure GlideAltLost is a multiple of 500}
1096     Temp := ( GlideAltLost/500.0 ) - trunc( GlideAltLost/500.0 );
1097
1098     if Temp < 0 then
1099     begin {GlideAltLost not multiple of 500 feet}
1100         write('Given altitude lost is not a multiple of 500 ft. ');
1101         writeln('Try again!')
1102     end; {GlideAltLost not multiple of 500 feet}
1103
1104     if GlideAltLost < 500 then
1105     begin {GlideAltLost not >= 500 feet}
1106         write('This altitude must be at least 500 feet. ');
1107         writeln('Try again!')
1108     end; {GlideAltLost not >= 500 feet}
1109
1110     if GlideAltLost > AssignedAlt then
1111     begin {GlideAltLost greater than possible}
1112         write('It is impossible to lose more altitude than you have. ');
1113         writeln('Try again!')
1114     end; {GlideAltLost greater than possible}
1115
1116     writeln
1117     until (Temp = 0.0) and (GlideAltLost >= 500) and
1118         (GlideAltLost (= AssignedAlt);
1119
1120     ChuteAlt := AssignedAlt - GlideAltLost;
1121     AeroGlide
1122
1123 end; {GlideData}
1124
1125 {*****}
1126 {Gathers additional information concerning the aerospace object's parachute }
1127 {descent before commencing calculations. }
1128
1129 procedure ChuteData;
1130
1131 begin {ChuteData}
1132
1133     repeat {until valid response}
1134         writeln('After receiving report of the search object's last known');
1135         writeln('coordinates (lat/long), did the object successfully use');
1136         write('parachutes to descend? (y/n) => ');
1137         readln(chute);
1138         writeln
1139     until (chute in ValidAnswers);

```

```

1140
1141   if (chute in Yes) then
1142   begin {parachute(s) used}
1143       MethodDrift := 'P'; {Selects "Parachute" mode as the method of drift.}
1144
1145       repeat {Until valid IncidentAlt response}
1146           write('Approximate (to the nearest 500 feet) at what ');
1147           writeln('altitude');
1148           write('above mean sea level (MSL) did the parachute(s) ');
1149           writeln('open?');
1150           writeln('Enter a zero if descent began below 500 feet AGL');
1151           write('ALTITUDE= ');
1152           readln(IncidentAlt);
1153
1154           if ( (IncidentAlt-TerrainHeight) ( 500 ) and
1155               (IncidentAlt () 0.0) then
1156               begin {IncidentAlt not >= 500 feet AGL}
1157                   write('This altitude must be at least 500 feet AGL. ');
1158                   writeln('Try again!');
1159                   writeln('Type a zero if not applicable');
1160               end; {IncidentAlt not >= 500 feet AGL}
1161
1162           if IncidentAlt = 0 then
1163               begin {Parachute(s) not effectively used}
1164                   writeln;
1165                   writeln('By entering a zero to this question you have');
1166                   writeln('indicated that parachute(s) were not used 500');
1167                   writeln('feet or more above surface level. This means');
1168                   writeln('parachute drift does not significantly affect');
1169                   writeln('the surface position of your search object. ');
1170                   writeln('You should, therefore, proceed with search');
1171                   writeln('planning using the object''s last known position');
1172                   writeln('as the surface starting (DATUM) point. ');
1173               end; {Parachute(s) not effectively used}
1174
1175           {Used to ensure IncidentAlt is a multiple of 500}
1176           Temp := ( IncidentAlt/500.0 ) - trunc( IncidentAlt/500.0 );
1177           writeln
1178           {IncidentAlt must be 500-42000 feet and in units of 500 feet, or = 0}
1179           until (IncidentAlt >= 0.0) and (IncidentAlt <= 42000.0) and
1180               (Temp = 0.0);
1181
1182           ChuteAlt := IncidentAlt - TerrainHeight; {Convert to AGL altitude}
1183           {If ChuteAlt >= 500 feet then continue with program run, else terminate.}
1184           if ChuteAlt () 0.0 then InputWinds(ChuteAlt)
1185
1186       end {parachute(s) used}
1187   end; {ChuteData}
1188
1189   {*****}
1190   {Prints out record of search planning inputs and calculations from}
1191   {this program. }

```

```

1192
1193 procedure WriteToDisk;
1194
1195 var I          : integer;
1196     LastDegree,
1197     LastMinute : real;
1198     AeroDat     : text;
1199
1200 begin {WriteToDisk}
1201     assign(AeroDat,'AeroData');
1202     rewrite(AeroDat);
1203
1204     LastDegree := trunc(LastLatitudeKnown);
1205     LastMinute := round( (LastLatitudeKnown-LastDegree) * 100);
1206     write(AeroDat,'Missing aerospace object/pilot(s) last known position:');
1207     if LastDegree < 10 then write(AeroDat,' 0',LastDegree:1:0,'-')
1208     else write(AeroDat,' ',LastDegree:2:0,'-');
1209     if LastMinute < 10 then write(AeroDat,'0',LastMinute:1:0,' ')
1210     else write(AeroDat,LastMinute:2:0,' ');
1211     if LatNS = 'N' then write(AeroDat,'North') else write(AeroDat,'South');
1212
1213     LastDegree := trunc(LastLongitudeKnown);
1214     LastMinute := round( (LastLongitudeKnown-LastDegree) * 100);
1215     if LastDegree < 10 then write(AeroDat,' 0',LastDegree:1:0,'-')
1216     else write(AeroDat,' ',LastDegree:2:0,'-');
1217     if LastMinute < 10 then write(AeroDat,'0',LastMinute:1:0,' ')
1218     else write(AeroDat,LastMinute:2:0,' ');
1219     if LongEW = 'W' then write(AeroDat,'West') else write(AeroDat,'East');
1220
1221     {Prints a zero in front of one-digit dates in Z DTG format}
1222     if trunc(LastDateTime / 10000) < 10 then
1223     write(AeroDat,'Time: 0',LastDateTime:5:0,'Z ',LastMonth:2,'/',LastYear:4)
1224     else
1225     write(AeroDat,'Time: ',LastDateTime:6:0,'Z ',LastMonth:2,'/',LastYear:4);
1226
1227     if GlideAltLost > 0.0 then
1228     begin {indicates procedure AeroGlide was used}
1229         write(AeroDat,'Incident or last assigned altitude = ');
1230         write(AeroDat,(AssignedAlt+TerrainHeight):6:0,' feet MSL');
1231         write(AeroDat,'Surface level or terrain height = ');
1232         write(AeroDat,TerrainHeight:6:0,' feet MSL');
1233         write(AeroDat,'Altitude lost during glide or descent = ');
1234         write(AeroDat,GlideAltLost:6:0,' feet');
1235         write(AeroDat,'Power-off glide ratio (Hz/Vertical) = ');
1236         write(AeroDat,GlideRatio:6,' to 1');
1237         write(AeroDat,'Maximum horizontal glide distance = ');
1238         write(AeroDat,GlideDistance:6:1,' nautical miles');
1239         if DescentHeading < 361 then
1240         begin {descent heading known}
1241             write(AeroDat,'Descent or bailout heading = ');
1242             write(AeroDat,DescentHeading:6:0,' degrees True')
1243         end {descent heading known}

```

```

1244     else
1245     begin {descent heading unknown}
1246         write(AeroDat,'Descent or bailout heading          = ');
1247         writeln(AeroDat,'Unknown')
1248     end; {descent heading unknown}
1249     if DescentRate > 0 then
1250     begin
1251         write(AeroDat,'Rate of descent                      = ');
1252         writeln(AeroDat,DescentRate:6:0,' feet per minute');
1253         write(AeroDat,'Time of descent                      = ');
1254         writeln(AeroDat,DescentTime:6:0,' minutes')
1255     end;
1256     write(AeroDat,'Average descent wind bearing            = ');
1257     writeln(AeroDat,BrgGlideWinds:6:0,' degrees True');
1258     write(AeroDat,'Average descent wind velocity (knots) = ');
1259     writeln(AeroDat,GlideWindSpeed:6:0,' knots');
1260     write(AeroDat,'Aircraft displacement due to winds     = ');
1261     writeln(AeroDat,WindDisplacement:6:1,' nautical miles');
1262     write(AeroDat,'Maximum glide bearing                   = ');
1263     writeln(AeroDat,BrgMaxGlide:6:0,' degrees True');
1264     write(AeroDat,'Maximum aircraft glide                  = ');
1265     writeln(AeroDat,MaxDistanceGlide:6:1,' nautical miles');
1266     if MaxRadiusGlide > 0.0 then
1267     begin {MaxRadiusGlide} 0 due to uncertain data inputs}
1268         write(AeroDat,'Maximum glide radius                 = ');
1269         writeln(AeroDat,MaxRadiusGlide:6:1,' nautical miles.')
1270     end; {MaxRadiusGlide} 0 due to uncertain data inputs}
1271     if EjectDistance > 0.0 then
1272     begin {aerospace object is/are pilot(s) ejecting}
1273         write(AeroDat,'Ejection displacement               = ');
1274         writeln(AeroDat,EjectDistance:6:1,' nautical miles')
1275     end; {aerospace object is/are pilot(s) ejecting}
1276     write(AeroDat,'Aerospace-trajectory drift bearing      = ');
1277     writeln(AeroDat,TrajectBrg:6:0,' degrees True');
1278     write(AeroDat,'Aerospace-trajectory drift distance     = ');
1279     writeln(AeroDat,TrajectDistance:6:1,' nautical miles');
1280     if TrajectRadius > 0.0 then
1281     begin {TrajectRadius} 0 due to uncertain data inputs}
1282         write(AeroDat,'Aerospace-trajectory drift radius   = ');
1283         writeln(AeroDat,TrajectRadius:6:1,' nautical miles.')
1284     end {TrajectRadius} 0 due to uncertain data inputs}
1285     end; {indicates procedure AeroGlide was used};
1286
1287     if MethodDrift = 'P' then
1288     begin {indicates parachute(s) used}
1289         write(AeroDat,'Bailout/Parachute opening altitude   = ');
1290         writeln(AeroDat,(ChuteAlt+TerrainHeight):6:0,' feet MSL');
1291         if GlideAltLost = 0.0 then
1292         begin {if TerrainHeight not previously recorded via glide calculations}
1293             write(AeroDat,'Surface level or terrain height = ');
1294             writeln(AeroDat,TerrainHeight:6:0,' feet MSL')
1295         end; {if TerrainHeight not previously recorded via glide calculations}

```

```

1296 write(AeroDat,'Wind bearing affecting para-descent = ');
1297 writeln(AeroDat,BrgParaDrift:6:0,' degrees True');
1298 write(AeroDat,'Wind velocity affecting para-descent = ');
1299 writeln(AeroDat,ParaDriftSpeed:6:1,' knots');
1300 write(AeroDat,'Parachute table descent rate = ');
1301 writeln(AeroDat,RateParaDescent:6:1,' feet per second');
1302 write(AeroDat,'Parachute drift table distance = ');
1303 writeln(AeroDat,DistanceParaDrift:6:1,' nautical miles');
1304 if (GlideAltLost = 0.0) and (EjectDistance) 0.0 then
1305 begin {if EjectDistance not previously recorded}
1306 write(AeroDat,'Ejection displacement = ');
1307 writeln(AeroDat,EjectDistance:6:1,' nautical miles')
1308 end; {if EjectDistance not previously recorded}
1309 write(AeroDat,'Total aerospace drift bearing = ');
1310 writeln(AeroDat>TotalBrg:6:0,' degrees True');
1311 write(AeroDat,'Total aerospace drift distance = ');
1312 writeln(AeroDat>TotalDistance:6:1,' nautical miles');
1313 if TotalRadius > 0.0 then
1314 begin {TotalRadius > 0 due to uncertain data inputs}
1315 write(AeroDat,'Total aerospace drift radius = ');
1316 writeln(AeroDat>TotalRadius:6:1,' nautical miles')
1317 end; {TotalRadius > 0 due to uncertain data inputs}
1318 write(AeroDat,'Total time of parachute descent = ');
1319 writeln(AeroDat,TimeParaDescent:6:1,' minutes');
1320
1321 end; {indicates parachute(s) used}
1322
1323 writeln(AeroDat,' ');
1324 writeln(AeroDat,' Wind Data Used To Calculate Above Results:');
1325 {Write out record of winds aloft used by program}
1326 for I := 1 to N do
1327 begin {I = 1 to number of winds/altitudes used by program}
1328 write(AeroDat,I:2,'. Winds at ',WindsAloft[I,1]:6:0,' feet: ');
1329 write(AeroDat,WindsAloft[I,2]:4:0,' degrees at ',WindsAloft[I,3]:3:0);
1330 writeln(AeroDat,' knots')
1331 end; {I = 1 to number of winds/altitudes used by program}
1332
1333 close(AeroDat)
1334
1335 end; {WriteToDisk}
1336
1337 {*****}
1338 {Provides user with program information, limitations on use and license. }
1339
1340 procedure Warranty;
1341
1342 var continue : char; {Bogus read variable provides tize to read warranty}
1343
1344 begin {Warranty}
1345
1346 write('*****');
1347 writeln('*****');

```

```

1348 write('(* SEARCH PLANNING SOFTWARE ');
1349 writeln(' (PROGRAM #1 OF 3) ');
1350 write('(* TITLE: AERODRIF.COM (Aerospace Drift ');
1351 writeln('Algorithm) ');
1352 write('(* VERSION: 1.1 for CP/M ');
1353 writeln('Operating System ');
1354 write('(* DATE WRITTEN: September 1984');
1355 writeln(' ');
1356 write('(* LICENSE: COPYRIGHT 1984');
1357 writeln(' D. RICK DOUGLAS ');
1358 write('*****');
1359 writeln('*****');
1360 write('The author makes no express or implied ');
1361 writeln('warranty of any kind with regard to');
1362 write('this program material, including, but ');
1363 writeln('not limited to, the implied warranty of');
1364 write('fitness for a particular purpose. ');
1365 writeln('The author shall not be liable for');
1366 write('incidental or consequential damages ');
1367 writeln('in connection with or arising out of');
1368 write('furnishing, use, or performance of this ');
1369 writeln('program. The reader MUST HAVE a solid');
1370 write('understanding of search and rescue ');
1371 writeln('methodology before using this software in');
1372 write('making decisions where human life is at ');
1373 writeln('risk. In fact, since no amount of');
1374 write('testing can uncover 100% of program ');
1375 writeln('errors, this program is recommended for');
1376 write('training use only. Prior attendance ');
1377 writeln('at the United States Coast Guard's');
1378 writeln('National SAR School is highly-encouraged. ');
1379 writeln;
1380 write('*****');
1381 writeln(' WARNING! *****');
1382 write('(* THIS SOFTWARE MAY BE FREELY-');
1383 writeln('DISTRIBUTED PROVIDED NO FEE ');
1384 write('(* IS CHARGED AND THIS');
1385 writeln(' COPYRIGHT NOTICE IS RETAINED ');
1386 write('*****');
1387 writeln('*****');
1388 writeln;
1389 write('PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE');
1390 readln(continue)
1391
1392 end; {Warranty}
1393
1394 {*****}
1395 begin {main program}
1396
1397 {initialize program variables}
1398 AssignedAlt := -1000.0; AvgWindFrom := 0.0; BrgGlideWinds := 0.0;
1399 BrgMaxGlide := 0.0; BrgParaDrift := 0.0; ChuteAlt := 0.0;

```

```

1400 chute := 'Q'; descent := 'Q'; DescentHeading := -1.0;
1401 DescentTime := 0.0; DistanceParaDrift := 0.0;
1402 EjectDistance := -1.0; GlideAltLost := 0.0; GlideRatio := 0;
1403 GlideWindSpeed := 0.0; IncidentAlt := -1000.0; LastDateTime := 0.0;
1404 LastLatitudeKnown := 0.0; LastLongitudeKnown := 0.0; LatNS := 'Q';
1405 LongEW := 'Q'; N := 0; MaxDistanceGlide := 0.0; MaxRadiusGlide := 0.0;
1406 MethodDrift := 'Q'; OnePassCompleted := false; ParaDriftSpeed := 0.0;
1407 ResultMagnitude := 0.0; TerrainHeight := -1000.0; TimeParaDescent := 0.0;
1408 TotalBrq := 0.0; TotalDistance := 0.0; TotalRadius := 0.0;
1409 TrajectBrq := 0.0; TrajectDistance := 0.0; TrajectRadius := 0.0;
1410 WindDisplacement := 0.0;
1411
1412 ValidAnswers := ['Y','y','N','n']; No := ['N','n']; Yes := ['Y','y'];
1413 for I := 1 to 20 do for J := 1 to 3 do WindsAloft[I,J] := 0.0;
1414 for I := 1 to 20 do for J := 1 to 3 do AeroWinds[I,J] := 0.0;
1415
1416 Warranty;
1417 CLRSCL;
1418 writeln('This program calculates the initial surface position coordinates');
1419 writeln('(latitude/longitude) of a falling, gliding, or parachuting');
1420 writeln('aerospace object or person. If these coordinates are all ready');
1421 writeln('known, or the search object did not fall at least 500 feet, then');
1422 writeln('enter N to the next question and proceed with other search');
1423 writeln('planning as outlined in the National Search and Rescue Manual. ');
1424 writeln;
1425 repeat {until valid response}
1426   write('Do you wish to continue with this program? (y/n) = ');
1427   readln(continue);
1428   writeln
1429 until (continue in ValidAnswers);
1430 if (continue in Yes) then
1431 begin {program run}
1432   CLRSCL;
1433   AeroPosition;
1434
1435   repeat {until valid TerrainHeight response}
1436     writeln('Please enter the surface level altitude or terrain height');
1437     writeln('(in feet) above or below mean sea level. Enter a zero if');
1438     writeln('altitude equals sea level, or, enter a negative altitude if');
1439     writeln('below sea level (e.g. Death Valley, California.). ');
1440     write('SURFACE LEVEL/TERRAIN HEIGHT = ');
1441     readln(TerrainHeight);
1442     writeln
1443   until TerrainHeight > -1000;
1444
1445   repeat {until valid response}
1446     writeln('After receiving report of the search object's last known');
1447     writeln('coordinates (lat/long), did the object fall or glide to a');
1448     writeln('lower altitude before parachute(s) opened or ground contact');
1449     write('was made? (y/n) = ');
1450     readln(descent);
1451     writeln

```

