

# BASIC SIG

April 1984

'Twas the night before startup,  
And all through the shop  
Not a program was working,  
Not even a lookup.

The coders huns by their VT100s in despair,  
With hopes that a miracle soon would be there.

The users were nestled all snus in their beds,  
While visions of reports danced in their heads.

When out in the coffee room there arose such a clatter,  
I sprang from my cubicle to see what was the matter.



And what to my wonderins eyes should showup,  
But a super coder, in his hand a DECUS coffee cup,  
His resume' showed he'd been hacking for seasons,  
He turned out clean code that used the latest version.  
More rapid than eagles, the programs they came,  
With whistles and bells and 6 letter discriptive names:

RUN RECADD	RUN INQUIRY	RUN UPDATE	RUN DELETE
RUN MTHEND	RUN YEREND	RUN BATJOB	RUN COMPLT

His eyes were glassy, his body pale and lean,  
From nights and weekends in front of the screen.

A wink of his eye, and a twist of his head,  
Soon save me to know I had nothings to dread.

He spoke not a word, but went straight to his work,  
Turnins specs into code, then turned with a Jerk,

And layins his finger on the <return> key,  
The system come up and ran perfectly.

UPDATE updated, and DELETE, it deleted,  
And when he ran COMPLT, the whole thins completed.

He tested each whistle, he tested each bell  
Not once usind ON ERROR GOTO, the whole thins ran swell.



The testins was finished, the system concluded,  
The user's last changes were even included.

He picked up his check, and took his DECUS coffee cup,  
And when off to work for a friend at a start-up.

We signed off the system, and turned it all in,  
and waited for the comments and praise to besin.



But the user replied with new requests and the taunt,  
'It's exactlyly what I asked for, but not what I want.'



The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	PDT
DECnet	Digital Logo	RSTS
DECsystem-10	EduSystem	RSX
DECSYSTEM-20	IAS	UNIBUS
DECUS	MASSBUS	VAX
DECwriter	PDP	VMS
		VT

UNIX is a trademark of Bell Laboratories.

Copyright © Digital Equipment Corporation 1984  
All Rights Reserved

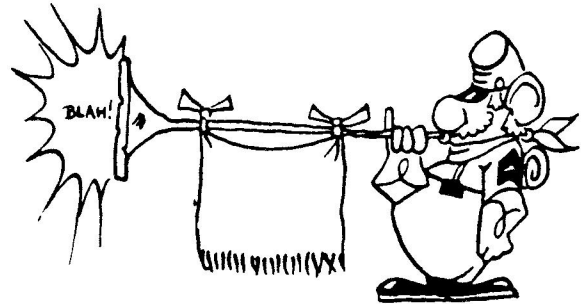
It is assumed that all articles submitted to the editor of this newsletter are with the authors' permission to publish in any DECUS publication. The articles are the responsibility of the authors and, therefore, DECUS, Digital Equipment Corporation, and the editor assume no responsibility or liability for articles or information appearing in the document. The views herein expressed are those of the authors and do not necessarily express the views of DECUS or Digital Equipment Corporation.

BASIC SIG Steering Committee



Chairman

Daniel Esbensen  
Touch Technologies, Inc.  
609 S. Escondido Blvd.  
Suite 101  
Escondido, CA 92025  
619/743-0494