```

1452      until (descent in ValidAnswers);
1453
1454      if (descent in Yes) then GlideData;
1455      if (descent in No) then ChuteData;
1456
1457      {Only write transaction record to disk if program procedures AeroGlide}
1458      {or InputWinds were actually used}
1459
1460      {If program used, then create record of inputs and results.}
1461      if (MethodDrift = 'P') or (MethodDrift = 'G') or (MethodDrift = 'B') then
1462          WriteToDisk;
1463      writeln;
1464      writeln('A record of significant input and output data used during this');
1465      writeln('program run is stored in an external file named "AERODATA."');
1466      writeln('If you desire to keep this record permanently, please rename');
1467      writeln('file AERODATA before running this program again!')
1468
1469      end      {program run}
1470  end. {main program}

```

**Aerospace Drift Determination Program (01 of 3)
Variable & Operator Cross-Reference Listing**

<u>Variable</u>	<u>Program Line Number</u>									
AddVectors	291	614	680	703	1011	1039				
AeroDat	1198	1201	1202	1206	1207	1208	1209	1210	1211	1211
	1215	1216	1217	1218	1219	1219	1223	1225	1229	1230
	1231	1232	1233	1234	1235	1236	1237	1238	1241	1242
	1246	1247	1251	1252	1253	1254	1256	1257	1258	1259
	1260	1261	1262	1263	1264	1265	1268	1269	1273	1274
	1276	1277	1278	1279	1282	1283	1289	1290	1293	1294
	1296	1297	1298	1299	1300	1301	1302	1303	1306	1307
	1309	1310	1311	1312	1315	1316	1318	1319	1323	1324
	1328	1329	1330	1332						
AeroGlide	952	1121								
AeroPosition	213	1432								
AeroSpaceWinds	78									
AeroWinds	141	352	352	365	365	366	373	373	379	379
	386	386	389	389	400	401	401	438	439	448
	448	449	453	453	454	462	471	472	481	486
	491	491	499	499	500	509	509	514	514	519
	519	529	530	530	580	607	610	611	1413	
AssignedAlt	101	584	598	997	1022	1075	1078	1078	1082	1084
	1084	1110	1110	1120	1230	1397				
AvgWindFrom	102	301	305	306	309	310	313	314	315	681
	704	1397								
AvgWindTo	103	314	315	618	625	1012	1040			
BrgGlideWinds	104	618	1004	1017	1257	1397				
BrgMaxGlide	105	1012	1017	1027	1045	1263	1398			
BrgParaDrift	106	625	629	671	686	697	1297	1398		
BrgRads	107	573	607	610	611	671	672	673	677	678
	679	697	698	699	700	701	702	1004	1005	1006
	1008	1009	1010	1027	1028	1029	1034	1037	1038	
chute	87	1137	1139	1141	1399					
ChuteAlt	108	438	439	459	462	471	1054	1120	1182	1184
	1184	1290	1398							

ChuteData	1129	1454									
continue	88	832	1341	1389	1426	1428	1429				
count	149	152	153	156	156						
descent	89	1399	1449	1451	1453	1454					
DescentHeading	109	663	664	664	667	667	675	677	688	979	
	980	980	983	983	1000	1000	1031	1034	1047	1239	
	1242	1399									
DescentRate	110	988	989	992	993	1249	1252				
DescentTime	111	993	998	1254	1400						
DistanceParaDrift	112	640	672	673	687	698	699	1303	1400		
EjectDistance	113	561	562	563	675	678	679	689	690	1031	
	1037	1038	1048	1049	1271	1274	1304	1307	1401		
Ejection	544	668	1024								
FtNm	954	974									
GlideAltLost	114	584	598	619	974	993	1022	1093	1096	1096	
	1104	1110	1117	1118	1120	1227	1234	1291	1304	1401	
GlideData	1062	1453									
GlideDistance	115	974	1009	1010	1019	1238					
GlideOrChute	336	585	589								
GlideRatio	94	968	969	972	974	1236	1401				
GlideThenChute	419	598									
GlideWindSpeed	116	620	998	1259	1402						
H	421	515	527								
HighAltitude	342	347	357	365	368	370	372	376	425	430	
	453	480	481	494	499	501	501	504	506	508	
	522										

I	95	338	347	347	348	348	360	365	365	365
	366	368	368	370	372	372	373	373	374	374
	376	379	379	380	385	386	386	389	389	390
	392	398	400	401	401	421	430	430	431	431
	435	438	438	438	439	448	448	448	449	453
	453	453	454	459	461	462	469	471	471	472
	480	481	481	486	491	491	494	496	496	499
	499	499	500	501	501	504	504	506	508	508
	509	509	513	514	514	515	519	522	527	529
	530	530	572	580	580	580	604	607	610	611
	719	722	724	724	725	725	796	859	866	866
	874	876	876	883	889	889	897	903	903	904
	904	908	909	909	915	915	919	920	920	923
	923	926	936	936	938	1195	1326	1328	1328	1329
	1329	1412	1412	1413	1413					
IncidentAlt	117	336	352	370	376	385	419	480	486	494
	501	501	506	513	519	522	570	585	589	598
	626	656	794	805	862	926	936	943	1152	1154
	1155	1162	1176	1176	1179	1179	1182	1402		
InputWinds	794	997	1184							
J	95	572	580	580	580	737	770	770	1412	1412
	1413	1413								
L	421	474	527							
LastDateTime	120	174	180	184	184	191	195	195	202	245
	1222	1223	1225	1402						
LastDegree	1196	1204	1205	1207	1207	1208	1213	1214	1215	1215
	1216									
LastLatitudeKnown	118	260	261	261	264	264	1204	1205	1403	
LastLongitudeKnown	119	277	278	278	281	281	1213	1214	1403	
LastMinute	1197	1205	1209	1209	1210	1214	1217	1217	1218	
LastMonth	97	227	228	228	231	231	1223	1225		
LastYear	98	236	1223	1225						
LatNS	90	252	254	254	254	254	1211	1403		
lines	147	153								
LongEW	91	269	271	271	271	271	1219	1404		
LowAltitude	341	348	368	372	385	424	431	448	459	471
	504	508	513							

MaxDistanceGlide	121	1013	1018	1028	1029	1046	1265	1404		
MaxRadiusGlide	122	1019	1049	1050	1266	1269	1404			
MethodDrift	92 1460	584 1460	589 1460	616	623	1053	1065	1143	1287	1405
N	96 580	355 604	380 722	390 860	392 867	398 867	461 1326	469 1404	474	519
No	137	754	1411	1454						
OnePassCompleted	84	593	658	694	710	1405				
P	797	801	928	928	929	936				
ParaDriftSpeed	123	627	630	1299	1405					
radians	80 1034	298	607	671	677	697	700	1004	1008	1027
RateParaDescent	124	652	656	1301						
ResultMagnitude	125	319	620	627	682	705	1013	1041	1406	
Temp	126 1117	165 1176	184 1180	185	195	196	1078	1079	1096	1098
Temp1	339 509	347	372	373	422	430	462	470	491	508
Temp2	340 514	348 790	385 889	386 890	423	431	481	486	491	513
TempAlt	574	619	620	626	627					
TempAngle	293	298	301	304	305	306	308	309	310	
TempDate	166	174	175	175	205	205				
TempHour	167	185	186	186	205	206				
TempMinutes	168	196	197	197	206	206				
TerrainHeight	127	584	862	1002	1004	1154	1182	1230	1232	1290
	1294	1406	1440	1442						
TimeParaDescent	128	656	1319	1406						
TotalBrg	129	681	686	704	1310	1407				

TotalDistance	130	682	687	705	1312	1407				
TotalRadius	131	690	706	1313	1316	1407				
TrajectBrq	132	700	1040	1045	1277	1408				
TrajectDistance	133	701	702	1041	1046	1279	1408			
TrajectRadius	134	706	1049	1050	1280	1283	1408			
TypeVehicle	99	557	559	559	561	562	563			
ValidAnswers	139	752	781	1139	1411	1420	1451			
VerifyDTG	163	246								
VerifyWinds	734	942								
Warranty	1339	1415								
WindChart	717	744	774							
WindCheck	736	741	750	752	754	779	781			
WindDisplacement	135	998	1005	1006	1018	1261	1409			
WindDrift	570	943	1054							
WindErrorr	738	741	759	761	761	763	767	768	769	770
Winds	82	142								
WindsAloft	142	580	724	725	725	767	768	769	770	842
	843	843	848	848	852	853	853	856	856	874
	876	876	883	889	889	897	903	903	904	904
	908	909	909	915	915	919	920	920	923	923
	926	936	1328	1329	1329	1412				
writeln	147	771	784	802	833	941	1416	1431		
WriteToDisk	1193	1461								
Xcomponent	291	298	301	302	303	307	319	319	321	575
	603	610	610	614	672	678	678	680	690	701
	701	703	956	960	1005	1009	1009	1011	1020	1037
	1037	1039								
Ycomponent	291	298	301	302	303	307	319	319	322	576
	603	611	611	614	673	679	679	680	699	702
	702	703	957	960	1006	1010	1010	1011	1029	1038
	1038	1039								

Yes 138 1141 1411 1429 1453

<u>Operator</u>	<u>Program Line Number</u>									
arctan	298									
assign	1201									
boolean	84									
char	92	139	736	1341						
close	1332									
cos	611	673	679	699	702	1006	1010	1029	1038	
false	593	658	1405							
input	78									
integer	99 1195	147	149	338	421	572	719	737	738	797
output	78									
read	770									
readln	180 557 874 1389	191 640 908 1426	202 652 919 1440	227 663 968 1449	236 750 979	245 759 988	252 779 1075	260 832 1093	269 842 1137	277 852 1152
real	82 426	135 570	168 576	291 794	293 798	336 957	340 1197	343	419	423
rewrite	1202									
round	1205	1214								
sin	610	672	678	698	701	1005	1009	1028	1037	
sqrt	319									
text	1198									
true	694	710								
trunc	174 1213	184 1222	185	195	196	889	1078	1096	1176	1204

write	179	190	201	226	235	244	251	259	268	276
	394	400	401	529	530	556	629	639	651	662
	724	725	749	758	765	767	768	769	778	826
	831	836	841	851	873	878	892	907	911	918
	967	978	987	1068	1070	1072	1074	1092	1100	1106
	1112	1136	1146	1148	1151	1157	1206	1207	1208	1209
	1210	1211	1211	1215	1216	1217	1218	1229	1231	1233
	1235	1237	1241	1246	1251	1253	1256	1258	1260	1262
	1264	1268	1273	1276	1278	1282	1289	1293	1296	1298
	1300	1302	1306	1309	1311	1315	1318	1328	1329	1345
	1347	1349	1351	1353	1355	1357	1359	1361	1363	1365
	1367	1369	1371	1373	1375	1379	1381	1383	1385	1388
	1425	1439	1448							

writeln	155	177	178	188	189	199	200	217	219	220
	221	222	223	224	225	229	230	233	234	237
	239	240	241	242	243	247	250	253	257	258
	262	263	267	270	274	275	279	280	282	321
	322	323	395	396	397	402	404	524	525	526
	531	533	549	550	551	552	553	554	555	558
	628	630	631	632	633	634	635	636	637	638
	641	643	644	645	646	647	648	649	650	653
	661	665	666	726	747	748	751	757	760	766
	776	780	809	810	811	812	813	814	815	816
	817	818	819	820	821	822	823	824	825	827
	828	829	830	837	838	845	846	854	855	879
	880	885	886	893	894	899	900	912	913	921
	922	930	931	963	964	965	966	970	971	977
	981	982	986	990	991	1069	1071	1073	1080	1081
	1087	1088	1089	1090	1091	1101	1107	1113	1116	1134
	1135	1138	1147	1149	1150	1158	1159	1164	1165	1166
	1167	1168	1169	1170	1171	1172	1177	1219	1219	1223
	1225	1230	1232	1234	1236	1238	1242	1247	1252	1254
	1257	1259	1261	1263	1265	1269	1274	1277	1279	1283
	1290	1294	1297	1299	1301	1303	1307	1310	1312	1316
	1319	1323	1324	1330	1346	1348	1350	1352	1354	1356
	1358	1360	1362	1364	1366	1368	1370	1372	1374	1376
	1377	1378	1380	1382	1384	1386	1387	1417	1418	1419
	1420	1421	1422	1423	1427	1435	1436	1437	1438	1441
	1445	1446	1447	1450	1462	1463	1464	1465	1466	

117 Variables & Operators Used 1588 Occurrences

```

0001  (*****
0002  (*)
0003  (*) SEARCH PLANNING SOFTWARE (PROGRAM #2 OF 3) (*)
0004  (*)
0005  (*) TITLE: SURFDRIF.COM (Surface Drift Algorithm) (*)
0006  (*) VERSION: 1.1 for CP/M Operating System (*)
0007  (*) DATE WRITTEN: September 1984 (*)
0008  (*)
0009  (*) DESCRIPTION: (*)
0010  (*) - User asked for last known position (coordinates) and (*)
0011  (*) date-time-group of search object (*)
0012  (*) - User asked for desired Datum position (coordinates) (*)
0013  (*) and date-time-group of search object (*)
0014  (*) - User must verify search object affected by oceanic (*)
0015  (*) drift forces (object more than 20 miles off-shore and (*)
0016  (*) in water deeper than 100 feet) (*)
0017  (*) - User asked to input reported surface winds in the (*)
0018  (*) search object's vicinity for at least the last 48 (*)
0019  (*) hours (*)
0020  (*) - User must then input the wind latitude coefficient (*)
0021  (*) directions and magnitudes for the object's latitude as (*)
0022  (*) found in the National Search and Rescue (SAR) Manual (*)
0023  (*) - Wind latitude coefficient vectors are printed back to (*)
0024  (*) the user for verification/correction (*)
0025  (*) - User asked which of the three types of drift (*)
0026  (*) uncertainty found in the National SAR Manual applies, (*)
0027  (*) then the appropriate information is requested (*)
0028  (*) - User is asked for the sea current vector and published (*)
0029  (*) reference source it was found in (*)
0030  (*) - User is asked for the total observed water current, if (*)
0031  (*) applicable (*)
0032  (*) - User is asked for the tidal current, if applicable (*)
0033  (*) - Program calculates applicable, resultant drift vector (*)
0034  (*) direction(s) and magnitude(s) and creates an "audit (*)
0035  (*) trail"/record file of program input, significant (*)
0036  (*) calculations, and output, named "SEADATA" (*)
0037  (*)
0038  (*) LICENSE: COPYRIGHT 1984 D. RICK DOUGLAS (*)
0039  (*)
0040  (*) The author makes no express or implied warranty of any (*)
0041  (*) kind with regard to this program material, including, but (*)
0042  (*) not limited to, the implied warranty of fitness for a (*)
0043  (*) particular purpose. The author shall not be liable for (*)
0044  (*) incidental or consequential damages in connection with or (*)
0045  (*) arising out of furnishing, use, or performance of this (*)
0046  (*) program. The reader MUST HAVE a solid understanding of (*)
0047  (*) search and rescue methodology before using this software (*)
0048  (*) in making decisions where human life is at risk. In fact, (*)
0049  (*) since no amount of testing can uncover 100% of program (*)
0050  (*) errors, this program is recommended for training use only. (*)
0051  (*) Prior attendance at the United States Coast Guard's (*)

```

```

0052 (* National SAR School is highly-encouraged. *)
0053 (* *)
0054 (* THIS SOFTWARE MAY BE FREELY-DISTRIBUTED *)
0055 (* PROVIDED NO FEE IS CHARGED AND *)
0056 (* THIS COPYRIGHT NOTICE IS RETAINED. *)
0057 (* ----- *)
0058 (* LANGUAGE: PASCAL *)
0059 (* USED : Borland International, TURBO.PAS, Version 2.0 *)
0060 (* ----- *)
0061 (* MODULES CALLED (Sequentially listed); (OPT) = "Optional": *)
0062 (* *)
0063 (* SurfacePosition *)
0064 (* VerifyDTG (Last Known Position) *)
0065 (* VerifyDTG (Datum Position) *)
0066 (* WindPeriods *)
0067 (* PeriodTimes *)
0068 (* DaysInMonth (OPT) *)
0069 (* DaysInMonth (OPT) *)
0070 (* DaysInMonth (OPT) *)
0071 (* InputSeaWinds *)
0072 (* WindLatCoeffs *)
0073 (* VerifyWinds *)
0074 (* WindChart *)
0075 (* WindChart (OPT) *)
0076 (* WindCurrent *)
0077 (* AddVectors *)
0078 (* AddVectors (OPT) *)
0079 (* AvgSurfaceWind *)
0080 (* AddVectors *)
0081 (* LeewayDrift *)
0082 (* DriftDirUncertain (OPT) *)
0083 (* SeaCurrent *)
0084 (* Datum *)
0085 (* WriteToDisk *)
0086 (* RecordCurrents *)
0087 (* *)
0088 (*****
0089
0090
0091
0092 program SurfaceDrift(input,output);
0093
0094 const max = 4; {Maximum number of wind current periods allowed}
0095 radians = 57.2957795; {Standard radian conversion factor}
0096
0097 var continue, {Determines if this program should be used}
0098 LatNS, {Indicates whether latitude is North or South}
0099 LongEW : char; {Indicates whether longitude is East or West}
0100
0101 DatumMonth, {Month object will be at desired Datum position}
0102 DatumYear, {Year object will be at desired Datum position}
0103 Days, {Number of days in specified month}

```

0104	DegreesLat,	{Lat known position latitude in degrees}
0105	I,J,K,	{Used as counters}
0106	N,	{Tracks number of 48-hr wind current periods used}
0107	HoursElapsed,	{Datum minus last known position time in hours}
0108	LastDay,	{Day object at last known position}
0109	LastHour,	{Hour object at last known position}
0110	LastMonth,	{Month object at last known position}
0111	LastYear,	{Year object at last known position}
0112	LeewayMethod,	{Indicates type of leeway drift uncertainty chosen}
0113	MaxDivergence,	{Maximum divergence from downwind in degrees}
0114	MaxHoursDrift,	{Maximum number of hours search object drifted}
0115	MinHoursDrift,	{Minimum number of hours search object drifted}
0116	SourceSeaCurr :integer;	{Source used to determine sea current vector}
0117		
0118	AvgWindFrom,	{Average surface wind direction from (in degrees)}
0119	AvgWindTo,	{AvgWindFrom plus-or-minus 180}
0120	DatumDateTime,	{Day/Time object will be at desired Datum position}
0121	DirSeaCurrent,	{Sea current vector direction}
0122	DirSurfWind,	{Average surface wind direction}
0123	DirTidalCurrent,	{User input Tidal Current direction}
0124	DirTotalCurrent,	{Total water current vector direction}
0125	DmaxDir,	{Maximum combined surface drift direction}
0126	DminDir,	{Minimum combined surface drift direction}
0127	DmaxDistance,	{Maximum combined surface drift distance}
0128	DminDistance,	{Minimum combined surface drift distance}
0129	LastDateTime,	{Time object was at last known position}
0130	LastLatitudeKnown,	{Last known latitude of aerospace object}
0131	LastLongitudeKnown,	{Last known longitude of aerospace object}
0132	LeewayBrg,	{Average surface wind bearing}
0133	MaxDirWindCurrent,	{Total ocean surface current bearing due to wind}
0134	MinDirWindCurrent,	{Total ocean surface current bearing due to wind}
0135	MaxLeeBrg,	{Average surface wind bearing minus MaxDivergence}
0136	MinLeeBrg,	{Average surface wind bearing plus MaxDivergence}
0137	MaxLeeDistance,	{Maximum leeway drift distance of search object}
0138	MinLeeDistance,	{Minimum leeway drift distance of search object}
0139	MaxLeeSpeed,	{Maximum leeway drift rate of search object}
0140	MinLeeSpeed,	{Minimum leeway drift rate of search object}
0141	MaxSeaCurrDist,	{Maximum sea current drift distance}
0142	MinSeaCurrDist,	{Minimum sea current drift distance}
0143	MaxTideDistance,	{User input Tidal Current maximum distance}
0144	MinTideDistance,	{User input Tidal Current minimum distance}
0145	MaxTotCurrDist,	{Total water current vector maximum distance}
0146	MinTotCurrDist,	{Total water current vector minimum distance}
0147	MaxWindCurrDist,	{Total ocean surface current distance due to wind}
0148	MinWindCurrDist,	{Total ocean surface current distance due to wind}
0149	ResultMagnitude,	{Resultant distance computed from vector addition}
0150	SeaCurrSpeed,	{Sea current vector velocity}
0151	SurfWindMagnitude,	{Average surface wind speed times affected hours}
0152	SurfWindSpeed,	{Average surface wind in nautical miles per hour}
0153	TotCurrSpeed : real;	{Total water current vector velocity}
0154		
0155	No,	{'N,n' = Legal "No" characters}

```

0156      Yes,                                ('Y,y' = Legal "Yes" characters)
0157      ValidAnswers : set of char; ('Y,y,N,n' = Legal "Yes" and "No" characters)
0158
0159      SeaDat      : text; {External record file of program input/output data}
0160
0161      {Indicates the 1 to 6 hours of wind effect in each 48-hour period of }
0162      {ocean surface current. }
0163
0164      HoursWindEffect : array[1..max] of integer;
0165
0166
0167      {Indicates the velocity contributions of each 6-hour wind block times }
0168      {the number of hours of wind effect in that block to calculate average }
0169      {surface wind. }
0170
0171      VelocityComponent : array[1..max] of real;
0172
0173
0174      {Indicates the wind current latitude coefficient vector directions }
0175      {(position 1) and magnitudes (position 2) from the National SAR Manual.}
0176
0177      WindCoeffs      : array[1..8,1..3] of real;
0178
0179
0180      {Wind vectors affecting ocean surface current over time: First position}
0181      {indicates the number of 48-hour periods; Second position indicates }
0182      {where the reported, 6-hour wind vector fits amongst the 8 times given }
0183      {in each 48-hour period; and, Third position indicates the days (1), }
0184      {hours (2), directions (3), and velocities (4) of each 6-hour wind }
0185      {vector, as well as the sum of these vectors and the Wind Current }
0186      {Latitude Coefficient vectors directions (5) and magnitudes (6). }
0187
0188      WindsOverTime   : array[1..max,1..8,1..6] of real;
0189
0190      {The total contribution (sum) of WindsOverTime and WindCoeffs vector }
0191      {directions and velocities for the eight intervals per period. }
0192
0193      PeriodVectors : array[1..max,1..3] of real;
0194
0195      {*****}
0196      {Writes out a specified number of blank lines; can be used to clear screen. }
0197
0198      procedure writelns (lines : integer);
0199
0200      var count : integer;
0201
0202      begin
0203          count := 0;
0204          while lines > count do
0205              begin
0206                  writelns;
0207                  count := count + 1

```

```

0208     end
0209 end;
0210
0211 {*****}
0212 {Verifies legitimate date/time/group data input.}
0213
0214 procedure VerifyDTG (var DateTime : real);
0215
0216     var Temp,           {Used as a temporary computation variable}
0217         TempDate,       {Used as temporary date variable}
0218         TempHour,       {Used as temporary hour variable}
0219         TempMinutes : real; {Used as temporary minutes variable}
0220
0221     begin {VerifyDTG}
0222         repeat {until correct date/time format input}
0223
0224             {Verify the day is between 1 and 31}
0225             TempDate := trunc( DateTime/10000 );
0226             if (TempDate < 1) or (TempDate > 31) then
0227                 begin {TempDate not between 1 and 31}
0228                     writeln;
0229                     writeln('You have incorrectly entered the date. Try again!');
0230                     write('Re-enter Z DTG= ');
0231                     readln(DateTime)
0232                 end; {TempDate not between 1 and 31}
0233
0234             {Verify the hour is between 0000 and 2400}
0235             Temp := (DateTime/10000 - (trunc(DateTime/10000)) ) * 100;
0236             TempHour := trunc(Temp);
0237             if (TempHour < 0) or (TempHour > 23) then
0238                 begin {TempHour not between 0 and 23}
0239                     writeln;
0240                     writeln('You have incorrectly entered the hour. Try again!');
0241                     write('Re-enter Z DTG= ');
0242                     readln(DateTime)
0243                 end; {TempHour not between 0 and 23}
0244
0245             {Verify there are no minutes, e.g. minutes = 0}
0246             Temp := (DateTime/100 - (trunc(DateTime/100)) ) * 100;
0247             TempMinutes := trunc(Temp);
0248             if TempMinutes < 0 then
0249                 begin {TempMinutes not 0}
0250                     writeln;
0251                     writeln('Please enter the date/time to the nearest hour. Try again!');
0252                     write('Re-enter Z DTG= ');
0253                     readln(DateTime)
0254                 end {TempMinutes not 0}
0255
0256             until (TempDate >= 1) and (TempDate <= 31) and (TempHour >= 0) and
0257                 (TempHour <= 23) and (TempMinutes = 0)
0258
0259         end; {VerifyDTG}

```

```

0260
0261 {*****}
0262 {Asks user for the last known and the desired Datum times and positions of a }
0263 {search object on the ocean's surface. }
0264
0265 procedure SurfacePosition;
0266
0267 var Temp : real;    {Used as a temporary computation variable}
0268
0269 begin {SurfacePosition}
0270
0271     writeln;
0272     repeat {until valid response}
0273         writeln('Please enter the number for the month the search object was');
0274         writeln('at the last known position:');
0275         writeln;
0276         writeln(' 1 = Jan          4 = Apr          7 = Jul          10 = Oct');
0277         writeln(' 2 = Feb          5 = May          8 = Aug          11 = Nov');
0278         writeln(' 3 = Mar          6 = Jun          9 = Sep          12 = Dec');
0279         writeln;
0280         write('LAST KNOWN POSITION MONTH= ');
0281         readln(LastMonth);
0282         if (LastMonth < 1) or (LastMonth > 12) then
0283             writeln('Incorrect number entered. Please try again!');
0284         writeln
0285     until (LastMonth >= 1) and (LastMonth <= 12);
0286
0287     writeln('Please enter the year the search object was at the last known');
0288     writeln('position (e.g., 1985, 1986, etc.)');
0289     write('LAST KNOWN POSITION YEAR= ');
0290     readln(LastYear);
0291     writeln;
0292
0293     CLRSCR;
0294     writeln('Please enter the day-hour-minute (TO THE NEAREST HOUR) the');
0295     writeln('search object was at the last known position. Enter it in');
0296     writeln('Z DTG (Zulu Date-Time-Group) format. For example: 9:37 PM,');
0297     writeln('August 4, should appear as 042200 Greenwich-Mean Time (Z=0).');
0298     write(2);
0299     writeln('If the number of hours of search object drift is uncertain,');
0300     writeln('you must run this program twice (as mentioned earlier). On');
0301     writeln('the first run enter here the time the SHORTER drift period');
0302     writeln('started (LATER than the last known position time). On the');
0303     writeln('second run, or if two runs are not applicable, enter here the');
0304     writeln('the LKP time as requested above');
0305     write('LAST KNOWN POSITION TIME (Z DTG)= ');
0306     readln(LastDateTime);
0307     VerifyDTG(LastDateTime);
0308
0309     Temp := LastDateTime/10000;
0310     LastDay := trunc(Temp);
0311     LastHour := (round ((LastDay - Temp) * -100) ) * 100; {nearest hundred}

```

```

0312
0313 CLRSR;
0314 repeat (until valid latitude)
0315     writeln('Was search object's last known latitude north or south?');
0316     write(' (Enter N or S)      Answer= ');
0317     readln(LatNS);
0318     writeln
0319 until (LatNS = 'N') or (LatNS = 'n') or (LatNS = 'S') or (LatNS = 's');
0320 if (LatNS = 'N') or (LatNS = 'n') then LatNS := 'N' else LatNS := 'S';
0321
0322 repeat (until valid latitude)
0323     writeln('Please enter the search object's last known latitude ');
0324     writeln(' (For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)');
0325     write('LATITUDE = ');
0326     readln(LastLatitudeKnown);
0327     if (LastLatitudeKnown < 0) or (LastLatitudeKnown > 90) then
0328         writeln('Input latitude must be between 0-90. Try again!');
0329     writeln
0330 until (LastLatitudeKnown = 0) and (LastLatitudeKnown <= 90);
0331 DegreesLat := trunc(LastLatitudeKnown);
0332
0333 repeat (until valid longitude)
0334     writeln('Was search object's last known longitude east or west?');
0335     write(' (Enter E or W)      Answer= ');
0336     readln(LongEW);
0337     writeln
0338 until (LongEW = 'E') or (LongEW = 'e') or (LongEW = 'W') or (LongEW = 'w');
0339 if (LongEW = 'E') or (LongEW = 'e') then LongEW := 'E' else LongEW := 'W';
0340
0341 repeat (until valid longitude)
0342     writeln('Please enter the search object's last known longitude ');
0343     writeln(' (For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)');
0344     write('LONGITUDE = ');
0345     readln(LastLongitudeKnown);
0346     if (LastLongitudeKnown < 0) or (LastLongitudeKnown > 180) then
0347         writeln('Input longitude must be between 0-180. Try again!');
0348     writeln
0349 until (LastLongitudeKnown = 0) and (LastLongitudeKnown <= 180);
0350
0351 CLRSR;
0352 writeln('Now you will be asked to enter the year, month, and date/time for');
0353 writeln('the desired Datum position, which MUST be later than the time you');
0354 writeln('just entered for the last known position!');
0355 writeln;
0356
0357 repeat (until valid response)
0358     writeln('Please enter the desired Datum month:');
0359     writeln;
0360     writeln(' 1 = Jan      4 = Apr      7 = Jul      10 = Oct');
0361     writeln(' 2 = Feb      5 = May      8 = Aug      11 = Nov');
0362     writeln(' 3 = Mar      6 = Jun      9 = Sep      12 = Dec');
0363     writeln;

```

```

0364     write('DATUM MONTH= ');
0365     readln(DatumMonth);
0366     if (DatumMonth ( 1) or (DatumMonth ) 12) then
0367         writeln('Incorrect number entered. Please try again!');
0368     writeln
0369     until (DatumMonth )= 1) and (DatumMonth (= 12);
0370
0371     writeln('Please enter the desired Datum year (e.g., 1985, 1986, etc.)');
0372     write('DATUM YEAR= ');
0373     readln(DatumYear);
0374     writeln;
0375
0376     writeln;
0377     write('Please enter the desired Datum day-hour-minute ');
0378     writeln(' (TO THE NEAREST HOUR).');
0379     writeln('Enter it in Z DTG (Zulu Date-Time-Group) format. For example:');
0380     write('9:37 PM, August 4, should appear as 042200 ');
0381     writeln('Greenwich-Mean Time (Z=0).');
0382     write('DATUM TIME (Z DTG)= ');
0383     readln(DatumDateTime);
0384     VerifyDTG(DatumDateTime);
0385     writeln;
0386
0387     Temp := DatumDateTime/10000
0388
0389 end; {SurfacePosition}
0390
0391 {*****}
0392 {Given the last known surface position time, and the hours elapsed from then }
0393 {until the desired Datum time, this procedure determines the number of }
0394 {48-hour wind-current-effect periods, and the number of hours (1 to 6) in }
0395 {each period, required to calculate the average ocean surface current caused }
0396 {by the wind. }
0397
0398 procedure WindPeriods;
0399
0400 var HoursRemaining : integer; {Elapsed hours minus those used in calculations}
0401
0402 begin {WindPeriods}
0403
0404     {Find number of hours in first 48-hour period}
0405     if (LastHour )= 0000) and (LastHour ( 0300) then
0406         HoursWindEffect[1] := 3 - round(LastHour/100);
0407     if (LastHour )= 0300) and (LastHour ( 0900) then
0408         HoursWindEffect[1] := 9 - round(LastHour/100);
0409     if (LastHour )= 0900) and (LastHour ( 1500) then
0410         HoursWindEffect[1] := 15 - round(LastHour/100);
0411     if (LastHour )= 1500) and (LastHour ( 2100) then
0412         HoursWindEffect[1] := 21 - round(LastHour/100);
0413     if LastHour )= 2100 then
0414         HoursWindEffect[1] := (24 - round(LastHour/100) ) + 3;
0415

```

```

0416 N := 1; {initialize number of 48-hour periods}
0417 HoursRemaining := HoursElapsed - HoursWindEffect[1];
0418 if HoursRemaining < 0 then
0419   begin {More 48-hour periods are required}
0420
0421     repeat {until HoursRemaining = 0}
0422       N := N + 1;
0423       if HoursRemaining >= 6 then
0424         begin {More than six hours remain between current and Datum hour}
0425           HoursWindEffect[N] := 6;
0426           HoursRemaining := HoursRemaining - 6
0427         end {More than six hours remain between current and Datum hour}
0428       else
0429         begin {One to five hours remain between current and Datum hour}
0430           HoursWindEffect[N] := HoursRemaining;
0431           HoursRemaining := 0
0432         end {One to five hours remain between current and Datum hour}
0433       until HoursRemaining = 0
0434     end {More 48-hour periods are required}
0435   end; {WindPeriods}
0436
0437 {*****}
0438 {Given the month, determines the number of days in that month, including a }
0439 {check to see if that month is a leap-year-February with 29 days. }
0440
0441 procedure DaysInMonth (TempMonth : integer);
0442
0443   var LeapYearCheck,      {If zero, specified year is a leap year}
0444       TempYearDiv      : real; {Result of specified year divided by 4}
0445
0446   begin {DaysInMonth}
0447     if TempMonth = 2 {February} then
0448       begin {Leap year check}
0449         TempYearDiv := LastYear/4;
0450         LeapYearCheck := TempYearDiv - trunc(TempYearDiv);
0451         if LeapYearCheck = 0 then TempMonth := 13 {Leap year February}
0452       end; {Leap year check}
0453
0454     case TempMonth of
0455       1 : Days := 31;
0456       2 : Days := 28;
0457       3 : Days := 31;
0458       4 : Days := 30;
0459       5 : Days := 31;
0460       6 : Days := 30;
0461       7 : Days := 31;
0462       8 : Days := 31;
0463       9 : Days := 30;
0464      10 : Days := 31;
0465      11 : Days := 30;
0466      12 : Days := 31;
0467      13 : Days := 29 {(- Leap year February)}

```

```

0468     end {case of number of days in specified month}
0469
0470 end; {DaysInMonth}
0471
0472 {*****}
0473 {Determines the day and hour for which wind directions and velocities are }
0474 {required in the eight intervals of each period to calculate each period's }
0475 {ocean surface wind current component vector. }
0476
0477 procedure PeriodTimes;
0478
0479 var I,J,                {Used as counters}
0480     MonthBefore : integer; {The month before the last known position month}
0481
0482 begin {PeriodTimes}
0483
0484     {Find day and time for period #1, interval #1}
0485     WindsOverTime[1,1,1] := LastDay; {initialize at last known position day}
0486     if LastHour = 0000 {midnight} then WindsOverTime[1,1,2] := 0000;
0487     if (LastHour) = 0100 and (LastHour <= 0600) then
0488         WindsOverTime[1,1,2] := 0600;
0489     if (LastHour) = 0700 and (LastHour <= 1200) then
0490         WindsOverTime[1,1,2] := 1200;
0491     if (LastHour) = 1300 and (LastHour <= 1800) then
0492         WindsOverTime[1,1,2] := 1800;
0493
0494     if (LastHour) = 1900 and (LastHour < 2400) then
0495     {Find next day and start with time = 0000}
0496     begin {LastHour = 7-11 P.M.}
0497         DaysInMonth(LastMonth);
0498         if LastDay < Days then
0499             begin {Last known position-day is NOT the last day of the month}
0500                 WindsOverTime[1,1,1] := LastDay + 1;
0501                 WindsOverTime[1,1,2] := 0000
0502             end {Last known position-day is NOT the last day of the month}
0503         else
0504             begin {Last known position-day is the last day of the month}
0505                 WindsOverTime[1,1,1] := 1;
0506                 WindsOverTime[1,1,2] := 0000
0507             end {Last known position-day is the last day of the month}
0508         end; {LastHour = 7-11 P.M.}
0509
0510     for J := 2 to 8 do
0511     begin {Find days/times of remaining 7, 6-hr intervals in the 48-hr-period}
0512
0513         if WindsOverTime[1,J-1,2] > 0000 {not equal to midnight} then
0514             begin {this period's previous time = 6 A.M. or P.M. or 12 P.M.}
0515
0516                 {Next interval's date equals the same as the previous interval's date}
0517                 WindsOverTime[1,J,1] := WindsOverTime[1,J-1,1];
0518
0519                 {Next interval's hour equals last interval's hour minus six hours}

```

```

0520      WindsOverTime[I,J,2] := WindsOverTime[I,J-1,2] - 0600
0521
0522      end {this period's previous time = 6 A.M. or P.M. or 12 P.M.}
0523      else
0524      begin {this period's previous time = midnight}
0525          if WindsOverTime[I,J-1,1] > 1 then {Day of month is 2nd through last}
0526              begin {Set period's interval time to 1800 of previous day}
0527                  WindsOverTime[I,J,1] := WindsOverTime[I,J-1,1] - 1;
0528                  WindsOverTime[I,J,2] := 1800
0529              end {Set period's interval time to 1800 of previous day}
0530              else {First day of month}
0531                  begin {Determine previous month's last day and set at 6 P.M.}
0532                      if LastMonth > 1 then MonthBefore := LastMonth - 1
0533                          else MonthBefore := 12;
0534                          DaysInMonth(MonthBefore);
0535                          WindsOverTime[I,J,1] := Days;
0536                          WindsOverTime[I,J,2] := 1800
0537                      end {Determine previous month's last day and set at 6 P.M.}
0538                  end {this period's previous time = midnight}
0539              end; {Find days/times of remaining 7, 6-hr intervals in the 48-hr-period}
0540
0541          if N > 1 then
0542              begin {More than one period}
0543                  I := 1; {initialize}
0544
0545                  repeat {until I = N}
0546                      I := I + 1;
0547
0548                      {Find days and times of intervals in each period}
0549                      WindsOverTime[I,1,1] := WindsOverTime[I-1,1,1]; {initialize}
0550                      if WindsOverTime[I-1,1,2] = 0000 then WindsOverTime[I,1,2] := 0600;
0551                      if WindsOverTime[I-1,1,2] = 0600 then WindsOverTime[I,1,2] := 1200;
0552                      if WindsOverTime[I-1,1,2] = 1200 then WindsOverTime[I,1,2] := 1800;
0553                      if WindsOverTime[I-1,1,2] = 1800 then
0554                          begin {Find next day and start with 0000}
0555                              DaysInMonth(LastMonth);
0556
0557                              {if not last day of month, add a day}
0558                              if WindsOverTime[I-1,1,1] < Days then
0559                                  WindsOverTime[I,1,1] := WindsOverTime[I-1,1,1] + 1
0560
0561                              {if last day of month, set to first day (of next month)}
0562                              else WindsOverTime[I,1,1] := 1;
0563
0564                              WindsOverTime[I,1,2] := 0000
0565                          end; {Find next day and start with 0000}
0566
0567                      for J := 2 to 8 do
0568                          begin {fill in days and times of each period's other seven intervals}
0569                              WindsOverTime[I,J,1] := WindsOverTime[I-1,J-1,1];
0570                              WindsOverTime[I,J,2] := WindsOverTime[I-1,J-1,2]
0571                          end {fill in days and times of each period's other seven intervals}

```

```

0572         until I = N
0573     end {More than one period}
0574 end; {PeriodTimes}
0575
0576 {*****}
0577 {Queries user for ocean surface wind directions and velocities.      }
0578
0579 procedure InputSeaWinds;
0580
0581 var J : integer; {Used as a counter}
0582
0583 begin {InputSeaWinds}
0584     CLRSCR;
0585
0586     for J := 8 downto 1 do
0587         begin {Enter individual ocean surface wind vectors}
0588             write('Please enter the ocean surface wind direction and ');
0589             if WindsOverTime[1,J,1] < 10 then
0590                 write('velocity at 0',WindsOverTime[1,J,1]:1:0);
0591             else write('velocity at ',WindsOverTime[1,J,1]:2:0);
0592             if WindsOverTime[1,J,2] = 0000 then
0593                 writeln('000',WindsOverTime[1,J,2]:1:0,'Z:');
0594             else if WindsOverTime[1,J,2] = 0600 then
0595                 writeln('0',WindsOverTime[1,J,2]:3:0,'Z:');
0596             else writeln(WindsOverTime[1,J,2]:4:0,'Z:');
0597
0598             repeat {until degrees = 0 to 360}
0599                 write('WIND DIRECTION = ');
0600                 readln(WindsOverTime[1,J,3]);
0601                 until (WindsOverTime[1,J,3] = 0) and (WindsOverTime[1,J,3] (= 360);
0602
0603             repeat {until velocity = 0 to 99}
0604                 write('WIND VELOCITY = ');
0605                 readln(WindsOverTime[1,J,4]);
0606                 writeln
0607                 until (WindsOverTime[1,J,4] = 0) and (WindsOverTime[1,J,4] (= 99)
0608             end; {Enter individual ocean surface wind vectors}
0609
0610         if N > 1 then {Checks for more than one 48-hour wind period}
0611             begin {more than one period}
0612                 I := 1;
0613                 {Input wind vectors from last interval in each period}
0614                 repeat {until I = N}
0615                     I := I + 1;
0616                     write('Please enter the ocean surface wind direction and ');
0617                     write('velocity at ',WindsOverTime[I,1,1]:2:0);
0618                     if WindsOverTime[I,1,2] = 0000 then
0619                         writeln('000',WindsOverTime[I,1,2]:1:0,'Z:');
0620                     else if WindsOverTime[I,1,2] = 0600 then
0621                         writeln('0',WindsOverTime[I,1,2]:3:0,'Z:');
0622                     else writeln(WindsOverTime[I,1,2]:4:0,'Z:');
0623

```

```

0624         repeat {until degrees = 0 to 360}
0625             write('WIND DIRECTION = ');
0626             readln(WindsOverTime[I,1,3]);
0627         until (WindsOverTime[I,1,3] >= 0) and
0628             (WindsOverTime[I,1,3] <= 360);
0629
0630         repeat {until velocity = 0 to 99}
0631             write('WIND VELOCITY = ');
0632             readln(WindsOverTime[I,1,4]);
0633             writeln
0634         until (WindsOverTime[I,1,4] >= 0) and
0635             (WindsOverTime[I,1,4] <= 99);
0636
0637         for J := 2 to 8 do
0638             begin {fill in wind vectors of each period's last 7 intervals}
0639                 WindsOverTime[I,J,3] := WindsOverTime[I-1,J-1,3];
0640                 WindsOverTime[I,J,4] := WindsOverTime[I-1,J-1,4]
0641             end {fill in wind vectors of each period's last 7 intervals}
0642         until I = N
0643     end {more than one period}
0644 end; {InputSeaWinds}
0645
0646 {*****}
0647 {Prints keyboard-input wind coefficient directions and velocities on the}
0648 {video screen for user verification.}
0649
0650 procedure WindChart;
0651
0652 var I : integer; {Used as a counter}
0653
0654 begin {WindChart}
0655     write('Wind Current Latitude Coefficients for ',DegreesLat:2,'-degrees ');
0656     if LatNS = 'N' then writeln('North are:') else writeln('South are:');
0657     writeln;
0658
0659     for I := 1 to 8 do
0660         begin {for I = 1 to number of individual wind vectors}
0661             write(' ',I:1,'. Period #',I:1,' ',WindCoeffs[I,1]:3:0);
0662             writeln(' / ',WindCoeffs[I,2]:4:3)
0663         end; {for I = 1 to number of individual wind vectors}
0664
0665     writeln
0666 end; {WindChart}
0667
0668 {*****}
0669 {Allows user to change ocean surface wind directions and velocities entered }
0670 {in procedure InputSeaWinds.}
0671
0672 procedure VerifyWinds;
0673
0674 var WindCheck : char; {Used to verify input wind data}
0675     J : integer; {Used as a counter}

```

```

0676      WindError : integer; {Used to indicate which input wind data is in error}
0677
0678      begin {VerifyWinds}
0679          WindCheck := 'Y'; WindError := -1; {initialize}
0680
0681          {Verify input wind directions and velocities.}
0682          WindChart; {Prints current wind data on video screen}
0683
0684          repeat {until WindCheck = valid response}
0685              writeln;
0686              writeln('Are these altitudes, wind directions, and wind');
0687              write('velocities all correct? (y/n) ');
0688              readln(WindCheck);
0689              writeln
0690          until (WindCheck in ValidAnswers);
0691
0692          while (WindCheck in No) do
0693              begin {while WindCheck = NO}
0694                  repeat {until WindError = 0 to 20}
0695                      writeln('Which line is in error?');
0696                      write('Enter number, or zero for none => ');
0697                      readln(WindError);
0698                      writeln
0699                  until (WindError = 0.0) and (WindError < 21.0);
0700
0701                  if WindError > 0.0 then
0702                      begin {if an input line is identified as being in error}
0703                          write('Please enter altitude, wind direction ');
0704                          writeln('and velocity to replace:');
0705                          write(WindCoeffs[WindError,1]:4:0,' ');
0706                          write(WindCoeffs[WindError,2]:4:0);
0707                          write(' ',WindCoeffs[WindError,3]:4:0,' => ');
0708                          for J := 1 to 2 do read(WindCoeffs[WindError,J]);
0709                          CLRSCR
0710                      end; {if an input line is identified as being in error}
0711
0712                      WindChart; {Prints current wind data on video screen}
0713
0714                      writeln;
0715                      repeat {until valid WindCheck response}
0716                          write('Are all other lines correct? (y/n) ');
0717                          readln(WindCheck);
0718                          writeln
0719                      until (WindCheck in ValidAnswers)
0720
0721                  end; {while WindCheck = NO}
0722                  CLRSCR
0723              end; {VerifyWinds}
0724
0725          {*****}
0726          {When given X and Y components this procedure calculates resultant vector    }
0727          {bearing (AvgWindFrom) and magnitude (ResultMagnitude).                    }

```

```

0728
0729 procedure AddVectors (var Xcomponent, Ycomponent : real);
0730
0731 var TempAngle : real; {Temporary calculation variable}
0732
0733 begin {AddVectors}
0734
0735     {Find resultant angle (in degrees) uncorrected for compass position.}
0736     TempAngle := arctan(Xcomponent/Ycomponent) * radians;
0737
0738     {Find bearing resulting from TempAngle corrected for compass position.}
0739     if (Xcomponent > 0) and (Ycomponent > 0) then AvgWindFrom := TempAngle;
0740     if ((Xcomponent < 0) and (Ycomponent < 0)) or
0741         ((Xcomponent < 0) and (Ycomponent > 0)) then
0742         if (TempAngle + 180) > 360 then
0743             AvgWindFrom := TempAngle - 180
0744         else AvgWindFrom := TempAngle + 180;
0745     if (Xcomponent < 0) and (Ycomponent < 0) then
0746         if (TempAngle + 360) > 360 then
0747             AvgWindFrom := TempAngle - 360
0748         else AvgWindFrom := TempAngle + 360;
0749
0750     {Convert AvgWindFrom to AvgWindTo bearing.}
0751     if (AvgWindFrom + 180) > 360 then
0752         AvgWindTo := AvgWindFrom - 180
0753     else AvgWindTo := AvgWindFrom + 180;
0754
0755     {Resultant bearing magnitude is the square root of the sum of}
0756     { the squared components.}
0757     ResultMagnitude := sqrt(Xcomponent * Xcomponent + Ycomponent * Ycomponent);
0758
0759     writeln('Xcomponent = ', Xcomponent:5:3);
0760     writeln('Ycomponent = ', Ycomponent:5:3);
0761     writeln
0762 end; {AddVectors}
0763
0764 {*****}
0765 {Queries user for Wind Latitude Coefficient Vector directions and velocities }
0766 {from National SAR Manual tables. }
0767
0768 procedure WindLatCoeffs;
0769
0770 var I : integer; {Used as a counter}
0771
0772 begin {WindLatCoeffs}
0773     if DegreesLat >= 5 then
0774         begin {Last known position's latitude >= 5 degrees}
0775             CLRSCR;
0776             write('Refer to the latitude coefficient-vectors table for the ');
0777             if LatNS = 'N' then writeln('Northern') else writeln('Southern');
0778             write('latitudes (approximately page ');
0779             if LatNS = 'N' then writeln('8-16c) in the National SAR Manual.')

```

```

0780     else writeln('8-16d in the National SAR Manual.');
```

```

0781     write('Find the degrees-latitude column nearest to ',DegreesLat:2);
0782     write('-degrees-');
0783     if LatNS = 'N' then writeln('North.') else writeln('South.');
```

```

0784     write('You will now be asked to enter the directions and ');
0785     writeln('magnitudes for');
0786     writeln('each of the eight periods appearing under that column.');
```

```

0787     writeln;
0788
0789     for I := 1 to 8 do
0790     begin {Input applicable latitude coefficient vectors for the 8 periods}
0791
0792         repeat {until valid compass direction}
0793             write('PERIOD #',I:1,' DIRECTION = ');
0794             readln(WindCoeffs[I,1])
0795         until (WindCoeffs[I,1] >= 0.0) and (WindCoeffs[I,1] <= 360.0);
0796
0797         repeat {until valid compass direction}
0798             write('PERIOD #',I:1,' MAGNITUDE = ');
0799             readln(WindCoeffs[I,2])
0800         until (WindCoeffs[I,2] >= 0.003) and
0801             (WindCoeffs[I,2] <= 0.03);
0802
0803         writeln
0804     end; {Input applicable latitude coefficient vectors for the 8 periods}
0805     CLRSCR;
0806     VerifyWinds
0807 end {Last known position's latitude >= 5 degrees}
0808
0809 else
0810 {If last known position's latitude is less than 5-degrees, then the eight}
0811 {Wind Current Latitude Coefficient directions remain at their initialized}
0812 {value of zero, and their magnitudes are set to one. This is done to }
0813 {ensure each periods overall wind current vector is unaffected until }
0814 {later, when 5% of their velocity value will be determined as the total }
0815 {wind current vector of all combined periods as directed in the National }
0816 {SAR Manual, approximately pages 8-16b, 8-16c, and 8-16d. }
0817
0818     for I := 1 to 8 do WindCoeffs[I,2] := 1
0819
0820 end; {WindLatCoeffs}
0821
0822 {*****}
0823 {Uses vector addition to calculate the resultant wind current vector }
0824 {direction and magnitude for each period and for the sum of all period }
0825 {vectors. }
0826
0827 procedure WindCurrent;
0828
0829 var I,J      : integer; {Used as counters}
0830     PrgRads, {Individual bearings converted to radians}
0831     Xcomponent, {Horizontal component of individual vectors}

```