Newsletter Editor

Ted Bear  
2185 Cox Road  
Aptos, CA 95003  
408/245-7990 ext. 578



Handout Coordinator

Bob van Keuren  
NCCS  
2235 Meyers Avenue  
Escondido, CA 92025  
619/745-6006

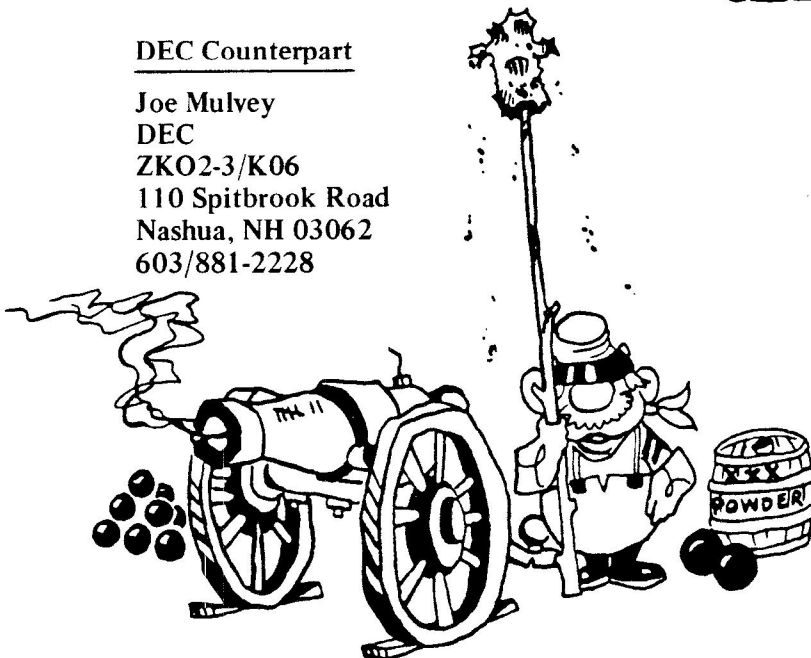


Symposium Coordinator

Ray Strackbein  
ICS  
180 Luring Drive  
Palm Springs, CA 92262  
714/320-8007

DEC Counterpart

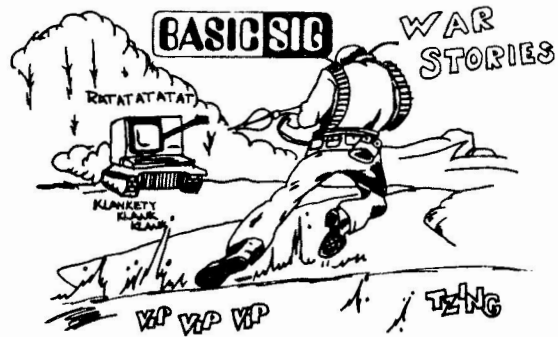
Joe Mulvey  
DEC  
ZKO2-3/K06  
110 Spitbrook Road  
Nashua, NH 03062  
603/881-2228



Wish List

Bill Tabor  
Computer Products, Inc.  
1400 NW 70th Street  
Ft. Lauderdale, FL 22209  
305/974-5500 ext. 269

# BASIC Wars



I got the January '84 issue of the BASIC SIG newsletter today. So you want war stories, do you? Well, here's one from way back in the trenches of time, at least so far as DEC's software goes.

For all who worship at the alter of User-Friendly, particularly from the pew of BASIC-11, version 1, (yeah, yeah, we're ultra-conservative; we also don't have a super budget) BEWARE. In the continual quest of user input errors in control strings it was observed that some people had trouble remembering that 'Y<CR>' = 'yes', etc. So we began to repeat the prompts.

And then it was discovered that a simple test would suffice. Instead of

```
100 IF C$(<>"Y" THEN (XXX)
```

We tried

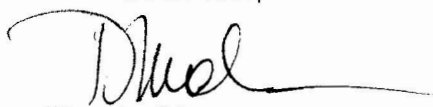
```
100 IF SEG$(C$,1,1)<>"Y" THEN (XXX)
```

Works fine, too. Until a non-literal minded user (programmer, actually) just typed a <CR> for a negative response. OOPS! The program promptly went out to lunch; trapped to 000004 and back into the monitor. Goodbye, application code!

It seems that BASIC-11 won't take a segment of a null string. One would think that since the string is carried as a null, SEG\$ would return a null, and, of course, the conditional's test for equality would fail. But no, SEG\$ gets itself all wrapped up. So it became a choice, force explicit responses from the users (Y or N, for example) or allow them more freedom. You have to remember that we're not dealing with engineers, but secretaries, book keepers and the like. Not that the engineers are much better. Allowing them to type "Y", "N", "yes", "no", or anything else is easier. For them. And simply testing for a null string before taking the SEG\$ has avoided the problem.

Maybe DEC fixed this little insect in later releases.

Good luck all,

  
Tim Mueller

Energy Technology, Inc.  
1440 Phoenixville Pike  
West Chester, PA 19380  
(215) 647-6810



# Letter From Spain

Since I'm a little bit far from the "Software Free World" (I'm 60 miles west of the Sahara Desert), the only information I get is from the BASIC SIG.

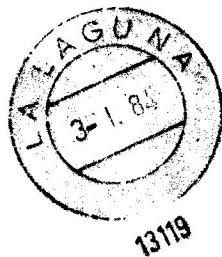
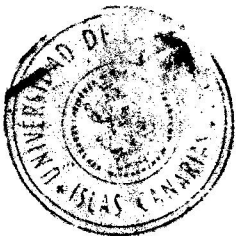
I want to exchange information about COMPUTERS AND LAWS, software will be appreciated. This University would like to establish relations with U.S. universities researching in Laws and computers.