```

0832     Ycomponent : real;    {Vertical component of individual vectors}
0833
0834 begin {WindCurrent}
0835     for I := 1 to N do
0836     begin {repeat calculations for each period}
0837         Xcomponent := 0.0; Ycomponent := 0.0; {initialize}
0838
0839         for J := 1 to 8 do
0840         begin {find total contribution of WindsOverTime and WindCoeffs}
0841             WindsOverTime[I,J,5] := WindsOverTime[I,J,3] + WindCoeffs[J,1];
0842             WindsOverTime[I,J,6] := WindsOverTime[I,J,4] * WindCoeffs[J,2];
0843             BrgRads := WindsOverTime[I,J,5] / radians;
0844             Xcomponent := Xcomponent + sin(BrgRads) * WindsOverTime[I,J,6];
0845             Ycomponent := Ycomponent + cos(BrgRads) * WindsOverTime[I,J,6];
0846         end; {find total contribution of WindsOverTime and WindCoeffs}
0847
0848         writeln('Period #',I:1);
0849         AddVectors(Xcomponent,Ycomponent);
0850         PeriodVectors[I,1] := AvgWindFrom;
0851         PeriodVectors[I,2] := ResultMagnitude;
0852         PeriodVectors[I,3] := PeriodVectors[I,2] * HoursWindEffect[I]
0853     end; {repeat calculations for each period}
0854
0855     if N = 1 then {Only one 48-hour wind period}
0856     begin {Resultant wind current vector = resultant period #1 vector}
0857         MaxDirWindCurrent := PeriodVectors[1,1];
0858         MaxWindCurrDist := PeriodVectors[1,3]
0859     end {Resultant wind current vector = resultant period #1 vector}
0860
0861     else
0862     begin {There are more than one 48-hour wind periods}
0863         Xcomponent := 0.0; Ycomponent := 0.0; {re-initialize}
0864         for I := 1 to N do
0865         begin {Find each PeriodVector's Xcomponent and Ycomponent}
0866             BrgRads := PeriodVectors[I,1] / radians;
0867             Xcomponent := Xcomponent + sin(BrgRads) * PeriodVectors[I,3];
0868             Ycomponent := Ycomponent + cos(BrgRads) * PeriodVectors[I,3]
0869         end; {Find each PeriodVector's Xcomponent and Ycomponent}
0870
0871         writeln('Total Wind Current:');
0872         AddVectors(Xcomponent,Ycomponent);
0873         MaxDirWindCurrent := AvgWindFrom;
0874         MaxWindCurrDist := ResultMagnitude
0875     end; {There are more than one 48-hour wind periods}
0876
0877     {if last known position's latitude < 5-degrees, then only 5% of the total}
0878     {downwind drift distance applies as referenced above from the National }
0879     {SAR Manual. }
0880     if DegreesLat < 5 then MaxWindCurrDist := 0.05 * MaxWindCurrDist;
0881
0882     write('HIT RETURN (Once or twice, as necessary) TO CONTINUE');
0883     readln(continue);

```

```

0884 CLRSCR;
0885
0886 write('Total Wind Current Direction is ',MaxDirWindCurrent:3:0,'-degrees ');
0887 writeln('for ',MaxWindCurrDist:4:2,' nautical miles.');
```

0888 writeln(2);

```

0889     writeln('If the number of hours of search object drift is uncertain, you');
0890     writeln('must run this program twice (as mentioned earlier). On the first');
0891     writeln('run, RECORD this Total Wind Current vector direction and distance');
0892     writeln('for input during the second run.');
```

0893 writeln;

```

0894     writeln('If this is the second run, please now ENTER the previously-');
0895     writeln('recorded Wind Current vector over the shorter drift period');
0896     writeln('(If not applicable, enter a heading of 361)');
0897     repeat {until valid compass heading or 361}
0898         write('WIND CURRENT DIRECTION (from Run #1) = ');
0899         readln(MinDirWindCurrent)
0900     until (MinDirWindCurrent >= 0) and (MinDirWindCurrent <= 361);
0901
0902     if MinDirWindCurrent < 361 then
0903         repeat {until valid response}
0904             write('WIND CURRENT DISTANCE (from Run #1) = ');
0905             readln(MinWindCurrDist)
0906         until MinWindCurrDist >= 0
0907     end; {WindCurrent}
0908
0909
0910 {*****}
0911 {Determines the average surface (leeway) wind blowing on the search object's }
0912 {exposed area above the ocean's surface. }
0913
0914 procedure AvgSurfaceWind;
0915
0916 var I             : integer; {Used as counter}
0917     BrgRads,         {Individual bearings converted to radians}
0918     Xcomponent,       {Horizontal component of individual vectors}
0919     Ycomponent : real;    {Vertical component of individual vectors}
0920
0921 begin {AvgSurfaceWind}
0922     Xcomponent := 0.0; Ycomponent := 0.0; {re-initialize}
0923
0924     for I := 1 to N do
0925         begin {Find each 6-hour wind block's Xcomponent and Ycomponent}
0926             VelocityComponent[I] := WindsOverTime[I,1,4] * HoursWindEffect[I];
0927             BrgRads := WindsOverTime[I,1,3] / radians;
0928             Xcomponent := Xcomponent + sin(BrgRads) * VelocityComponent[I];
0929             Ycomponent := Ycomponent + cos(BrgRads) * VelocityComponent[I];
0930         end; {Find each 6-hour wind block's Xcomponent and Ycomponent}
0931
0932     CLRSCR;
0933     writeln('Average Surface Wind:');
0934     AddVectors(Xcomponent,Ycomponent);
0935     LeewayBrg := AvgWindTo;
```

```

0936   DirSurfWind := AvgWindFrom;
0937   SurfWndMagnitude := ResultMagnitude;
0938   SurfWndSpeed := SurfWndMagnitude / HoursElapsed;
0939   write('Average Surface Wind Direction ');
0940   write(DirSurfWind:3:0, '-degrees True at ');
0941   writeln(SurfWndSpeed:4:2, ' knots. ');
0942   writeln(2);
0943   writeln('If the number of hours of search object drift is uncertain, you');
0944   writeln('must run this program twice (as mentioned earlier). On the first');
0945   writeln('run, RECORD this Average Surface Wind vector direction and');
0946   writeln('speed for use in calculating the minimum leeway speed during the');
0947   writeln('second program run. Then, ABORT this program run, CLEAR your');
0948   writeln('microcomputer's memory, and RERUN the entire program');
0949   writeln(6);
0950   write('HIT RETURN (Once or twice, as necessary) TO CONTINUE ');
0951   readln(continue)
0952
0953 end; {AvgSurfaceWind}
0954
0955 {*****}
0956 {Used by procedure Leeway (if drift rate and time are known with certainty) }
0957 {to calculate minimum and maximum leeway drift direction. }
0958
0959 procedure DriftDirUncertain;
0960
0961 begin {Directional Drift Uncertainty}
0962
0963   writeln('Refer to the Leeway Speed Graph in the National SAR Manual');
0964   writeln(' (approximately page 8-15). Please enter the search object's');
0965   writeln('maximum expected degrees-divergence from the downwind vector:');
0966   repeat (until ) 0;
0967     write('MAX EXPECTED DIVERGENCE = ');
0968     readln(MaxDivergence)
0969   until (MaxDivergence) = 0 and (MaxDivergence) (= 60);
0970
0971   if (LeewayBrq - MaxDivergence) < 0 then {adjusts for compass limits}
0972     MinLeeBrq := (LeewayBrq + 360.0) - MaxDivergence
0973   else MinLeeBrq := LeewayBrq - MaxDivergence;
0974
0975   if (LeewayBrq + MaxDivergence) > 360 then {adjusts for compass limits}
0976     MaxLeeBrq := (LeewayBrq + MaxDivergence) - 360.0
0977   else MaxLeeBrq := LeewayBrq + MaxDivergence;
0978
0979   writeln;
0980   writeln('Refer to the Leeway Speed Formulae in the National SAR Manual');
0981   writeln(' (approximately page 8-13). Please enter the leeway speed at');
0982   writeln('which the search object drifts:');
0983   repeat (until ) 0;
0984     write('LEEWAY SPEED = ');
0985     readln(MaxLeeSpeed)
0986   until MaxLeeSpeed > 0;
0987

```

```

0988     writeln;
0989     writeln('Please enter the number of hours the search object drifted:');
0990     repeat (until ) 0;
0991         write('HOURS OF DRIFT => ');
0992         readln(MaxHoursDrift);
0993     until MaxHoursDrift > 0;
0994
0995     writeln;
0996     MinLeeDistance := MaxLeeSpeed * MaxHoursDrift;
0997     MaxLeeDistance := MinLeeDistance
0998
0999 end; {Directional Drift Uncertainty}
1000
1001 {*****}
1002 {Calculates the total leeway drift direction and distance.}
1003
1004 procedure LeewayDrift;
1005
1006 begin {LeewayDrift}
1007
1008     CLRSCR;
1009     write('This program will now calculate the LEEWAY vector, ');
1010     writeln('or, that drift caused');
1011     write('by average surface winds pushing on the exposed ');
1012     writeln('area of the search object. ');
1013     writeln('There are three leeway vector calculation options:');
1014     writeln;
1015     write('(1) DRIFT RATE UNCERTAINTY - Used when the search ');
1016     writeln('object is unknown, ');
1017     write('    or, when it is not known whether the object ');
1018     writeln('has deployed an ');
1019     write('    anti-drift device (e.g., sea drogue). All ');
1020     writeln('drift is computed as ');
1021     writeln('    a minimum and maximum distance downwind ');
1022     writeln;
1023     write('(2) DRIFT TIME UNCERTAINTY - Used if the number ');
1024     writeln('of hours the search ');
1025     write('    object has been drifting is not known for ');
1026     writeln('certain. All drift ');
1027     writeln('    is computed as a minimum and maximum distance downwind ');
1028     writeln;
1029     write('(3) DIRECTIONAL DRIFT UNCERTAINTY - Used if above ');
1030     writeln('options do not ');
1031     write('    apply. However, drift is computed for a ');
1032     writeln('divergent bearing left ');
1033     writeln('    and right of the downwind vector. ');
1034     writeln;
1035     writeln('Refer to the National SAR Manual for additional information. ');
1036     writeln;
1037     writeln;
1038     writeln;
1039     writeln;

```

```

1040
1041 repeat {until valid response}
1042     write('PLEASE SELECT OPTION #1, #2, or #3 => ');
1043     readln(LeewayMethod)
1044 until (LeewayMethod = 1) and (LeewayMethod = 3);
1045 CLRSCR;
1046
1047 if LeewayMethod = 1 then
1048 begin {LeewayMethod = Drift Rate Uncertainty}
1049
1050     writeln('Refer to the Leeway Speed Formulae in the National SAR Manual');
1051     write(' (approximately page 8-13). Please enter the minimum ');
1052     writeln('leeway speed');
1053     writeln('at which the search object drifts (e.g., sea drogue deployed):');
1054     repeat {until } 0}
1055         write('MINIMUM LEEWAY SPEED => ');
1056         readln(MinLeeSpeed)
1057     until MinLeeSpeed > 0;
1058
1059     writeln;
1060     writeln('Now, enter the maximum leeway speed at which the search object');
1061     writeln('drifts (e.g. sea drogue NOT deployed):');
1062     repeat {until } 0}
1063         write('MAXIMUM LEEWAY SPEED => ');
1064         readln(MaxLeeSpeed)
1065     until MaxLeeSpeed > 0;
1066
1067     writeln;
1068     writeln('Please enter the number of hours the search object drifted:');
1069     repeat {until } 0}
1070         write('HOURS OF DRIFT => ');
1071         readln(MaxHoursDrift)
1072     until MaxHoursDrift > 0;
1073
1074     writeln;
1075     MaxLeeBrg := LeewayBrg;
1076     MinLeeDistance := MinLeeSpeed * MaxHoursDrift;
1077     MaxLeeDistance := MaxLeeSpeed * MaxHoursDrift
1078
1079 end; {LeewayMethod = Drift Rate Uncertainty}
1080
1081 if LeewayMethod = 2 then
1082 begin {LeewayMethod = Drift Time Uncertainty}
1083
1084     write('Please enter the MINimum number of hours the ');
1085     writeln('search object drifted:');
1086     repeat {until } 0}
1087         write('MINIMUM HOURS OF DRIFT => ');
1088         readln(MinHoursDrift)
1089     until MinHoursDrift > 0;
1090     writeln;
1091

```

```

1092     write('Please enter the MAXimum number of hours the ');
1093     writeln('search object drifted:');
1094     repeat {until } 0}
1095         write('MAXIMUM HOURS OF DRIFT = ');
1096         readln(MaxHoursDrift)
1097     until MaxHoursDrift > 0;
1098     writeln;
1099
1100     writeln('Refer to the Leeway Speed Formulae in the National SAR Manual');
1101     writeln('(approximately page 8-13). Please enter the leeway speed at');
1102     writeln('which the search object drifts:');
1103     repeat {until } 0}
1104         write('MAXIMUM LEEWAY SPEED = ');
1105         readln(MaxLeeSpeed)
1106     until MaxLeeSpeed > 0;
1107
1108     writeln(2);
1109     write('If the number of hours of search object drift ');
1110     writeln('is uncertain, you must');
1111     write('run this program twice (as mentioned ');
1112     writeln('earlier). If this is the');
1113     write('second run, please ENTER the Leeway Speed');
1114     writeln('(read from the graph)');
1115     write('using the previously-recorded Average Surface Wind ');
1116     writeln('vector from the');
1117     writeln('shorter drift period:');
1118     repeat {until valid response}
1119         write('MINIMUM LEEWAY SPEED = ');
1120         readln(MinLeeSpeed)
1121     until MinLeeSpeed >= 0;
1122
1123     MaxLeeBrg := LeewayBrg;
1124     MinLeeDistance := MinLeeSpeed * MinHoursDrift;
1125     MaxLeeDistance := MaxLeeSpeed * MaxHoursDrift
1126
1127     end; {LeewayMethod = Drift Time Uncertainty}
1128     if LeewayMethod = 3 then DriftDirUncertain
1129
1130 end; {LeewayDrift}
1131
1132 {*****}
1133 {Determines the sea (or slope) current affecting search object drift, and, }
1134 {queries user for total observed water current vector, if known. }
1135
1136 procedure SeaCurrent;
1137
1138 begin {SeaCurrent}
1139     CLASCR;
1140     writeln('Please enter the sea current vector as described in the National');
1141     writeln('SAR Manual (approximately pages 8-16i and 8-16j):');
1142
1143     repeat {until valid compass heading}

```

```

1144     write('SEA CURRENT DIRECTION (SET) = ');
1145     readln(DirSeaCurrent)
1146 until (DirSeaCurrent = 0) and (DirSeaCurrent (= 360);
1147 writeln;
1148
1149 repeat {until valid response}
1150     write('SEA CURRENT VELOCITY = ');
1151     readln(SeaCurrSpeed)
1152 until SeaCurrSpeed >= 0;
1153 writeln;
1154
1155 if MinHoursDrift > 0 then
1156     MinSeaCurrDist := MinHoursDrift * SeaCurrSpeed;
1157     MaxSeaCurrDist := MaxHoursDrift * SeaCurrSpeed;
1158
1159 write('Which publication did you use as the source of ');
1160 writeln('your sea current data?:');
1161 writeln;
1162 write('(1) Naval Oceanographic Office Spec. Pub. ');
1163 writeln('Series 4000, Surface Currents');
1164 write('(2) Publication No. 700');
1165 writeln('Oceanographic Atlas');
1166 write('(3) Oceanographic Atlas');
1167 writeln('(4) Atlas of Surface Currents');
1168 write('(5) Pilot Charts');
1169 writeln('(6) Other');
1170 writeln;
1171 repeat {until an above source is selected}
1172     write('SELECT ONLY ONE = ');
1173     readln(SourceSeaCurr)
1174 until (SourceSeaCurr = 1) and (SourceSeaCurr (= 6);
1175
1176 CLASCR;
1177 writeln('Please enter the observed total water current, if known. ');
1178 writeln('This current is determined by observing the drift positions');
1179 writeln('and times of surface debris, oil slicks, or, by inserting');
1180 writeln('an electronic Datum Marker Buoy in the search area. This');
1181 writeln('total water current vector replaces the surface wind and');
1182 writeln('sea current vectors previously calculated by this program. ');
1183 writeln('If unknown, enter a heading of 361:');
1184 repeat {until valid compass heading}
1185     write('TOTAL WATER CURRENT DIRECTION = ');
1186     readln(DirTotalCurrent)
1187 until (DirTotalCurrent = 0) and (DirTotalCurrent (= 361);
1188 writeln;
1189
1190 if DirTotalCurrent < 361 then
1191     repeat {until valid response}
1192         write('TOTAL WATER CURRENT VELOCITY = ');
1193         readln(TotCurrSpeed)
1194     until TotCurrSpeed > 0;
1195
1196 if MinHoursDrift > 0 then MinTotCurrDist := MinHoursDrift * TotCurrSpeed;

```

```

1196     MaxTotCurrDist := MaxHoursDrift * TotCurrSpeed;
1197
1198     CLRSCR;
1199     writeln('If you have calculated a Tidal Current vector, please enter it here');
1200     writeln('If not applicable, enter a heading of 361:');
1201     repeat {until valid compass heading}
1202     write('TIDAL CURRENT DIRECTION = ');
1203     readln(DirTidalCurrent)
1204     until (DirTidalCurrent >= 0) and (DirTidalCurrent <= 361);
1205     writeln;
1206
1207     if DirTidalCurrent < 361 then
1208     begin {Tidal Current calculated/known}
1209     repeat {until valid response}
1210     write('MAXIMUM TIDAL CURRENT DISTANCE = ');
1211     readln(MaxTideDistance)
1212     until MaxTideDistance >= 0;
1213
1214     repeat {until valid response}
1215     write('MINIMUM TIDAL CURRENT DISTANCE = ');
1216     readln(MinTideDistance)
1217     until MinTideDistance >= 0;
1218
1219     end {Tidal Current calculated/known}
1220 end; {SeaCurrent}
1221
1222 {*****}
1223 {Determines the Datum Drift Vector (Min and Max, if applicable) from the }
1224 {vector sum of all previous surface drift vectors. }
1225
1226 procedure Datum;
1227
1228 var BrgRads,           {Individual bearings converted to radians}
1229     TempX,             {Used for min Datum vector calculations}
1230     TempY,             {Used for min Datum vector calculations}
1231     Xcomponent,        {Horizontal component of individual vectors}
1232     Ycomponent : real;  {Vertical component of individual vectors}
1233
1234 begin {Datum}
1235
1236     {Start Maximum Datum Vector Calculations}
1237     Xcomponent := 0.0; Ycomponent := 0.0; {re-initialize}
1238
1239     if DirTotalCurrent = 361 then {Observed Total Water Current unknown}
1240     begin {Find Wind and Sea Current vectors X & Ycomponents}
1241     BrgRads := DirSeaCurrent / radians;
1242     Xcomponent := sin(BrgRads) * MaxSeaCurrDist;
1243     Ycomponent := cos(BrgRads) * MaxSeaCurrDist;
1244     BrgRads := MaxDirWindCurrent / radians;
1245     Xcomponent := Xcomponent + sin(BrgRads) * MaxWindCurrDist;
1246     Ycomponent := Ycomponent + cos(BrgRads) * MaxWindCurrDist
1247     end {Find Wind and Sea Current vectors X & Ycomponents}

```

```

1248
1249     else {Use Observed Total Water Current in lieu of Wind or Sea Currents}
1250     begin {Find Observed Total Water Current vector X & Ycomponents}
1251         BrgRads := DirTotalCurrent / radians;
1252         Xcomponent := sin(BrgRads) * MaxTotCurrDist;
1253         Ycomponent := cos(BrgRads) * MaxTotCurrDist
1254     end; {Find Observed Total Water Current vector X & Ycomponents}
1255
1256     {Use the current X & Ycomponents in the Minimum Datum Vector calculations}
1257     {below if LeewayMethod = 1 or 3.} TempX := Xcomponent; TempY := Ycomponent;
1258
1259     {Find Leeway Drift vector X & Ycomponents}
1260     BrgRads := MaxLeeBrg / radians;
1261     Xcomponent := Xcomponent + sin(BrgRads) * MaxLeeDistance;
1262     Ycomponent := Ycomponent + cos(BrgRads) * MaxLeeDistance;
1263
1264     if MaxTideDistance > 0 then {Tidal Current calculated/known}
1265     begin {Find Tidal Water Current vector X & Ycomponents}
1266         BrgRads := DirTidalCurrent / radians;
1267         Xcomponent := sin(BrgRads) * MaxTideDistance;
1268         Ycomponent := cos(BrgRads) * MaxTideDistance
1269     end; {Find Tidal Water Current vector X & Ycomponents}
1270
1271     CLRSCL;
1272     writeln(' (Max) Datum Vector:');
1273     AddVectors(Xcomponent,Ycomponent);
1274     writeln(2);
1275     DmaxDir := AvgWindFrom;
1276     DmaxDistance := ResultMagnitude;
1277 {End Maximum Datum Vector Calculations}
1278
1279 {Start Minimum Datum Vector Calculations}
1280     Xcomponent := 0.0; Ycomponent := 0.0; {re-initialize}
1281
1282     if DirTotalCurrent = 361 then {Observed Total Water Current unknown}
1283     begin {Find Min Wind and Min Sea Current vectors X & Ycomponents}
1284         if MinSeaCurrDist > 0 then {If Min Sea Current vector exists}
1285         begin {Find Min Sea Current vector X & Y components}
1286             BrgRads := DirSeaCurrent / radians;
1287             Xcomponent := sin(BrgRads) * MinSeaCurrDist;
1288             Ycomponent := cos(BrgRads) * MinSeaCurrDist
1289         end; {Find Min Sea Current vector X & Y components}
1290
1291         if MinDirWindCurrent < 361 then {If Min Wind Current vector exists}
1292         begin {Find Min Wind Current vector X & Y components}
1293             BrgRads := MinDirWindCurrent / radians;
1294             Xcomponent := Xcomponent + sin(BrgRads) * MinWindCurrDist;
1295             Ycomponent := Ycomponent + cos(BrgRads) * MinWindCurrDist
1296         end {Find Min Wind Current vector X & Y components}
1297     end {Find Min Wind and Min Sea Current vectors X & Ycomponents}
1298
1299     else {Use Observed Total Water Current in lieu of Wind or Sea Currents}

```

```

1300 begin {Find Observed Total Water Current vector X & Ycomponents}
1301   BrgRads := DirTotalCurrent / radians;
1302   Xcomponent := sin(BrgRads) * MinTotCurrDist;
1303   Ycomponent := cos(BrgRads) * MinTotCurrDist
1304 end; {Find Observed Total Water Current vector X & Ycomponents}
1305
1306 if LeewayMethod () 2 then
1307 begin {Set Min X & Y component values = Max Sea and WindCurrent values}
1308   Xcomponent := TempX; Ycomponent := TempY
1309 end; {Set Min X & Y component values = Max Sea and WindCurrent values}
1310
1311 {Find Leeway Drift vector X & Ycomponents}
1312 BrgRads := MinLeeBrg / radians;
1313 Xcomponent := Xcomponent + sin(BrgRads) * MinLeeDistance;
1314 Ycomponent := Ycomponent + cos(BrgRads) * MinLeeDistance;
1315
1316 if MinTideDistance > 0 then {Tidal Current calculated/known}
1317 begin {Find Tidal Water Current vector X & Ycomponents}
1318   BrgRads := DirTidalCurrent / radians;
1319   Xcomponent := Xcomponent + sin(BrgRads) * MinTideDistance;
1320   Ycomponent := Ycomponent + cos(BrgRads) * MinTideDistance
1321 end; {Find Tidal Water Current vector X & Ycomponents}
1322
1323 writeln(' (Min) Datum Vector:');
1324 AddVectors(Xcomponent,Ycomponent);
1325 writeln(2);
1326 DminDir := AvgWindFrom;
1327 DminDistance := ResultMagnitude;
1328 {End Minimum Datum Vector Calculations}
1329
1330 write('Total Surface Drift Dir. : ',DminDir:3:0);
1331 writeln('-degrees ',DmaxDir:3:0,'-degrees');
1332 write('Total Surface Drift Dist.: ',DminDistance:4:2);
1333 writeln(' naut. mi. ',DmaxDistance:4:2,' naut. mi. ');
1334 writeln(7);
1335 write('HIT RETURN (Once or twice, as necessary) TO CONTINUE ');
1336 readln(continue);
1337
1338 CLRSCR;
1339 writeln(4);
1340 writeln('Calculating . . . Please stand by')
1341
1342 end; {Datum}
1343
1344 {*****}
1345 {Provides user with program information, limitations on use and license. }
1346
1347 procedure Warranty;
1348
1349 var continue : char; {Bogus read variable provides time to read warranty}
1350
1351 begin {Warranty}

```

```

1352
1353 write('*****');
1354 writeln('*****');
1355 write('(* SEARCH PLANNING SOFTWARE ');
1356 writeln('PROGRAM #2 OF 3 *)');
1357 write('(* TITLE: SURFDRIF.COM (Surface Drift ');
1358 writeln('Algorithm *)');
1359 write('(* VERSION: 1.1 for CP/M ');
1360 writeln('Operating System *)');
1361 write('(* DATE WRITTEN: September 1984');
1362 writeln(' *)');
1363 write('(* LICENSE: COPYRIGHT 1984');
1364 writeln(' D. RICK DOUGLAS *)');
1365 write('*****');
1366 writeln('*****');
1367 write('The author makes no express or implied ');
1368 writeln('warranty of any kind with regard to');
1369 write('this program material, including, but ');
1370 writeln('not limited to, the implied warranty of');
1371 write('fitness for a particular purpose. ');
1372 writeln('The author shall not be liable for');
1373 write('incidental or consequential damages ');
1374 writeln('in connection with or arising out of');
1375 write('furnishing, use, or performance of this ');
1376 writeln('program. The reader MUST HAVE a solid');
1377 write('understanding of search and rescue ');
1378 writeln('methodology before using this software in');
1379 write('making decisions where human life is at ');
1380 writeln('risk. In fact, since no amount of');
1381 write('testing can uncover 100% of program ');
1382 writeln('errors, this program is recommended for');
1383 write('training use only. Prior attendance ');
1384 writeln('at the United States Coast Guard's');
1385 writeln('National SAR School is highly-encouraged. ');
1386 writeln;
1387 write('*****');
1388 writeln(' WARNING! *****');
1389 write('(* THIS SOFTWARE MAY BE FREELY-');
1390 writeln('DISTRIBUTED PROVIDED NO FEE *)');
1391 write('(* IS CHARGED AND THIS');
1392 writeln(' COPYRIGHT NOTICE IS RETAINED *)');
1393 write('*****');
1394 writeln('*****');
1395 writeln;
1396 write('PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE');
1397 readln(continue)
1398
1399 end; {Warranty}
1400
1401 {*****}
1402 {Called by procedure WriteToDisk; Prints out record of Wind and Sea Current }
1403 {vectors or Observed Total Water Current vector, as applicable. }

```

```

1404
1405 procedure RecordCurrents;
1406
1407 var I,J      : integer; {Used as counters}
1408
1409 begin {RecordCurrents}
1410
1411     if DirTotalCurrent = 361 then
1412     begin {Observed Total Water Current unavailable}
1413
1414         {Begin record of Wind Current Vector}
1415         for I := 1 to N do
1416         begin {repeat for each period}
1417             writeln(SeaDat,' ');
1418             writeln(SeaDat,'WIND CURRENT PERIOD #',I:1);
1419             write(SeaDat,'Interval Date-Time-Group Wind ');
1420             writeln(SeaDat,'Coefficients Contributions');
1421
1422             for J := 1 to 8 do
1423             begin {repeat for each interval}
1424                 write(SeaDat,' ',J:1,' ',WindsOverTime[I,J,1]:2:0);
1425                 if WindsOverTime[I,J,2] = 0 then write(SeaDat,'000Z ');
1426                 else if WindsOverTime[I,J,2] = 600 then write(SeaDat,'0600Z ');
1427                 else write(SeaDat,WindsOverTime[I,J,2]:4:0,'Z ');
1428                 write(SeaDat,WindsOverTime[I,J,3]:3:0,' / ');
1429                 write(SeaDat,WindsOverTime[I,J,4]:2:0,' ');
1430                 write(SeaDat,WindCoeffs[J,1]:3:0,' / ');
1431                 write(SeaDat,WindCoeffs[J,2]:4:3,' ');
1432                 write(SeaDat,WindsOverTime[I,J,5]:3:0,' / ');
1433                 writeln(SeaDat,WindsOverTime[I,J,6]:4:3);
1434             end; {repeat for each interval}
1435             writeln(SeaDat,' ');
1436             write(SeaDat,'Local Wind Current This Period ');
1437             writeln(SeaDat,PeriodVectors[I,1]:3:0,' / ',PeriodVectors[I,2]:4:3);
1438             write(SeaDat,'Number Of Hours In This Period ');
1439             writeln(SeaDat,HoursWindEffect[I]:1);
1440             write(SeaDat,'Wind Current Vector This Period ');
1441             write(SeaDat,PeriodVectors[I,1]:3:0,' / ',PeriodVectors[I,3]:4:3);
1442             writeln(SeaDat,' nautical miles');
1443             write(SeaDat,'-----');
1444             writeln(SeaDat,'-----');
1445         end; {repeat for each period}
1446
1447         write(SeaDat,'Total Wind Current Direction ');
1448         if MaxDirWindCurrent > 100 then
1449             writeln(SeaDat,MaxDirWindCurrent:3:0,'-degrees True')
1450         else if MaxDirWindCurrent > 10 then
1451             writeln(SeaDat,'0',MaxDirWindCurrent:2:0,'-degrees True')
1452         else writeln(SeaDat,'00',MaxDirWindCurrent:1:0,'-degrees True');
1453
1454         if MinHoursDrift = 0 then
1455         begin {Minimum wind current vector not computed}

```

```

1456     write(SeaDat,'Total Wind Current Distance      ');
1457     writeln(SeaDat,MaxWindCurrDist:4:2,' nautical miles')
1458 end   {Minimum wind current vector not computed}
1459 else
1460 begin {max and min wind current drift distances exist}
1461     write(SeaDat,'Total Wind Curr. Distance: Min = ',MinWindCurrDist:4:2);
1462     writeln(SeaDat,' naut.mi.   Max = ',MaxWindCurrDist:4:2,' naut. mi.')
1463 end;   {max and min seacurrent drift distances exist}
1464
1465     write(SeaDat,'=====');
1466     writeln(SeaDat,'=====');
1467     writeln(SeaDat,' ');
1468     writeln(SeaDat,'SEA CURRENT VECTOR:      ');
1469
1470     if SourceSeaCurr = 1 then
1471     begin {Source #1 used}
1472         write(SeaDat,'Used Naval Oceanographic Office Spec. Pub. ');
1473         writeln(SeaDat,'Series 4000, Surface Currents')
1474     end;   {Source #1 used}
1475     if SourceSeaCurr = 2 then writeln(SeaDat,'Used Publication No. 700');
1476     if SourceSeaCurr = 3 then writeln(SeaDat,'Used Oceanographic Atlas');
1477     if SourceSeaCurr = 4 then
1478         writeln(SeaDat,'Used Atlas of Surface Currents');
1479     if SourceSeaCurr = 5 then writeln(SeaDat,'Used Pilot Charts');
1480     writeln;
1481     if SourceSeaCurr = 6 then
1482         writeln(SeaDat,'Source: _____');
1483
1484     write(SeaDat,'Sea Current Direction (Set)      ');
1485     if DirSeaCurrent > 100 then
1486         writeln(SeaDat,DirSeaCurrent:3:0,'-degrees True')
1487     else if DirSeaCurrent > 10 then
1488         writeln(SeaDat,'0',DirSeaCurrent:2:0,'-degrees True')
1489     else writeln(SeaDat,'00',DirSeaCurrent:1:0,'-degrees True');
1490     write(SeaDat,'Sea Current Drift Rate          ');
1491     writeln(SeaDat,SeaCurrSpeed:4:2,' knots');
1492     if MinHoursDrift = 0 then
1493     begin {No minimum vector calculations}
1494         write(SeaDat,'Sea Current Drift Distance:      ');
1495         writeln(SeaDat,MaxSeaCurrDist:4:2,' knots')
1496     end   {No minimum vector calculations}
1497     else
1498     begin {max and min seacurrent drift distances exist}
1499         write(SeaDat,'Sea Current Drift Distance: Min = ',MinSeaCurrDist:4:2);
1500         writeln(SeaDat,' knots   Max = ',MaxSeaCurrDist:4:2,' knots')
1501     end   {max and min seacurrent drift distances exist}
1502
1503 end;   {Observed Total Water Current unavailable}
1504
1505 if (DirTotalCurrent < 361) and (TotCurrSpeed < 0) then
1506 begin {Total water current vector is known}
1507     write(SeaDat,'=====');

```

```

1508      writeln(SeaDat,'=====');
1509      writeln(SeaDat,' ');
1510      writeln(SeaDat,'OBSERVED TOTAL WATER CURRENT VECTOR:      ');
1511      write(SeaDat,'Tot Current Direction:      ');
1512      if DirTotalCurrent > 100 then
1513          writeln(SeaDat,DirTotalCurrent:3:0,'-degrees True')
1514      else if DirTotalCurrent > 10 then
1515          writeln(SeaDat,'0',DirTotalCurrent:2:0,'-degrees True')
1516      else writeln(SeaDat,'00',DirTotalCurrent:1:0,'-degrees True');
1517      write(SeaDat,'Tot Current Drift Speed:      ',TotCurrSpeed:4:2);
1518      writeln(SeaDat,' knots');
1519      if MinHoursDrift = 0 then
1520          begin
1521              write(SeaDat,'Tot Current Drift Distance:      ');
1522              writeln(SeaDat,MaxTotCurrDist:4:2,' knots')
1523          end
1524      else
1525          begin {max and min total current drift distances exist}
1526              write(SeaDat,'Tot Current Drift Distance: Min = ',MinTotCurrDist:4:2);
1527              writeln(SeaDat,' knots      Max = ',MaxTotCurrDist:4:2,' knots')
1528          end {max and min total current drift distances exist}
1529      end
1530      end {Total water current vector is known}
1531  end; {RecordCurrents}
1532
1533  {*****}
1534  {Prints out record of search planning inputs and calculations from}
1535  {this program.      }
1536
1537  procedure WriteToDisk;
1538
1539      var I      : integer; {Used as counter}
1540          LastDegree,      {Last known position degrees}
1541          LastMinute : real; {Last known position minutes}
1542
1543  begin {WriteToDisk}
1544      assign(SeaDat,'SeaData');
1545      rewrite(SeaDat);
1546
1547      {Begin record of last known position and time}
1548      LastDegree := trunc(LastLatitudeKnown);
1549      LastMinute := round( (LastLatitudeKnown-LastDegree) * 100);
1550      write(SeaDat,'Missing aerospace object/pilot(s) last known position:');
1551      if LastDegree < 10 then write(SeaDat,' 0',LastDegree:1:0,'-')
1552      else write(SeaDat,' ',LastDegree:2:0,'-');
1553      if LastMinute < 10 then write(SeaDat,'0',LastMinute:1:0,' ')
1554      else write(SeaDat,LastMinute:2:0,' ');
1555      if LatNS = 'N' then write(SeaDat,'North') else write(SeaDat,'South');
1556
1557      LastDegree := trunc(LastLongitudeKnown);
1558      LastMinute := round( (LastLongitudeKnown-LastDegree) * 100);
1559      if LastDegree < 10 then write(SeaDat,' 0',LastDegree:1:0,'-')

```

```

1560     else write(SeaDat, ' ', LastDegree:2:0, '-');
1561     if LastMinute < 10 then write(SeaDat, ' 0', LastMinute:1:0, ' ');
1562     else write(SeaDat, ' ', LastMinute:2:0, ' ');
1563     if LongEW = 'W' then writeln(SeaDat, 'West') else writeln(SeaDat, 'East');
1564
1565     {Prints a zero in front of one-digit dates in Z DTG format}
1566     if trunc(LastDateTime / 10000) < 10 then
1567       writeln(SeaDat, 'Time: 0', LastDateTime:5:0, 'Z ', LastMonth:2, '/', LastYear:4)
1568     else
1569       writeln(SeaDat, 'Time: ', LastDateTime:6:0, 'Z ', LastMonth:2, '/', LastYear:4);
1570     write(SeaDat, '=====');
1571     writeln(SeaDat, '=====');
1572
1573     {Begin record of Average Surface (Leeway) Winds}
1574     writeln(SeaDat, 'AVERAGE SURFACE WINDS:');
1575     writeln(SeaDat, 'Date-Time-Group    Hours    Wind    Contributions');
1576     for I := 1 to N do
1577       begin {repeat for each wind block}
1578         write(SeaDat, ' ', WindsOverTime[I,1,1]:2:0);
1579         if WindsOverTime[I,1,2] = 0 then write(SeaDat, '0000Z ');
1580         else if WindsOverTime[I,1,2] = 600 then write(SeaDat, '0600Z ');
1581         else write(SeaDat, WindsOverTime[I,1,2]:4:0, 'Z ');
1582         write(SeaDat, HoursWindEffect[I,1,1], ' ');
1583         write(SeaDat, WindsOverTime[I,1,3]:3:0, ' / ');
1584         write(SeaDat, WindsOverTime[I,1,4]:2:0, ' ');
1585         write(SeaDat, WindsOverTime[I,1,3]:3:0, ' / ');
1586         writeln(SeaDat, VelocityComponent[I]:3:0)
1587       end; {repeat for each block}
1588
1589     writeln(SeaDat, ' ');
1590     write(SeaDat, 'Total Average Surface Wind Direction ');
1591     if DirSurfWind > 100 then
1592       write(SeaDat, DirSurfWind:3:0, '-degrees True')
1593     else if DirSurfWind > 10 then
1594       write(SeaDat, '0', DirSurfWind:2:0, '-degrees True')
1595     else write(SeaDat, '00', DirSurfWind:1:0, '-degrees True');
1596     writeln(SeaDat, ' at ', SurfWndSpeed:4:2, ' knots');
1597     write(SeaDat, '=====');
1598     writeln(SeaDat, '=====');
1599
1600     writeln(SeaDat, ' ');
1601     write(SeaDat, 'LEEWAY DRIFT VECTOR CALCULATIONS: ');
1602     if LeewayMethod = 1 then
1603       begin {Drift rate uncertainty}
1604         writeln(SeaDat, 'Drift Rate Uncertainty');
1605         writeln(SeaDat, 'Hours Of Drift ', MaxHoursDrift:2);
1606         write(SeaDat, 'Drift Rate ', Min = ', MinLeeSpeed:4:2);
1607         writeln(SeaDat, ' knots Max = ', MaxLeeSpeed:4:2, ' knots');
1608         write(SeaDat, 'Leeway Direction ');
1609         writeln(SeaDat, MaxLeeDir:3:0, '-degrees True');
1610         if MaxLeeDir > 100 then
1611           writeln(SeaDat, MaxLeeDir:3:0, '-degrees True')

```

```

1612     else if MaxLeeBrg > 10 then
1613         writeln(SeaDat,'0',MaxLeeBrg:2:0,'-degrees True')
1614     else writeln(SeaDat,'00',MaxLeeBrg:1:0,'-degrees True');
1615     write(SeaDat,'Leeway Distance           Min = ',MinLeeDistance:4:2);
1616     writeln(SeaDat,' naut.mi. Max = ',MaxLeeDistance:4:2,' naut.mi.')
1617 end; {Drift rate uncertainty}
1618
1619 if LeewayMethod = 2 then
1620 begin {Drift time uncertainty}
1621     writeln(SeaDat,'Drift Time Uncertainty');
1622     write(SeaDat,'Hours Of Drift           Min = ',MinHoursDrift:2);
1623     writeln(SeaDat,'                        Max = ',MaxHoursDrift:2);
1624     write(SeaDat,'Drift Rate               ',MaxLeeSpeed:4:2);
1625     writeln(SeaDat,' knots');
1626     write(SeaDat,'Leeway Direction         ');
1627     if MaxLeeBrg > 100 then
1628         writeln(SeaDat,MaxLeeBrg:3:0,'-degrees True')
1629     else if MaxLeeBrg > 10 then
1630         writeln(SeaDat,'0',MaxLeeBrg:2:0,'-degrees True')
1631     else writeln(SeaDat,'00',MaxLeeBrg:1:0,'-degrees True');
1632     write(SeaDat,'Leeway Distance           Min = ',MinLeeDistance:4:2);
1633     writeln(SeaDat,' naut.mi. Max = ',MaxLeeDistance:4:2,' naut.mi.')
1634 end; {Drift time uncertainty}
1635
1636 if LeewayMethod = 3 then
1637 begin {Directional drift uncertainty}
1638     writeln(SeaDat,'Directional Drift Uncertainty');
1639     write(SeaDat,'Maximum Expected Divergence ');
1640     writeln(SeaDat,MaxDivergence:2,'-degrees');
1641     write(SeaDat,'Drift Rate               ');
1642     writeln(SeaDat,MaxLeeSpeed:5:3,' knots');
1643     writeln(SeaDat,'Hours Of Drift           ',MaxHoursDrift:2);
1644     write(SeaDat,'Leeway Direction         Min = ');
1645     if MinLeeBrg > 100 then
1646         write(SeaDat,MinLeeBrg:3:0,'-degrees Max = ')
1647     else if MinLeeBrg > 10 then
1648         write(SeaDat,'0',MinLeeBrg:2:0,'-degrees Max = ')
1649     else write(SeaDat,'00',MinLeeBrg:1:0,'-degrees Max = ');
1650     if MaxLeeBrg > 100 then
1651         writeln(SeaDat,MaxLeeBrg:3:0,'-degrees True')
1652     else if MaxLeeBrg > 10 then
1653         writeln(SeaDat,'0',MaxLeeBrg:2:0,'-degrees True')
1654     else writeln(SeaDat,'00',MaxLeeBrg:1:0,'-degrees True');
1655     write(SeaDat,'Leeway Distance           Min = ',MinLeeDistance:4:2);
1656     writeln(SeaDat,' naut.mi. Max = ',MaxLeeDistance:4:2,' naut.mi.')
1657 end; {Directional drift uncertainty}
1658
1659 write(SeaDat,'=====');
1660 writeln(SeaDat,'=====');
1661 RecordCurrents;
1662
1663 if MaxTideDistance > 0 then

```

```

1664 begin {Tidal Current calculated/known}
1665   write(SeaDat,'Tidal Current Direction:      ');
1666   if DirTidalCurrent > 100 then
1667     writeln(SeaDat,DirTidalCurrent:3:0,'-degrees True')
1668   else if DirTidalCurrent > 10 then
1669     writeln(SeaDat,'0',DirTidalCurrent:2:0,'-degrees True')
1670   else writeln(SeaDat,'00',DirTidalCurrent:1:0,'-degrees True');
1671   if MinTideDistance > 0 then
1672     begin {max and min tidal current drift distances exist}
1673       write(SeaDat,'Tidal Curr. Drift Distance: Min = ');
1674       write(SeaDat,MinTideDistance:4:2,' naut.mi.  Max = ');
1675       writeln(SeaDat,MaxTideDistance:4:2,' naut.mi.')
1676     end {max and min tidal current drift distances exist}
1677   else
1678     begin
1679       write(SeaDat,'Tidal Curr. Drift Distance:      ');
1680       writeln(SeaDat,MaxTotCurrDist:4:2,' naut.mi.')
1681     end
1682   end; {Tidal Current calculated/known}
1683
1684   write(SeaDat,'=====');
1685   writeln(SeaDat,'=====');
1686   writeln(SeaDat,' ');
1687   writeln(SeaDat,'TOTAL SURFACE DRIFT (Dmin & Dmax) VECTORS:');
1688   write(SeaDat,'Tot.Surf.Drift Direction :  ');
1689   if DminDir > 100 then
1690     write(SeaDat,DminDir:3:0,'-degrees      ')
1691   else if DminDir > 10 then
1692     write(SeaDat,'0',DminDir:2:0,'-degrees      ')
1693   else write(SeaDat,'00',DminDir:1:0,'-degrees      ');
1694   if DmaxDir > 100 then
1695     writeln(SeaDat,DmaxDir:3:0,'-degrees True')
1696   else if DmaxDir > 10 then
1697     writeln(SeaDat,'0',DmaxDir:2:0,'-degrees True')
1698   else writeln(SeaDat,'00',DmaxDir:1:0,'-degrees True');
1699   write(SeaDat,'Tot.Surf.Drift Distance :  ',DminDistance:4:2);
1700   writeln(SeaDat,' naut. mi.  ',DmaxDistance:4:2,' naut. mi.');
```

```

1701
1702   write(SeaDat,'=====');
1703   writeln(SeaDat,'=====');
1704   writeln(SeaDat,' ');
1705   close(SeaDat)
1706 end; {WriteToDisk}
1707
1708
1709 {*****}
1710 begin {main program}
1711
1712   {initialize program variables}
1713   continue := 'Q'; DatumDateTime := 0.0; DatumMonth := 0; DatumYear := 0;
1714   Days := 0; DegreesLat := 180; DirSeaCurrent := 361.0;
1715   DirSurfWind := 0.0; DirTidalCurrent := -1.0; DirTotalCurrent := -1.0;

```