So, please HELP me, send me some BYTES.



  
Joseph A. de la Cuétara.

Comisión de Informática Jurídica  
Facultad de Derecho  
Universidad de la Laguna  
Tenerife, España



# BASIC SIG

Dear Ted,

Attached you will find the response to the SPR submitted by Milwaukee Public Schools concerning their difficulties using GFLOAT and HFLOAT datatypes in VAX BASIC.

A copy of this SPR was printed in your January 1984 issue of the BASIC SIG Newsletter and may be confusing to anyone wishing to use GFLOAT or HFLOAT datatypes in VAX BASIC. It seems appropriate that the response to this SPR be published in the next issue of the BASIC SIG Newsletter so that anyone interested in using GFLOAT or HFLOAT datatypes is provided with the correct information.

Note that the response refers to the KU750 option, which applies to the VAX-11/750 that this customer was running. The name of the option on other systems may be different, but the same concept applies.

Tom Lavigne



D I G I T A L  
SPR ANSWER FORM

SPR NO. 11-63458

	SYSTEM	VERSION	PRODUCT	VERSION	COMPONENT
SOFTWARE:	VAX/VMS	3.4	Basic	2.2	

The response received for your Software Performance Report is as follows. Please contact your local office for further assistance.

**PROBLEM:**

Any program which uses real values and which is compiled with the GFLOAT and HFLOAT qualifiers will fail.

**SOLUTION:**

Thank you for your SPR. The error you observe when attempting to use GFLOAT and HFLOAT operations indicates that your system does not have G & H microcode in operation.

In order to use GFLOAT and HFLOAT variables or constants in any language you must have G & H microcode in your system or link your program with LIB\$ESTEMU from SYS\$LIBRARY:STARLET.OLB. For example:

LINK program,SYS\$LIBRARY:STARLET/INCLUDE=LIB\$ESTEMU

If you link your program with LIB\$ESTEMU, the VAX/VMS run-time library will trap any OPCDEC faults generated by use of GFLOAT/HFLOAT instructions and then emulate the instructions in software. Please refer to page 6-33 of the VAX-11 Run Time Library User's Guide for more information on LIB\$ESTEMU.

If your system has G & H microcode support installed, no OPCDEC faults should be generated. The occurrence of OPCDEC faults can indicate either that you have not chosen to install the KU750 microcode option or that you are not loading the microcode when you boot the system.

# In Defense of BASIC-11

By Arthur H. Stroud

Department of Mathematics  
Texas A&M University  
College Station, Texas 77843

## 1. Introduction

It seems that most articles about BASIC are about data processing applications. Here I will say a few words about a scientific application. In particular I want to point out some features of BASIC-11 which make it well suited to this application and which I would hate to see omitted in future "improvements" of the language.

## 2. A Scientific Application

I am interested in algorithms for numerical analysis calculations, in particular algorithms for the numerical evaluation of definite integrals. There are many different types of integrals to be considered. There are single integrals, double integrals, triple integrals. There are integrals over bounded regions, over unbounded regions, over the entire space. There are integrals with special types of weight functions, integrals with rapidly oscillating integrands, integrals of numerical data, and so on. There are literally hundreds of useful algorithms which are known. (A standard reference is: P. J. Davis and P. Rabinowitz, Numerical Integration, Academic Press, 1975.)

I am developing an interactive system of such algorithms. Without going into a description of this system, the main features of it will be:

1. It will contain a brief description of each algorithm.
2. Each algorithm can be called upon to evaluate an integral provided by the user.
3. To evaluate an integral the user will have to provide, among other things, a subroutine that evaluates the function to be integrated (the integrand).

In order to be interactive as much as possible it is desirable that the subroutine that evaluates the integrand should not have to be compiled and linked to the main part of the system. To my knowledge this can only be done at



present with an interpreted language. This is where BASIC-11 comes in.

### 3. BASIC-11 and Enhancements

As pointed out in the article by J. Coleman and D. Nasater, "Interpretive Business Basic with RMS-11K," pp. 76-103 in the August 1983 issue of this Newsletter, the trend seems to be away from interpretive to compiled BASICs. They also point out advantages of interpretive BASIC for business applications.

To get to the point as quickly as possible I will simply state the features that I like about BASIC-11 for my application and the enhancements that I would make to it.