```

1716 DmaxDir := -1.0; DminDir := -1.0; DmaxDistance := 0.0;
1717 DminDistance := 0.0; HoursElapsed := 0; LastDateTime := 0.0;
1718 LastDay := 0; LastHour := 0; LastLatitudeKnown := 0.0;
1719 LastLongitudeKnown := 0.0; LastMonth := 0; LastYear := 0;
1720 LatNS := 'Q'; LeewayBrig := 0.0; LeewayMethod := 4; LongEW := 'Q';
1721 MaxDirWindCurrent := -1.0; MinDirWindCurrent := -1.0;
1722 MaxDivergence := 0; MaxHoursDrift := 0; MinHoursDrift := 0;
1723 MaxLeeBrig := 0.0; MinLeeBrig := 0.0; MaxLeeDistance := 0.0;
1724 MinLeeDistance := 0.0; MaxLeeSpeed := 0.0; MinLeeSpeed := 0.0;
1725 MaxSeaCurrDist := 0.0; MinSeaCurrDist := 0.0; MaxTideDistance := -1.0;
1726 MinTideDistance := -1.0; MaxTotCurrDist := 0.0; MinTotCurrDist := 0.0;
1727 MaxWindCurrDist := -1.0; MinWindCurrDist := -1.0; N := 0;
1728 ResultMagnitude := 0.0; SeaCurrSpeed := -1.0; SourceSeaCurr := 0;
1729 SurfWndMagnitude := 0.0; SurfWndSpeed := 0.0; TotCurrSpeed := 0.0;
1730
1731 {initialize character sets}
1732 ValidAnswers := ['Y','y','N','n']; No := ['N','n']; Yes := ['Y','y'];
1733
1734 {initialize program arrays}
1735 for I := 1 to 8 do for J := 1 to 2 do WindCoeffs[I,J] := 0.0;
1736 for I := 1 to max do
1737   begin
1738     HoursWindEffect[I] := 0;
1739     VelocityComponent[I] := 0.0;
1740     for J := 1 to 8 do
1741       for K := 1 to 6 do WindsOverTime[I,J,K] := 0.0;
1742       for J := 1 to 2 do PeriodVectors[I,J] := 0.0
1743   end;
1744
1745 Warranty;
1746 CLRSCR;
1747 writeln('Given the initial surface position coordinates (latitude and');
1748 writeln('longitude) of an object on the ocean's surface and starting');
1749 writeln('date/time, this program calculates an updated surface position');
1750 writeln('(Datum point) or search area for a specified (Datum) time. If');
1751 writeln('the updated surface position coordinates are all ready known,');
1752 writeln('then enter N to the next question and proceed with other recovery');
1753 writeln('planning as outlined in the National Search and Rescue Manual. ');
1754 writelns(2);
1755 writeln('P.S. Run this program TWICE if the number of hours of search');
1756 writeln('object drift is uncertain. On the first run, enter data only');
1757 writeln('for the SHORTER drift period (later drift start time). After');
1758 writeln('the program calculates the Average Surface Wind and Wind Current');
1759 writeln('vectors, record them, ABORT the program run, and CLEAR your');
1760 writeln('microcomputer's memory. Then rerun the entire program, making');
1761 writeln('sure to enter data for the LONGER drift period (earlier drift');
1762 writeln('start time). Enter the vectors you recorded above when asked');
1763 writeln('by the program. ');
1764 writelns(4);
1765 repeat {until valid response}
1766   write('Do you wish to continue with this program? (y/n) = ');
1767   readln(continue);

```

```

1768      writeln
1769      until (continue in ValidAnswers);
1770
1771      if (continue in Yes) then
1772      begin {program run}
1773          continue := 'Q'; {re-initialize}
1774          CLRSCR;
1775          SurfacePosition;
1776          CLRSCR;
1777
1778          writeln('This program includes wind current calculations to determine');
1779          writeln('Datum. However, according to the National SAR Manual, wind');
1780          writeln('currents are usually ignored in coastal, lake, river, and');
1781          writeln('harbor areas due to the many variable effects from the water-');
1782          writeln('land interface. This program is based on the assumption of');
1783          writeln('open-sea search where land masses do not interfere with the');
1784          writeln('action of the wind on the water or on the currents generated');
1785          writeln('by them. A rule of thumb is to calculate wind currents when');
1786          writeln('water depths are greater than 100 feet (32 meters) and at');
1787          writeln('distances of 20 miles (32 kilometers) or greater from shore. ');
1788          writeln('Wind currents are not usually used inside these limits');
1789          writeln('except where local knowledge makes it possible to estimate');
1790          writeln('one. This is especially true where, close to shore, the ');
1791          writeln('water''s depth increases rapidly. ');
1792          writeln(8);
1793          repeat {until valid response}
1794              write('Do you wish to continue with this program? (y/n) = ');
1795              readln(continue);
1796              writeln
1797          until (continue in ValidAnswers);
1798
1799          if (continue in Yes) then
1800          begin {Ocean Current calculations}
1801              CLRSCR;
1802
1803              writeln('Please enter the number of hours elapsed (to the NEAREST');
1804              writeln('HOUR) from the last known surface position (LKP) time to');
1805              writeln('the desired Datum time. ');
1806              writeln(2);
1807              writeln('If the number of hours of search object drift is uncertain, ');
1808              writeln('you must run this program twice (as mentioned earlier). On ');
1809              writeln('the first run enter here the SHORTER number of hours of ');
1810              writeln('search object drift. On the second run, or if two runs are ');
1811              writeln('not applicable, enter here the difference between the Datum ');
1812              writeln('and LKP times as requested above. ');
1813              repeat {until valid HoursElapsed response}
1814                  write('ELAPSED HOURS = ');
1815                  readln(HoursElapsed);
1816                  writeln
1817              until (HoursElapsed > 0);
1818
1819              WindPeriods;

```

```

1820      PeriodTimes;
1821      InputSeaWinds;
1822      WindLatCoeffs;
1823      WindCurrent;
1824      AvgSurfaceWind;
1825      LeewayDrift;
1826      SeaCurrent;
1827      Datum;
1828      WriteToDisk;
1829
1830      CLASCR;
1831      writeln;
1832      writeln('A record of significant input and output data used during this');
1833      writeln('program run is stored in an external file named "SEADATA".');
1834      writeln('If you desire to keep this record permanently, please rename');
1835      writeln('file SEADATA before running this program again!')
1836
1837      end      {Ocean Current calculations}
1838      end      {program run}
1839      end.    {main program}

```

Surface Drift Determination Program (82 of 3)
Variable & Operator Cross-Reference Listing

<u>Variable</u>	<u>Program Line Number</u>									
AddVectors	729	849	872	934	1273	1324				
AvgSurfaceWind	914	1824								
AvgWindFrom	118	739	743	744	747	748	751	752	753	850
	873	936	1275	1326						
AvgWindTo	119	752	753	935						
BrgRads	830	843	844	845	866	867	868	917	927	928
	929	1228	1241	1242	1243	1244	1245	1246	1251	1252
	1253	1260	1261	1262	1266	1267	1268	1286	1287	1288
	1293	1294	1295	1301	1302	1303	1312	1313	1314	1318
	1319	1320								
continue	97	883	951	1336	1349	1397	1713	1767	1769	1771
	1773	1795	1797	1799						
count	200	203	204	207	207					
DateTime	214	225	231	235	235	242	246	246	253	
Datum	1226	1827								
DatumDateTime	120	383	384	387	1713					
DatumMonth	101	365	366	366	369	369	1713			
DatumYear	102	373	1713							
Days	103	455	456	457	458	459	460	461	462	463
	464	465	466	467	498	535	558	1714		
DaysInMonth	441	497	534	555						
DegreesLat	104	331	655	773	781	880	1714			
DirSeaCurrent	121	1145	1146	1146	1241	1286	1485	1486	1487	1488
	1489	1714								
DirSurfWind	122	936	940	1591	1592	1593	1594	1595	1715	
DirTidalCurrent	123	1203	1204	1204	1207	1266	1318	1666	1667	1668
	1669	1670	1715							

DirTotalCurrent	124	1185	1186	1186	1189	1239	1251	1282	1301	1411
	1505	1512	1513	1514	1515	1516	1715			
DmaxDir	125	1275	1331	1695	1696	1697	1698	1699	1716	
DmaxDistance	127	1276	1333	1701	1716					
DminDir	126	1326	1330	1690	1691	1692	1693	1694	1716	
DminDistance	128	1327	1332	1700	1717					
DriftDirUncertain	959	1128								
HoursElapsed	107	417	938	1717	1815	1817				
HoursRemaining	400	417	418	423	426	426	430	431	433	
HoursWindEffect	164	406	408	410	412	414	417	425	430	852
	926	1439	1582	1738						
I	105	479	543	546	546	549	549	550	550	551
	551	552	552	553	558	559	559	562	564	569
	569	570	570	572	612	615	615	617	618	619
	620	621	622	626	627	628	632	634	635	639
	639	640	640	642	652	659	661	661	661	662
	770	789	793	794	795	795	798	799	800	801
	818	818	829	835	841	841	842	842	843	844
	845	848	850	851	852	852	852	864	866	867
	868	916	924	926	926	926	927	928	929	1407
	1415	1418	1424	1425	1426	1427	1428	1429	1432	1433
	1437	1437	1439	1441	1441	1539	1576	1578	1579	1580
	1581	1582	1583	1584	1585	1586	1735	1735	1736	1738
	1739	1741	1742							
InputSeaWinds	579	1821								
J	105	479	510	513	517	517	520	520	525	527
	527	528	535	536	567	569	569	570	570	581
	586	589	590	591	592	593	594	595	596	600
	601	601	605	607	607	637	639	639	640	640
	675	708	708	829	839	841	841	841	842	842
	842	843	844	845	1407	1422	1424	1424	1425	1426
	1427	1428	1429	1430	1431	1432	1433	1735	1735	1740
	1741	1742	1742							
K	105	1741	1741							
LastDateTime	129	306	307	309	1566	1567	1569	1717		
LastDay	100	310	311	485	498	500	1718			

LastDegree	1540 1560	1548	1549	1551	1551	1552	1557	1558	1559	1559
LastHour	109 410 489	311 411 491	405 411 491	405 412 494	406 413 494	407 414 1718	407 486	408 487	409 487	409 489
LastLatitudeKnown	130	326	327	327	330	330	331	1548	1549	1718
LastLongitudeKnown	131	345	346	346	349	349	1557	1558	1719	
LastMinute	1541	1549	1553	1553	1554	1558	1561	1561	1562	
LastMonth	110 1567	281 1569	282 1719	282	285	285	497	532	532	555
LastYear	111	290	449	1567	1569	1719				
LatNS	98 656	317 777	319 779	319 783	319 1555	319 1720	320	320	320	320
LeapYearCheck	443	450	451							
LeewayBrq	132 1720	935	971	972	973	975	976	977	1075	1123
LeewayDrift	1004	1025								
LeewayMethod	112 1636	1043 1720	1044	1044	1047	1081	1128	1306	1602	1519
lines	198	204								
LongEW	99 1563	336 1720	338	338	338	338	339	339	339	339
max	94	164	171	188	193	1736				
MaxDirWindCurrent	133 1721	857	873	886	1244	1448	1449	1450	1451	1452
MaxDivergence	113 1640	968 1722	969	969	971	972	973	975	976	977
MaxHoursDrift	114 1125	992 1157	993 1196	996 1605	1071 1623	1072 1643	1076 1722	1077	1096	1097
MaxLeeBrq	135 1613 1653	976 1614 1654	977 1627 1723	1075 1628	1123 1629	1260 1630	1609 1631	1610 1650	1611 1651	1612 1652
MaxLeeDistance	137	997	1077	1125	1261	1262	1616	1633	1656	1723

MaxLeeSpeed	139 1607	985 1624	986 1642	996 1724	1064	1065	1077	1105	1106	1125
MaxSeaCurrDist	141	1157	1242	1243	1495	1500	1725			
MaxTideDistance	143	1211	1212	1264	1267	1268	1663	1675	1725	
MaxTotCurrDist	145	1196	1252	1253	1522	1527	1680	1726		
MaxWindCurrDist	147 1727	858	874	880	880	887	1245	1246	1457	1462
MinDirWindCurrent	134	899	900	900	902	1291	1293	1721		
MinHoursDrift	115 1519	1088 1622	1089 1722	1124	1155	1156	1195	1195	1454	1492
MinLeeBrq	136	972	973	1312	1645	1646	1647	1648	1649	1723
MinLeeDistance	138 1724	996	997	1076	1124	1313	1314	1615	1632	1655
MinLeeSpeed	140	1056	1057	1076	1120	1121	1124	1606	1724	
MinSeaCurrDist	142	1156	1284	1287	1288	1499	1725			
MinTideDistance	144	1216	1217	1316	1319	1320	1671	1674	1726	
MinTotCurrDist	146	1195	1302	1303	1526	1726				
MinWindCurrDist	148	985	986	1294	1295	1461	1727			
MonthBefore	480	532	533	534						
N	106 835	416 855	422 864	422 924	425 1415	430 1576	541 1727	572	610	642
No	155	692	1732							
PeriodTimes	477	1820								
PeriodVectors	193 1437	850 1437	851 1441	852 1441	852 1742	857	858	866	867	868
radians	95 1286	736 1293	843 1301	866 1312	927 1318	1241	1244	1251	1260	1266
RecordCurrents	1485	1661								
ResultMagnitude	149	757	851	874	937	1276	1327	1728		
SeaCurrSpeed	150	1151	1152	1156	1157	1491	1728			

SeaCurrent	1136	1826								
SeaDat	159	1417	1418	1419	1420	1424	1425	1426	1427	1428
	1429	1430	1431	1432	1433	1435	1436	1437	1438	1439
	1440	1441	1442	1443	1444	1447	1449	1451	1452	1456
	1457	1461	1462	1465	1466	1467	1468	1472	1473	1475
	1476	1478	1479	1482	1484	1486	1488	1489	1490	1491
	1494	1495	1499	1500	1507	1508	1509	1510	1511	1513
	1515	1516	1517	1518	1521	1522	1526	1527	1544	1545
	1550	1551	1552	1553	1554	1555	1555	1559	1560	1561
	1562	1563	1563	1567	1569	1570	1571	1574	1575	1578
	1579	1580	1581	1582	1583	1584	1585	1586	1589	1590
	1592	1594	1595	1596	1597	1598	1600	1601	1604	1605
	1606	1607	1608	1609	1611	1613	1614	1615	1616	1621
	1622	1623	1624	1625	1626	1628	1630	1631	1632	1633
	1638	1639	1640	1641	1642	1643	1644	1646	1648	1649
	1651	1653	1654	1655	1656	1659	1660	1665	1667	1669
	1670	1673	1674	1675	1679	1680	1685	1686	1687	1688
	1689	1691	1693	1694	1696	1698	1699	1700	1701	1703
	1704	1705	1706							
SourceSeaCurr	116	1172	1173	1173	1470	1475	1476	1477	1479	1481
	1728									
SurfWndMagnitude	151	937	938	1729						
SurfWndSpeed	152	938	941	1596	1729					
SurfaceDrift	92									
SurfacePosition	265	1775								
Temp	216	235	236	246	247	267	309	310	311	387
TempAngle	731	736	739	742	743	744	746	747	748	
TempDate	217	225	226	226	256	256				
TempHour	218	236	237	237	256	257				
TempMinutes	219	247	248	257						
TempMonth	441	447	451	454						
TempX	1229	1257	1308							
TempY	1230	1257	1308							
TempYearDiv	444	449	450	450						
TotCurrSpeed	153	1192	1193	1195	1196	1505	1517	1729		

ValidAnswers	157	690	719	1732	1769	1797				
VelocityComponent	171	926	928	929	1586	1739				
VerifyDTG	214	307	384							
VerifyWinds	672	806								
Warranty	1347	1745								
WindChart	650	682	712							
WindCheck	674	679	688	690	692	717	719			
WindCoeffs	177	661	662	705	706	707	708	794	795	795
	799	800	801	818	841	842	1430	1431	1735	
WindCurrent	827	1823								
WindError	676	679	697	699	699	701	705	706	707	708
WindLatCoeffs	768	1822								
WindPeriods	398	1819								
WindsOverTime	188	485	486	488	490	492	500	501	505	506
	513	517	517	520	520	525	527	527	528	535
	536	549	549	550	550	551	551	552	552	553
	558	559	559	562	564	569	569	570	570	589
	590	591	592	593	594	595	596	600	601	601
	605	607	607	617	618	619	620	621	622	626
	627	628	632	634	635	639	639	640	640	841
	841	842	842	843	844	845	926	927	1424	1425
	1426	1427	1428	1429	1432	1433	1578	1579	1580	1581
	1583	1584	1585	1741						
writeln	198	293	298	313	351	584	709	722	775	805
	884	888	932	942	949	1008	1045	1108	1139	1175
	1198	1271	1274	1325	1334	1338	1339	1746	1754	1764
	1774	1776	1792	1801	1806	1830				
WriteToDisk	1537	1828								
xcomponent	729	736	739	740	741	745	757	757	759	831
	837	844	844	849	863	867	867	872	918	922
	928	928	934	1231	1237	1242	1245	1245	1252	1257
	1261	1261	1267	1273	1280	1287	1294	1294	1302	1308
	1313	1313	1319	1319	1324					

Ycomponent	729	736	739	740	741	745	757	757	760	832
	837	845	845	849	863	868	868	872	919	922
	929	929	934	1232	1237	1243	1246	1246	1253	1257
	1262	1262	1268	1273	1280	1288	1295	1295	1303	1308
	1314	1314	1320	1320	1324					

Yes	156	1732	1771	1799
-----	-----	------	------	------

<u>Operator</u>	<u>Program Line Number</u>									
arctan	736									
assign	1544									
char	99	157	674	1349						
close	1706									
cos	845	868	929	1243	1246	1253	1262	1268	1288	1295
	1303	1314	1320							
input	92									
integer	116	164	198	200	400	441	480	581	652	675
	676	770	829	916	1407	1539				
output	92									
read	708									
readln	231	242	253	281	290	306	317	326	336	345
	365	373	383	600	605	626	632	688	697	717
	794	799	883	899	905	951	968	985	992	1043
	1056	1064	1071	1088	1096	1105	1120	1145	1151	1172
	1185	1192	1203	1211	1216	1336	1397	1767	1795	1815
real	153	171	177	188	193	214	219	267	444	729
	731	832	919	1232	1541					
rewrite	1545									
round	311	406	408	410	412	414	1549	1558		
sin	844	867	928	1242	1245	1252	1261	1267	1287	1294
	1302	1313	1319							
sqrt	757									
text	159									

1625	1628	1630	1631	1633	1638	1640	1642	1643	1651
1653	1654	1656	1660	1667	1669	1670	1675	1680	1686
1687	1688	1696	1698	1699	1701	1704	1705	1747	1748
1749	1750	1751	1752	1753	1755	1756	1757	1758	1759
1760	1761	1762	1763	1768	1778	1779	1780	1781	1782
1783	1784	1785	1786	1787	1788	1789	1790	1791	1796
1803	1804	1805	1807	1808	1809	1810	1811	1812	1816
1831	1832	1833	1834	1835					

130 Variables & Operators Used 2122 Occurrences

```

0001 (#####)
0002 (*)
0003 (*) SEARCH PLANNING SOFTWARE (PROGRAM #3 OF 3) (*)
0004 (*)
0005 (*) TITLE: AREA.COM (Search Area Determination Algorithm) (*)
0006 (*) VERSION: 1.0 for CP/M Operating System (*)
0007 (*) DATE WRITTEN: August 1984 (*)
0008 (*)
0009 (*) DESCRIPTION: (*)
0010 (*) - User asked for search object's last known position (*)
0011 (*) latitude and longitude (*)
0012 (*) - User asked to input search object's aerospace (if (*)
0013 (*) applicable) and surface drift vectors, previously- (*)
0014 (*) calculated in Search Planning Software Programs #1 (*)
0015 (*) and #2 (*)
0016 (*) - User asked to input confidence factors, navigational (*)
0017 (*) fix and dead-reckoning errors, and search number, as (*)
0018 (*) prescribed in the National Search and Rescue Manual (*)
0019 (*) - Program calculates search area radius, square miles, (*)
0020 (*) and the latitudes and longitudes of the center and (*)
0021 (*) four corner points. Then it creates an "audit trail"/ (*)
0022 (*) record file of program input, significant (*)
0023 (*) calculations, and output, named "AREADATA" (*)
0024 (*)
0025 (*) LICENSE: COPYRIGHT 1984 D. RICK DOUGLAS (*)
0026 (*)
0027 (*) The author makes no express or implied warranty of any (*)
0028 (*) kind with regard to this program material, including, but (*)
0029 (*) not limited to, the implied warranty of fitness for a (*)
0030 (*) particular purpose. The author shall not be liable for (*)
0031 (*) incidental or consequential damages in connection with or (*)
0032 (*) arising out of furnishing, use, or performance of this (*)
0033 (*) program. The reader MUST HAVE a solid understanding of (*)
0034 (*) search and rescue methodology before using this software (*)
0035 (*) in making decisions where human life is at risk. In fact, (*)
0036 (*) since no amount of testing can uncover 100% of program (*)
0037 (*) errors, this program is recommended for training use only. (*)
0038 (*) Prior attendance at the United States Coast Guard's (*)
0039 (*) National SAR School is highly-encouraged. (*)
0040 (*)
0041 (*) THIS SOFTWARE MAY BE FREELY-DISTRIBUTED (*)
0042 (*) PROVIDED NO FEE IS CHARGED AND (*)
0043 (*) THIS COPYRIGHT NOTICE IS RETAINED. (*)
0044 (*)
0045 (*) LANGUAGE: PASCAL (*)
0046 (*) USED : Borland International, TURBO.PAS, Version 2.0 (*)
0047 (*)
0048 (*) MODULES CALLED (Sequentially listed); (OPT) = "Optional": (*)
0049 (*)
0050 (*) Position (*)
0051 (*) AeroSurfVectors (*)

```

```

0052 (* AreaSearch *)
0053 (* FindCoordinates *)
0054 (* NewCoordinates (Aerospace Drift Vector Position; OPT) *)
0055 (* NewCoordinates (Dmax Surface Drift Position) *)
0056 (* NewCoordinates (Dmin Surface Drift Position) *)
0057 (* NewCoordinates (Search Area Center Point) *)
0058 (* FindXYsToCorner *)
0059 (* NewCoordinates (Upper Right Corner Position) *)
0060 (* NewCoordinates (Lower Left Corner Position) *)
0061 (* WriteToDisk *)
0062 (* *)
0063 (*****
0064
0065
0066
0067 program SearchArea(input,output);
0068
0069 const radians = 57.2957795; {Standard radian conversion factor}
0070
0071 var LatNS,           {Indicates whether latitude is North or South}
0072     LongEW          : char; {Indicates whether longitude is East or West}
0073
0074     SearchNumber : integer; {Indicates which search is being calculated}
0075
0076     AeroConfidence, {Aerospace vector drift error confidence factor}
0077     AeroDriftDistance, {Total aerospace drift vector distance}
0078     AeroError, {Aerospace vector drift error}
0079     DatumLatitude, {Latitude of search area center point}
0080     DatumLongitude, {Longitude of search area center point}
0081     DirAeroDrift, {Total aerospace drift vector direction}
0082     DistBetween, {Distance between computed surface drift vectors}
0083     DmaxDir, {Maximum combined surface drift direction}
0084     DminDir, {Minimum combined surface drift direction}
0085     DmaxDistance, {Maximum combined surface drift distance}
0086     DminDistance, {Minimum combined surface drift distance}
0087     DmaxLatitude, {Latitude of Dmax position}
0088     DminLatitude, {Latitude of Dmin position}
0089     DmaxLongitude, {Longitude of Dmax position}
0090     DminLongitude, {Longitude of Dmin position}
0091     DRerrorSearcher, {Search craft navigational fix error}
0092     FixErrorSearcher, {Search craft dead-reckoning error}
0093     LastLatitudeKnown, {Last known latitude of aerospace object}
0094     LastLongitudeKnown, {Last known longitude of aerospace object}
0095     LatAeroDrift, {Latitude of original position + Aerospace vector}
0096     LatLwrLeftCorner, {Latitude of search area's lower left corner}
0097     LatLwrRtCorner, {Latitude of search area's lower right corner}
0098     LatUprLeftCorner, {Latitude of search area's upper left corner}
0099     LatUprRtCorner, {Latitude of search area's upper right corner}
0100     LongAeroDrift, {Longitude of original position + Aerospace vector}
0101     LongLwrLeftCorner, {Longitude of search area's lower left corner}
0102     LongLwrRtCorner, {Longitude of search area's lower right corner}
0103     LongUprLeftCorner, {Longitude of search area's upper left corner}

```

```

0104 LongUpRtCorner,      {Longitude of search area's upper right corner}
0105 MaxSurfaceError,    {Maximum surface vector drift error}
0106 MinSurfaceError,    {Minimum surface vector drift error}
0107 NewLatitude,        {Sum of previous latitude + displacement vector}
0108 NewLongitude,       {Sum of previous longitude + displacement vector}
0109 ObjectDRerror,      {Search object dead-reckoning determination error}
0110 ObjectFixError,     {Search object navigational fix determination error}
0111 PreviousDriftErrs,  {Sum of previously-computed drift errors}
0112 AreaOfSearch,       {Total search area in nautical miles^2}
0113 SearchRadius,       {TotProbableErr * SearchNumber (Safety Factor)}
0114 SurfaceMinMax,      {Surface drift error minimax distance}
0115 SurfConfidence,     {Surface vector drift error confidence factor}
0116 TotalDriftError,    {Sum of aerospace and surface drift errors}
0117 TotObjectErr,       {Search object navigational fix + DR error}
0118 TotProbableErr,     {Total probable search area error}
0119 TotSearcherErr : real; {Search craft navigational fix + DR error}
0120
0121 AreaDat : text; {External record file of program input/output data}
0122
0123 {*****}
0124 {Writes out a specified number of blank lines; can be used to clear screen. }
0125
0126 procedure writelns (lines : integer);
0127
0128 var count : integer;
0129
0130 begin
0131     count := 0;
0132     while lines > count do
0133         begin
0134             writeln;
0135             count := count + 1
0136         end
0137     end;
0138
0139 {*****}
0140 {Queries user for last known position latitude and longitude. }
0141
0142 procedure Position;
0143
0144 begin {Position}
0145
0146     writelns(24);
0147     repeat {until valid latitude}
0148         writeln('Was search object''s last known latitude north or south?');
0149         write(' (Enter N or S)      Answer= ');
0150         readln(LatNS);
0151         writeln
0152     until (LatNS = 'N') or (LatNS = 'n') or (LatNS = 'S') or (LatNS = 's');
0153     if (LatNS = 'N') or (LatNS = 'n') then LatNS := 'N' else LatNS := 'S';
0154
0155     repeat {until valid latitude}

```

```

0156      writeln('Please enter the search object''s last known latitude ');
0157      writeln(' (For example: 25-Degrees 45-Minutes 13-Seconds = 25.4513)');
0158      write('LATITUDE = ');
0159      readln(LastLatitudeKnown);
0160      if (LastLatitudeKnown < 0) or (LastLatitudeKnown > 90) then
0161        writeln('Input latitude must be between 0-90. Try again!');
0162      writeln
0163      until (LastLatitudeKnown >= 0) and (LastLatitudeKnown <= 90);
0164
0165      repeat (until valid longitude)
0166        writeln('Was search object''s last known longitude east or west?');
0167        write(' (Enter E or W)      Answer= ');
0168        readln(LongEW);
0169        writeln
0170        until (LongEW = 'E') or (LongEW = 'e') or (LongEW = 'W') or (LongEW = 'w');
0171        if (LongEW = 'E') or (LongEW = 'e') then LongEW := 'E' else LongEW := 'W';
0172
0173      repeat (until valid longitude)
0174        writeln('Please enter the search object''s last known longitude ');
0175        writeln(' (For example: 160-Degrees 45-Minutes 13-Seconds = 160.4513)');
0176        write('LONGITUDE = ');
0177        readln(LastLongitudeKnown);
0178        if (LastLongitudeKnown < 0) or (LastLongitudeKnown > 180) then
0179          writeln('Input longitude must be between 0-180. Try again!');
0180        writeln
0181        until (LastLongitudeKnown >= 0) and (LastLongitudeKnown <= 180);
0182
0183    end; {Position}
0184
0185    {*****}
0186    {Queries user for previously-calculated Aerospace and (Max & Min) Surface }
0187    {Drift vectors. }
0188
0189    procedure AeroSurfVectors;
0190
0191    var aerodrift : char; {Indicates if aerospace drift vector used in problem}
0192
0193    begin {AeroSurfVectors}
0194
0195      writeln(24);
0196      repeat
0197        writeln('Did you previously calculate an aerospace drift vector ');
0198        write('for this problem? (y/n): ');
0199        readln(aerodrift)
0200        until (aerodrift = 'Y') or (aerodrift = 'y') or
0201              (aerodrift = 'N') or (aerodrift = 'n');
0202      writeln;
0203
0204      {Start of calculations to find Aerospace Drift Error}
0205      if (aerodrift = 'Y') or (aerodrift = 'y') then
0206        begin {Aerospace drift vector previously-calculated}
0207

```

```

0208      writeln('Please enter the previously-calculated,');
0209      repeat {until valid response}
0210          write('TOTAL AEROSPACE DRIFT DIRECTION => ');
0211          readln(DirAeroDrift)
0212      until (DirAeroDrift >= 0) and (DirAeroDrift <= 360);
0213      writeln;
0214
0215      writeln('Please enter the previously-calculated');
0216      write('TOTAL AEROSPACE DRIFT DISTANCE => ');
0217      readln(AeroDriftDistance);
0218      writeln;
0219
0220      write('Refer to the "Individual Drift Error" ');
0221      writeln('section in the National SAR');
0222      write('Manual (approximately page 8-27). ');
0223      writeln('Please enter the desired');
0224      write('Aerospace Drift Error Confidence Factor ');
0225      writeln('(e.g., 0.125, 0.3, etc.):');
0226
0227      repeat {until 0 < AeroConfidence(1)}
0228          write('AEROSPACE DRIFT ERROR CONFIDENCE FACTOR => ');
0229          readln(AeroConfidence)
0230      until (AeroConfidence > 0) and (AeroConfidence < 1);
0231      writeln;
0232
0233      AeroError := AeroDriftDistance * AeroConfidence
0234      end; {Aerospace drift vector previously-calculated}
0235      {End of calculations to find Aerospace Drift Error}
0236
0237      writeln(24);
0238      writeln('Please enter the sum of previous drift errors');
0239      writeln('(Enter a zero if not applicable):');
0240      write('SUM OF DRIFT ERRORS => ');
0241      readln(PreviousDriftErrs);
0242      writeln;
0243
0244      writeln('Please enter the previously-calculated,');
0245      repeat {until valid response}
0246          write('MINIMUM TOTAL SURFACE DRIFT DIRECTION => ');
0247          readln(DminDir)
0248      until (DminDir >= 0) and (DminDir <= 360);
0249      writeln;
0250
0251      writeln('Please enter the previously-calculated,');
0252      write('MINIMUM TOTAL SURFACE DRIFT DISTANCE => ');
0253      readln(DminDistance);
0254      writeln;
0255
0256      writeln('Please enter the previously-calculated,');
0257      repeat {until valid response}
0258          write('MAXIMUM TOTAL SURFACE DRIFT DIRECTION => ');
0259          readln(DmaxDir)

```

```

0260     until (DmaxDir = 0) and (DmaxDir = 360);
0261     writeln;
0262
0263     writeln('Please enter the previously-calculated,');
0264     write('MAXIMUM TOTAL SURFACE DRIFT DISTANCE = ');
0265     readln(DmaxDistance);
0266     writeln;
0267
0268     writeln('Please enter the measured distance between the two, previously-');
0269     writeln('calculated, Total Surface Drift Distance positions:');
0270     write('DISTANCE BETWEEN SURFACE DRIFT POSITIONS = ');
0271     readln(DistBetween);
0272     writeln
0273 end; {AeroSurfVectors}
0274
0275 {*****}
0276 {When given X & Y components and a reference latitude/longitude by the calling}
0277 {procedures FindCoordinates and FindXYsToCorner, this procedure calculates }
0278 {the updated position's latitude/longitude. }
0279
0280 procedure NewCoordinates(var Xcomponent, Ycomponent : real;
0281                          TempLatitude, TempLongitude : real);
0282
0283 var DecimalLatitude,    {Latitude converted from Deg-Min-Sec to decimal}
0284     DecimalLongitude,   {Longitude converted from Deg-Min-Sec to decimal}
0285     Temp,               {Interim term used in calculations}
0286     TempDegree,         {Interim term for position in degrees only}
0287     TempDenom,          {Interim term for denominator}
0288     TempMin,            {Interim term for position in minutes only}
0289     TempMinSec,         {Interim term for position minutes and seconds}
0290     TempSec : real;     {Interim term for position in seconds only}
0291
0292 begin {NewCoordinates}
0293     {initialize variables}
0294     Temp := 0.0; TempDegree := 0.0; TempDenom := 0.0; TempMin := 0.0;
0295     TempMinSec := 0.0; TempSec := 0.0;
0296
0297     {Convert X & Y component sign values if LatNS = N or LongEW = W}
0298     if LatNS = 'N' then Xcomponent := Xcomponent * -1.0;
0299
0300     {Convert passed latitude to decimal}
0301     TempDegree := trunc(TempLatitude);
0302     TempMinSec := TempLatitude - TempDegree;
0303     DecimalLatitude := TempDegree + TempMinSec * (1 + 2/3);
0304
0305     {Combine passed, decimal latitude and vector to find new, decimal latitude}
0306     Ycomponent := Ycomponent / 60;
0307     TempLatitude := DecimalLatitude + Ycomponent;
0308
0309     {Convert new, decimal latitude to degrees-minutes-seconds}
0310     TempDegree := trunc(TempLatitude);
0311     TempMinSec := TempLatitude - TempDegree;

```

```

0312     Temp := TempMinSec * 60;
0313     TempMin := trunc(Temp);
0314     TempSec := (Temp - TempMin) * 60;
0315     NewLatitude := TempDegree + TempMin/100 + TempSec/10000;
0316
0317     {Convert passed longitude to decimal}
0318     TempDegree := trunc(TempLongitude);
0319     TempMinSec := TempLongitude - TempDegree;
0320     Decimallongitude := TempDegree + TempMinSec * (1 + 2/3);
0321
0322     {Combine decimal-latitude with vector to find new, decimal longitude}
0323     TempDenom := cos(TempLatitude / radians) * 60;
0324     TempLongitude := Xcomponent / TempDenom + Decimallongitude;
0325
0326     {Convert new, decimal longitude to degrees-minutes-seconds}
0327     TempDegree := trunc(TempLongitude);
0328     TempMinSec := TempLongitude - TempDegree;
0329     Temp := TempMinSec * 60;
0330     TempMin := trunc(Temp);
0331     TempSec := (Temp - TempMin) * 60;
0332     NewLongitude := TempDegree + TempMin/100 + TempSec/10000
0333
0334 end; {NewCoordinates}
0335
0336 {*****}
0337 {Finds the X & Ycomponents of the search-area-center-point-displacement- }
0338 {vectors created by the calling procedure, FindCoordinates. Then, it passes }
0339 {these components to procedure NewCoordinates for it to determine search area}
0340 {corner point latitudes/longitudes. }
0341
0342 procedure FindXYsToCorner(var Vector1, Vector2 : integer);
0343
0344 var BrgRads,           {Individual compass headings converted to radians}
0345     Xcomponent,        {Horizontal component of individual vectors}
0346     Ycomponent : real; {Vertical component of individual vectors}
0347
0348 begin {FindXYsToCorner}
0349     BrgRads := Vector1 / radians;
0350     Xcomponent := sin(BrgRads) * SearchRadius;
0351     Ycomponent := cos(BrgRads) * SearchRadius;
0352     BrgRads := Vector2 / radians;
0353     Xcomponent := Xcomponent + sin(BrgRads) * SearchRadius;
0354     Ycomponent := Ycomponent + cos(BrgRads) * SearchRadius;
0355     NewCoordinates(Xcomponent, Ycomponent, DatumLatitude, DatumLongitude)
0356
0357 end; {FindXYsToCorner}
0358
0359 {*****}
0360 {Calculates X & Ycomponents of aerospace and surface drift vectors and sends }
0361 {them to procedure NewCoordinates for it to determine the search area center }
0362 {point latitude/longitude. Next, this procedure creates two vectors (1 & 2), }
0363 {90-degrees apart, and of length equal to the search area's radius. It sends}

```

```

0364 {these two vectors to procedure FindXYsToCorner for it to calculate the }
0365 {vectors' X & Ycomponents enroute to procedure NewCoordinates, which finds }
0366 {the corner point latitudes/longitudes. }
0367
0368 procedure FindCoordinates;
0369
0370 var Vector1,           {Used to find get to search area corner points}
0371     Vector2           : integer; {Used to find get to search area corner points}
0372     BrgRads,          {Individual compass headings converted to radians}
0373     TempX,            {Xcomponent used for interim calculations only}
0374     TempY,            {Ycomponent used for interim calculations only}
0375     Xcomponent,       {Horizontal component of individual vectors}
0376     Ycomponent        : real;    {Vertical component of individual vectors}
0377
0378 begin {FindCoordinates}
0379     TempX := 0.0; TempY := 0.0; {initialize variables}
0380
0381     if AeroDriftDistance > 0 then
0382     begin {Find updated coordinates of Aerospace Drift Vector displacement}
0383         BrgRads := DirAeroDrift / radians;
0384         Xcomponent := sin(BrgRads) * AeroDriftDistance;
0385         Ycomponent := cos(BrgRads) * AeroDriftDistance;
0386         TempX := Xcomponent; TempY := Ycomponent; {Used below}
0387         NewCoordinates(Xcomponent, Ycomponent, LastLatitudeKnown,
0388             LastLongitudeKnown);
0389         LatAeroDrift := NewLatitude;
0390         LongAeroDrift := NewLongitude;
0391     end; {Find updated coordinates of Aerospace Drift Vector displacement}
0392
0393     {Find updated coordinates of Maximum Surface Drift Vector displacement}
0394     BrgRads := DmaxDir / radians;
0395     Xcomponent := TempX + sin(BrgRads) * DmaxDistance;
0396     Ycomponent := TempY + cos(BrgRads) * DmaxDistance;
0397     NewCoordinates(Xcomponent, Ycomponent, LastLatitudeKnown, LastLongitudeKnown);
0398     DmaxLatitude := NewLatitude;
0399     DmaxLongitude := NewLongitude;
0400
0401     {Find updated coordinates of Minimum Surface Drift Vector displacement}
0402     BrgRads := DminDir / radians;
0403     Xcomponent := TempX + sin(BrgRads) * DminDistance;
0404     Ycomponent := TempY + cos(BrgRads) * DminDistance;
0405     NewCoordinates(Xcomponent, Ycomponent, LastLatitudeKnown, LastLongitudeKnown);
0406     DminLatitude := NewLatitude;
0407     DminLongitude := NewLongitude;
0408
0409     {Find resultant X & Ycomponent of Aerospace and Surface Dmax and Dmin vectors}
0410     BrgRads := DmaxDir / radians;
0411     Xcomponent := TempX + sin(BrgRads) * DmaxDistance;
0412     Ycomponent := TempY + cos(BrgRads) * DmaxDistance;
0413     BrgRads := DminDir / radians;
0414     Xcomponent := Xcomponent + sin(BrgRads) * DminDistance;
0415     Ycomponent := Ycomponent + cos(BrgRads) * DminDistance;

```

```

0416
0417 {Find search area center point latitude/longitude}
0418 NewCoordinates(Xcomponent,Ycomponent,LastLatitudeKnown,LastLongitudeKnown);
0419 DatumLatitude := NewLatitude;
0420 DatumLongitude := NewLongitude;
0421
0422 {Find search area corner points using perpendicular vectors from center point}
0423 Vector1 := 360; Vector2 := 090; {Vectors to upper right corner}
0424 FindXYsToCorner(Vector1,Vector2);
0425 LatUpRtCorner := NewLatitude;
0426 LongUpRtCorner := NewLongitude;
0427
0428 Vector1 := 180; Vector2 := 270; {Vectors to lower left corner}
0429 FindXYsToCorner(Vector1,Vector2);
0430 LatLwrLeftCorner := NewLatitude;
0431 LongLwrLeftCorner := NewLongitude;
0432
0433 LatLwrRtCorner := LatLwrLeftCorner; {Lower right corner coordinates}
0434 LongLwrRtCorner := LongUpRtCorner;
0435
0436 LatUpLftCorner := LatUpRtCorner; {Upper left corner coordinates}
0437 LongUpLftCorner := LongLwrLeftCorner
0438
0439 end; {FindCoordinates}
0440
0441 {*****}
0442 {Given the search object's aerospace and/or surface drift vector(s), this }
0443 {procedure calculates the search area. }
0444
0445 procedure AreaSearch;
0446
0447 var continue : char; {Pauses program until user continues it}
0448
0449 begin {AreaSearch}
0450
0451 {Start of calculations to find Surface Drift Error}
0452 writeIn('Refer to the "Individual Drift Error" section in the National SAR');
0453 write('Manual (approximately page 8-27). ');
0454 writeIn('Please enter the desired');
0455 writeIn('Surface Drift Error Confidence Factor (e.g. 0.125, 0.3, etc.):');
0456
0457 repeat {until 0(SurfConfidence(1 )
0458     write('SURFACE DRIFT ERROR CONFIDENCE FACTOR = ');
0459     readln(SurfConfidence)
0460 until (SurfConfidence > 0) and (SurfConfidence < 1);
0461 writeIn;
0462
0463 MinSurfaceError := DminDistance * SurfConfidence;
0464 MaxSurfaceError := DmaxDistance * SurfConfidence;
0465 SurfaceMiniMax := (MinSurfaceError + MaxSurfaceError + DistBetween +
0466     PreviousDriftErrs) / 2;
0467 {End of calculations to find Surface Drift Error}

```

```

0468
0469 TotalDriftError := AeroError + SurfaceMinMax;
0470
0471 writeln(24);
0472 writeln('Refer to the "Initial Position Error" section in the National SAR');
0473 write('Manual (approximately page 8-29). ');
0474 writeln('Please enter the search object's');
0475 writeln('Navigational Fix Error (e.g, 0, 5, 10, or 15 naut. mi. radius),');
0476 writeln('and, if applicable, the Navigational Dead-Reckoning (DR) Error');
0477 writeln('(e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):');
0478 write('SEARCH OBJECT NAVIGATIONAL FIX ERROR = ');
0479 readln(ObjectFixError);
0480 writeln;
0481 write('SEARCH OBJECT NAVIGATIONAL DR ERROR = ');
0482 readln(ObjectDRError);
0483 writeln;
0484 TotObjectErr := ObjectFixError + ObjectDRError;
0485
0486 writeln(24);
0487 writeln('Refer to the "Search Craft Error" section in the National SAR');
0488 write('Manual (approximately page 8-29). ');
0489 writeln('Please enter the search craft's');
0490 writeln('Navigational Fix Error (e.g, 0, 5, 10, or 15 naut. mi. radius),');
0491 writeln('and, if applicable, the Navigational Dead-Reckoning (DR) Error');
0492 writeln('(e.g, 0, 5, 10, or 15 percent of DR distance in naut. mi.):');
0493 write('SEARCH CRAFT NAVIGATIONAL FIX ERROR = ');
0494 readln(FixErrorSearcher);
0495 writeln;
0496 write('SEARCH CRAFT NAVIGATIONAL DR ERROR = ');
0497 readln(DRerrorSearcher);
0498 writeln;
0499 TotSearcherErr := FixErrorSearcher + DRerrorSearcher;
0500
0501 TotProbableErr := TotalDriftError * TotalDriftError;
0502 TotProbableErr := TotProbableErr + (TotObjectErr * TotObjectErr);
0503 TotProbableErr := TotProbableErr + (TotSearcherErr * TotSearcherErr);
0504 TotProbableErr := sqrt(TotProbableErr);
0505
0506 writeln(24);
0507 writeln('You are calculating the search area for:');
0508 writeln;
0509 writeln('      1 The first search');
0510 writeln('      2 The second search');
0511 writeln('      3 The third search');
0512 writeln('      4 The fourth search');
0513 writeln('      5 The fifth search');
0514 writeln;
0515 repeat (until valid response)
0516     write('Input SEARCH NUMBER = ');
0517     readln(SearchNumber)
0518 until (SearchNumber > 0) and (SearchNumber < 6);
0519

```

```

0520     if SearchNumber = 1 then SearchRadius := TotProbableErr * 1.1
0521     else if SearchNumber = 2 then SearchRadius := TotProbableErr * 1.6
0522     else if SearchNumber = 3 then SearchRadius := TotProbableErr * 2.0
0523     else if SearchNumber = 4 then SearchRadius := TotProbableErr * 2.3
0524     else if SearchNumber = 5 then SearchRadius := TotProbableErr * 2.5;
0525     if (SearchRadius - trunc(SearchRadius) ) > 0 then
0526         SearchRadius := trunc(SearchRadius) + 1.0;
0527     AreaOfSearch := 4 * (SearchRadius * SearchRadius);
0528
0529     writeln;
0530     writeln;
0531     writeln('PRELIMINARY INFORMATION:');
0532     writeln('-----');
0533     writeln('Search Area Radius = ',SearchRadius:5:0,' naut. mi. ');
0534     writeln('Search Area      = ',AreaOfSearch:5:0,' sq-naut.mi. ');
0535     writeln(5);
0536     write('HIT RETURN (Once or twice, as necessary) TO CONTINUE');
0537     readln(continue);
0538
0539     writeln(24);
0540     writeln(4);
0541     writeln('Calculating . . . Please stand by')
0542
0543 end; {AreaSearch}
0544
0545 {*****}
0546 {Prints out record of error adjustments to calculations of search radius and }
0547 {area.                                     }
0548
0549 procedure WriteToDisk;
0550
0551 begin {WriteToDisk}
0552     assign(AreaDat,'AreaData');
0553     rewrite(AreaDat);
0554
0555     write(AreaDat,'-----');
0556     writeln(AreaDat,'-----');
0557     write(AreaDat,'DRIFT COORDINATES      : ');
0558     writeln(AreaDat,'LATITUDE      LONGITUDE');
0559
0560     write(AreaDat,'Last Known Position      : ');
0561     write(AreaDat,'LastLatitudeKnown:7:4,' ');
0562     writeln(AreaDat,'LastLongitudeKnown:7:4);
0563
0564     if AeroDriftDistance > 0 then
0565     begin {Aerospace vector known}
0566         write(AreaDat,'Aerospace Drift Position      : ');
0567         write(AreaDat,'LatAeroDrift:7:4,' ');
0568         writeln(AreaDat,'LongAeroDrift:7:4)
0569     end; {Aerospace vector known}
0570
0571     write(AreaDat,'Surface Dmin Position      : ');

```

```

0572 write(AreaDat,DminLatitude:7:4,' ');
0573 writeln(AreaDat,DminLongitude:7:4);
0574 write(AreaDat,'Surface Dmax Position : ');
0575 write(AreaDat,DmaxLatitude:7:4,' ');
0576 writeln(AreaDat,DmaxLongitude:7:4);
0577 writeln(AreaDat,' ');
0578
0579 write(AreaDat,'=====');
0580 writeln(AreaDat,'=====');
0581 writeln(AreaDat,'SEARCH AREA:');
0582 writeln(AreaDat,' ');
0583
0584 if AeroDriftDistance > 0 then
0585 begin {Aerospace vector known}
0586 write(AreaDat,'Aerospace Drift Distance : ');
0587 writeln(AreaDat,AeroDriftDistance:5:2,' naut. mi. ');
0588 write(AreaDat,'Drift Error Confidence : ');
0589 writeln(AreaDat,AeroConfidence:5:3);
0590 write(AreaDat,'Aerospace Drift Error : ');
0591 writeln(AreaDat,AeroError:5:2,' naut. mi. ');
0592 write(AreaDat,'-----');
0593 writeln(AreaDat,'-----');
0594 end; {Aerospace vector known}
0595
0596 write(AreaDat,'Sum of Previous Drift Errors : ');
0597 writeln(AreaDat,PreviousDriftErrs:5:2,' naut. mi. ');
0598 write(AreaDat,'Surface Drift Distance : ',DminDistance:5:2);
0599 writeln(AreaDat,' naut. mi. ',DmaxDistance:5:2,' naut. mi. ');
0600 writeln(AreaDat,'Drift Error Confidence : ',SurfConfidence:5:3);
0601 write(AreaDat,'Drift Error Min and Max : ',MinSurfaceError:5:2);
0602 writeln(AreaDat,' naut. mi. ',MaxSurfaceError:5:2,' naut. mi. ');
0603 write(AreaDat,'Distance Between Dmin/max : ');
0604 writeln(AreaDat,DistBetween:5:2,' naut. mi. ');
0605 write(AreaDat,'Surface Drift Error Minimax : ');
0606 writeln(AreaDat,SurfaceMiniMax:5:2,' naut. mi. ');
0607 write(AreaDat,'-----');
0608 writeln(AreaDat,'-----');
0609
0610 write(AreaDat,'Total Drift Error : ');
0611 writeln(AreaDat>TotalDriftError:5:2,' naut. mi. ');
0612 write(AreaDat,'-----');
0613 writeln(AreaDat,'-----');
0614
0615 write(AreaDat,'Object Navigation Fix Error : ');
0616 writeln(AreaDat,ObjectFixError:5:0,' naut. mi. ');
0617 write(AreaDat,'Object Dead-Reckoning Error : ');
0618 writeln(AreaDat,ObjectDRError:5:2,' naut. mi. ');
0619 write(AreaDat,'Object Total Position Error : ');
0620 writeln(AreaDat,TotObjectErr:5:2,' naut. mi. ');
0621 write(AreaDat,'-----');
0622 writeln(AreaDat,'-----');
0623

```

```

0624 write(AreaDat,'Searcher Navigation Fix Error : ');
0625 writeln(AreaDat,FixErrorSearcher:5:0,' naut. mi. ');
0626 write(AreaDat,'Searcher Dead-Reckoning Error : ');
0627 writeln(AreaDat,DRErrorSearcher:5:2,' naut. mi. ');
0628 write(AreaDat,'Searcher Total Position Error : ');
0629 writeln(AreaDat,TotSearcherErr:5:2,' naut. mi. ');
0630 write(AreaDat,'-----');
0631 writeln(AreaDat,'-----');
0632
0633 write(AreaDat,'Total Probable Error : ');
0634 writeln(AreaDat,TotProbableErr:5:2,' naut. mi. ');
0635
0636 if SearchNumber = 1 then
0637     writeln(AreaDat,'Safety Factor : 1.1')
0638 else if SearchNumber = 2 then
0639     writeln(AreaDat,'Safety Factor : 1.6')
0640 else if SearchNumber = 3 then
0641     writeln(AreaDat,'Safety Factor : 2.0')
0642 else if SearchNumber = 4 then
0643     writeln(AreaDat,'Safety Factor : 2.3')
0644 else if SearchNumber = 5 then
0645     writeln(AreaDat,'Safety Factor : 2.5');
0646
0647 write(AreaDat,'-----');
0648 writeln(AreaDat,'-----');
0649 write(AreaDat,'Search Radius : ');
0650 writeln(AreaDat,SearchRadius:5:0,' naut. mi. ');
0651 write(AreaDat,'Search Area : ');
0652 writeln(AreaDat,AreaOfSearch:5:0,' naut. mi.-squared. ');
0653
0654 write(AreaDat,'-----');
0655 writeln(AreaDat,'-----');
0656 write(AreaDat,'SEARCH AREA COORDINATES: : ');
0657 writeln(AreaDat,'LATITUDE LONGITUDE');
0658
0659 write(AreaDat,'Center Point : ');
0660 write(AreaDat,DatumLatitude:7:4,' ');
0661 writeln(AreaDat,DatumLongitude:7:4);
0662 writeln(AreaDat,' ');
0663
0664 write(AreaDat,'Corner Point (Upper Left) : ');
0665 write(AreaDat,LatUpRtCorner:7:4,' ');
0666 writeln(AreaDat,LongUpRtCorner:7:4);
0667
0668 write(AreaDat,'Corner Point (Lower Left) : ');
0669 write(AreaDat,LatLwrLeftCorner:7:4,' ');
0670 writeln(AreaDat,LongLwrLeftCorner:7:4);
0671
0672 write(AreaDat,'Corner Point (Upper Right) : ');
0673 write(AreaDat,LatUpRtCorner:7:4,' ');
0674 writeln(AreaDat,LongUpRtCorner:7:4);
0675

```

```

0676 write(AreaDat,'Corner Point (Lower Right) : ');
0677 write(AreaDat,LatLwrRtCorner:7:4,' ');
0678 writeln(AreaDat,LongLwrRtCorner:7:4);
0679
0680 write(AreaDat,'=====');
0681 writeln(AreaDat,'=====');
0682
0683 close(AreaDat)
0684 end; {WriteToDisk}
0685
0686 {*****}
0687 {Provides user with program information, limitations on use and license. }
0688
0689 procedure Warranty;
0690
0691 var continue : char; {Bogus read variable provides time to read warranty}
0692
0693 begin {Warranty}
0694
0695 write('*****');
0696 writeln('*****');
0697 write('(* SEARCH PLANNING SOFTWARE ');
0698 writeln('PROGRAM #3 OF 3) *');
0699 write('(* TITLE: AREA.COM (Search Area ');
0700 writeln('Determination Algorithm) *');
0701 write('(* VERSION: 1.0 for CP/M ');
0702 writeln('Operating System *');
0703 write('(* DATE WRITTEN: August 1984');
0704 writeln(' *');
0705 write('(* LICENSE: COPYRIGHT 1984');
0706 writeln(' D. RICK DOUGLAS *');
0707 write('*****');
0708 writeln('*****');
0709 write('The author makes no express or implied ');
0710 writeln('warranty of any kind with regard to');
0711 write('this program material, including, but ');
0712 writeln('not limited to, the implied warranty of');
0713 write('fitness for a particular purpose. ');
0714 writeln('The author shall not be liable for');
0715 write('incidental or consequential damages ');
0716 writeln('in connection with or arising out of');
0717 write('furnishing, use, or performance of this ');
0718 writeln('program. The reader MUST HAVE a solid');
0719 write('understanding of search and rescue ');
0720 writeln('methodology before using this software in');
0721 write('making decisions where human life is at ');
0722 writeln('risk. In fact, since no amount of');
0723 write('testing can uncover 100% of program ');
0724 writeln('errors, this program is recommended for');
0725 write('training use only. Prior attendance ');
0726 writeln('at the United States Coast Guard's');
0727 writeln('National SAR School is highly-encouraged.');
```

```

0728     writeln;
0729     write('*****');
0730     writeln(' WARNING! *****');
0731     write('(*      THIS SOFTWARE MAY BE FREELY-');
0732     writeln('DISTRIBUTED PROVIDED NO FEE      *');
0733     write('(*              IS CHARGED AND THIS');
0734     writeln('COPYRIGHT NOTICE IS RETAINED      *');
0735     write('*****');
0736     writeln('*****');
0737     writeln;
0738     write('PLEASE HIT RETURN (Once or twice, as necessary) TO CONTINUE');
0739     readln(continue)
0740
0741 end;  {Warranty}
0742
0743 {*****}
0744
0745 begin {Main Program}
0746
0747     {Initialize program variables}
0748     AeroConfidence := 0.0; AeroDriftDistance := 0.0; AeroError := 0.0;
0749     DatumLatitude := 0.0; DatumLongitude := 0.0; DirAeroDrift := 0.0;
0750     DistBetween := 0.0; DmaxDir := 0.0; DmaxDir := 0.0;
0751     DmaxDistance := 0.0; DminDistance := 0.0; DmaxLatitude := 0.0;
0752     DmaxLongitude := 0.0; DminLatitude := 0.0; DminLongitude := 0.0;
0753     DRerrorSearcher := 0.0; FixErrorSearcher := 0.0; LastLatitudeKnown := 0.0;
0754     LastLongitudeKnown := 0.0; LatAeroDrift := 0.0; LatLwrLeftCorner := 0.0;
0755     LatLwrRtCorner := 0.0; LatNS := 'Q'; LatUpRtCorner := 0.0;
0756     LatUpRtCorner := 0.0; LongAeroDrift := 0.0;
0757     LongEW := 'Q'; LongLwrLeftCorner := 0.0; LongLwrRtCorner := 0.0;
0758     LongUpRtCorner := 0.0; LongUpRtCorner := 0.0; MaxSurfaceError := 0.0;
0759     MinSurfaceError := 0.0; NewLatitude := 0.0; NewLongitude := 0.0;
0760     ObjectDRerror := 0.0; ObjectFixError := 0.0; PreviousDriftErrs := 0.0;
0761     AreaOfSearch := 0.0; SearchNumber := 0; SearchRadius := 0.0;
0762     SurfaceMinMax := 0.0; SurfConfidence := 0.0; TotalDriftError := 0.0;
0763     TotObjectErr := 0.0; TotProbableErr := 0.0; TotSearcherErr := 0.0;
0764
0765     Warranty;
0766     Position;
0767     AeroSurfVectors;
0768     AreaSearch;
0769     FindCoordinates;
0770     WriteToDisk;
0771     writeln(24);
0772     writeln('A record of significant input and output data used during this');
0773     writeln('program run is stored in an external file named "AREADATA".');
0774     writeln('If you desire to keep this record permanently, please rename');
0775     writeln('file AREADATA before running this program again!')
0776
0777 end.  {Main Program}

```

Search Area Determination Program (#3 of 3)
Variable & Operator Cross-Reference Listing

<u>Variable</u>	<u>Program Line Number</u>									
AeroConfidence	76	229	230	230	233	589	748			
aerodrift	191	199	200	200	201	201	205	205		
AeroDriftDistance	77	217	233	381	384	385	564	584	587	748
AeroError	78	233	469	591	748					
AeroSurfVectors	189	767								
AreaDat	121	552	553	555	556	557	558	560	561	562
	566	567	568	571	572	573	574	575	576	577
	579	580	581	582	586	587	588	589	590	591
	592	593	596	597	598	599	600	601	602	603
	604	605	606	607	608	610	611	612	613	615
	616	617	618	619	620	621	622	624	625	626
	627	628	629	630	631	633	634	637	639	641
	643	645	647	648	649	650	651	652	654	655
	656	657	659	660	661	662	664	665	666	668
	669	670	672	673	674	676	677	678	680	681
	683									
AreaOfSearch	112	527	534	652	761					
AreaSearch	445	768								
BrgRads	344	349	350	351	352	353	354	372	383	384
	385	394	395	396	402	403	404	410	411	412
	413	414	415							
continue	447	537	691	739						
count	120	131	132	135	135					
DRerrorSearcher	91	497	499	627	753					
DatumLatitude	79	355	419	660	749					
DatumLongitude	80	355	420	661	749					
DecimalLatitude	283	303	307							
DecimalLongitude	284	320	324							
DirAeroDrift	81	211	212	212	383	749				

DistBetween	82	271	465	604	750					
DmaxDir	83	259	260	260	394	410	750	750		
DmaxDistance	85	265	395	396	411	412	464	599	751	
DmaxLatitude	87	398	575	751						
DmaxLongitude	89	399	576	752						
DminDir	84	247	248	248	402	413				
DminDistance	86	253	403	404	414	415	463	598	751	
DminLatitude	88	406	572	752						
DminLongitude	90	407	573	752						
FindCoordinates	368	769								
FindXYsToCorner	342	424	429							
FixErrorSearcher	92	494	499	625	753					
LastLatitudeKnown	93 561	159 753	160	160	163	163	387	397	405	418
LastLongitudeKnown	94 562	177 754	178	178	181	181	388	397	405	418
LatAeroDrift	95	389	567	754						
LatLwrLeftCorner	96	430	433	669	754					
LatLwrRtCorner	97	433	677	755						
LatNS	71 298	150 755	152	152	152	152	153	153	153	153
LatUpLwrLeftCorner	98	436	665	755						
LatUpLwrRtCorner	99	425	436	673	756					
lines	126	132								
LongAeroDrift	100	390	568	756						
LongEW	72 757	168	170	170	170	170	171	171	171	171
LongLwrLeftCorner	101	431	437	670	757					

LongLwrRtCorner	102	434	678	757							
LongUpLftCorner	103	437	666	758							
LongUpRtCorner	104	426	434	674	758						
MaxSurfaceError	105	464	465	602	758						
MinSurfaceError	106	463	465	601	759						
NewCoordinates	280	355	387	397	405	418					
NewLatitude	107	315	389	398	406	419	425	430	759		
NewLongitude	108	332	390	399	407	420	426	431	759		
ObjectDRerror	109	482	484	618	760						
ObjectFixError	110	479	484	616	760						
Position	142	766									
PreviousDriftErrs	111	241	466	597	760						
radians	69	323	349	352	383	394	402	410	413		
SearchArea	67										
SearchNumber	74	517	518	518	520	521	522	523	524	636	
	638	640	642	644	761						
SearchRadius	113	350	351	353	354	520	521	522	523	524	
	525	525	526	526	527	527	533	650	761		
SurfConfidence	115	459	460	460	463	464	600	762			
SurfaceMiniMax	114	465	469	606	762						
Temp	285	294	312	313	314	329	330	331			
TempDegree	286	294	301	302	303	310	311	315	318	319	
	320	327	328	332							
TempDenom	287	294	323	324							
TempLatitude	281	301	302	307	310	311	323				
TempLongitude	281	318	319	324	327	328					
TempMin	288	294	313	314	315	330	331	332			
TempMinSec	289	295	302	303	311	312	319	320	328	329	

TempSec	290	295	314	315	331	332				
TempX	373	379	386	395	403	411				
TempY	374	379	386	396	404	412				
TotObjectErr	117	484	502	502	620	763				
TotProbableErr	118	501	502	502	503	503	504	504	520	521
	522	523	524	634	763					
TotSearcherErr	119	499	503	503	629	763				
TotalDriftError	116	469	501	501	611	762				
Vector1	342	349	370	423	424	428	429			
Vector2	342	352	371	423	424	428	429			
Warranty	689	765								
writeln	126	146	195	237	471	486	506	535	539	540
	771									
WriteToDisk	549	770								
Xcomponent	280	298	298	324	345	350	353	353	355	375
	384	386	387	395	397	403	405	411	414	414
	418									
Ycomponent	280	386	386	387	346	351	354	354	355	376
	385	386	387	396	397	404	405	412	415	415
	418									

<u>Operator</u>	<u>Program Line Number</u>									
assign	552									
char	72	191	447	691						
close	683									
cos	323	351	354	385	396	404	412	415		
input	67									
integer	74	126	128	342	371					
output	67									

readln	150	159	168	177	199	211	217	229	241	247
	253	259	265	271	459	479	482	494	497	517
	537	739								
real	119	280	281	290	346	376				
rewrite	553									
sin	350	353	384	395	403	411	414			
sqrt	504									
text	121									
trunc	301	310	313	318	327	330	525	526		
write	149	158	167	176	190	210	216	220	222	224
	228	240	246	252	258	264	270	453	458	473
	478	481	488	493	496	516	536	555	557	560
	561	566	567	571	572	574	575	579	586	588
	590	592	596	598	601	603	605	607	610	612
	615	617	619	621	624	626	628	630	633	647
	649	651	654	656	659	660	664	665	668	669
	672	673	676	677	680	695	697	699	701	703
	705	707	709	711	713	715	717	719	721	723
	725	729	731	733	735	738				
writeln	134	148	151	156	157	161	162	166	169	174
	175	179	180	197	202	208	213	215	218	221
	223	225	231	238	239	242	244	249	251	254
	256	261	263	266	268	269	272	452	454	455
	461	472	474	475	476	477	480	483	487	489
	490	491	492	495	498	507	508	509	510	511
	512	513	514	529	530	531	532	533	534	541
	556	558	562	568	573	576	577	580	581	582
	587	589	591	593	597	599	600	602	604	606
	608	611	613	616	618	620	622	625	627	629
	631	634	637	639	641	643	645	648	650	652
	655	657	661	662	666	670	674	678	681	696
	698	700	702	704	706	708	710	712	714	716
	718	720	722	724	726	727	728	730	732	734
	736	737	772	773	774	775				

96 Variables & Operators Used 951 Occurrences

Vita

Don Richard "Rick" Douglas was born in Salt Lake City, Utah on August 4, 1953. After graduating in 1975 from the United States Air Force Academy with a Bachelor of Science Degree in history, he attended helicopter pilot training in Alabama. He served as a Combat Rescue Aircraft Commander in USAF HH-53 "Super Jolly Green Giants" in Japan (air-sea rescue) from 1976-79 and USAF UH-1N "Twin Hueys" in Utah (mountain & desert rescue) from 1979-81. He was then "loaned" to the United States Coast Guard (USAF Exchange Officer Program), where he again flew air-sea rescue missions in USCG HH-3F "Pelicans" in Florida from 1981-83. He is credited with saving 20 lives and assisting another 26 to safety during 30 separate air search and rescue missions.

Rick is a graduate of the following rescue-related training courses: USAF Basic Survival in Colorado, USAF Jungle Survival in the Philippines, USN Aircraft Underwater Egress in Florida, USN SCUBA and Drown-Proofing in California, USA and West German Airborne, and USCG National Search and Rescue School in New York.

He is married to a criminologist and ex-New Orleans Playboy Bunny, the former Paula J. Hartzfeld of Bronaugh, Missouri. Their son, Craig Madison, is also credited with a "save". Craig received numerous newspaper and Scouting awards for saving the life of a Utah man who was seriously-injured in a highway accident in 1979.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS			
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE						
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GSO/MATH/84D-1			5. MONITORING ORGANIZATION REPORT NUMBER(S)			
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/EN		7a. NAME OF MONITORING ORGANIZATION			
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433			7b. ADDRESS (City, State and ZIP Code)			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State and ZIP Code)			10. SOURCE OF FUNDING NOS.			
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
11. TITLE (Include Security Classification) See Box #19 (Unclassified)						
12. PERSONAL AUTHOR(S) Douglas, Don Richard						
13a. TYPE OF REPORT Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Yr., Mo., Day) 1984 December 14		
				15. PAGE COUNT 186		
16. SUPPLEMENTARY NOTATION						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB. GR.	Coast Guard, Computer Applications, Microcomputer Pascal, Search and Rescue.			
08	03					
06	07					
19. ABSTRACT (Continue on reverse if necessary and identify by block number)						
Title: Microcomputer Application of Aerospace Asset Surface Search Planning Approved for public release: LAW AFR 180-17. LYNN E. WOLAVER Dean for Research and Professional Development Air Force Institute of Technology (AFIT)						
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ OTIC USERS ☐			21. ABSTRACT SECURITY CLASSIFICATION Unclassified			
22a. NAME OF RESPONSIBLE INDIVIDUAL DR. PANNA B. NAGARSENKER			22b. TELEPHONE NUMBER (Include Area Code) 513-255-7210		22c. OFFICE SYMBOL AFIT/ENC	

This paper tackles the most time-consuming and complicated type of search and recovery planning -- calculating the approximate surface position of an aerospace object which has been affected over time by glide or parachute winds aloft, as well as surface current winds, leeway drift, and sea current vectors. The three, highly-interactive, search applications programs herein are written in Standard Pascal using Borland International's "TURBO Pascal" (an inexpensive software package available for virtually every microcomputer on the market). They have been tested on a small, portable, 64K memory, Z-80A processor-based microcomputer (Osborne One), a Convergent Technologies C-3 Data System, and a Digital Equipment Corporation VAX 11-780 mainframe.

Search and Rescue/Recovery (SAR) in the United States is based on the humanitarian principle which compels people to render aid to those in distress. Search planning guidelines and formulae to help locate persons in distress or missing aerospace objects are described in the National Search and Rescue Manual (AFM 84-2). This methodology has not been implemented for microcomputers in a compiled, transportable programming language like Pascal. This research project does just that. It does not, however, teach the guidelines or formulae. The reader **MUST** have a solid understanding of SAR methodology before using the attached software package to assist in making decisions where human life is at risk. In fact, since no amount of testing can uncover 100% of program errors, the attached software package is recommended for training use only.

Appendices include: glossary, user's guide, sample problem with program runs and solutions, and source code listing.

END

FILMED

4-85

DTIC