The features most important to me are:

A1 - The OVERLAY statement

A2 - The ability to stop a program, enter a subroutine, and continue without compiling and linking.

Keeping A1 and A2, the enhancements that I would make are:

B1 - Provide FORTRAN-like independent subroutines; mainly the ability to pass arguments to the subroutine, and independent variable names. Independent line numbers are not that important.

B2 - Add to the OVERLAY statement the ability to shift all line numbers in the overlay segment by a given amount. In other words a relocatable overlay.

B3 - Have two identical versions of the same language, an interpreted version and a compiled version, both having features A1, A2, B1, B2. (Now there is a radical suggestion if there ever was one!)

I have never heard anyone mention both a compiled version and an interpreted version of exactly the same language. Having worked on my system for several years, I have reached the conclusion that every language -- BASIC, FORTRAN, Assembly Language, whatever -- should come both ways. Why not? Do we want to beat the Japanese or not?

### 4. Summary

Let's not "improve" BASIC-11 by making it like some other language. Let's show some imagination and really improve it. And in the process let's not forget the scientific user. There's no reason why DEC can't provide the best.

# **BASCAL or PASIC?**

## **A Structured Approach For Programming in VAX BASIC**

Dr. Kuriakose Athappilly  
College of Business  
Western Michigan University

Programming in BASIC had been very unstructured in the past. But that situation has been changed considerably since the introduction of VAX-BASIC. Many of the programming features in VAX-BASIC are so close to those in PASCAL (one of the most structured languages now available) that the VAX-BASIC is now nick-named BASCAL or PASIC. This paper attempts to show the structure features in VAX-BASIC. While doing so, the paper will briefly deal with some of the introductory concepts of structured programming such as the essential features and the basic approaches.

## INTRODUCTION

In the early days of computers, computer memories were limited and more expensive than they are today. As a result, most of the programs were written with an intent of minimal use of memory and maximum efficiency. Experienced programmers, therefore, taxed their skills to use complicated and nonstandard techniques in their programs. Subsequently, the programs were extremely difficult to understand and to use by others. No doubt, "debugging" and "maintenance" became more and more time consuming and expensive, especially because of the increasing salary rates of the programmers. Now, the computer costs have decreased markedly and memory sizes increased remarkably along with computer speed. The emphasis today is, therefore, on good design and clarity which gave birth to the notion of structured programming.

## ESSENTIALS OF A STRUCTURED PROGRAM

Structured programming is a way of designing and writing programs so that the programs may be clearly understood and easily modified by others. The emphasis here is on program clarity for the continued use of it with easy debugging and maintenance tasks. Thus essential features of a structured program are:

1. It is easy to read.
2. It is clearly understood.
3. It is easy to debug.
4. It is easy to modify.

## BASIC APPROACHES TO STRUCTURED PROGRAMMING

Theoretically, any structured process must be based on two fundamental concepts for its effective operation. These two concepts are analysis and synthesis. In the analysis stage, we break down a process into smaller parts until each part can be tackled in the desired manner. In the synthesis, however, we combine the individually tackled parts or units into one single system.

Structured programming, as it came into existence more under the influence of natural expediency than that of a sound theory, did not emphasize on the principles of analysis and synthesis in the beginning. As a practical approach, therefore, structured programming developed with the technique of the TOP-DOWN design.

## TOP-DOWN DESIGN

According to TOP-DOWN design, a program is conceived as a complete unit. This complete unit should represent a systematic development of a set of procedures in a certain sequence. A procedure may be simple or complex. A complex procedure will in turn be broken down into a sequence of simpler procedures until at last these procedures can be written into distinct codes such that each statement in the procedure yields exactly one statement in the program.

The top-down design has been a justifiable approach as long as the program has not been very complex and not very large. But, when large problems are to be solved, dependence on top-down design alone proved to be insufficient. Subsequently, the idea of "divide and conquer" came into existence. The technique used to implement this idea is called modularization or modular design.

## MODULAR DESIGN

According to MODULAR design, a program, because of its complexity, is not conceived as one single unit to start with. Rather, we start outlining the individual modules which consist of functions and subroutines. Once we decide the number of modules and their functions, we code and test these modules individually. Once we find that every module in the program works as desired, we combine them into one large module program.

In principle, these two techniques, top-down design and modular design, are opposite in direction. But, in practice, they are complementary. They are, indeed, the offshoots of the principles of analysis and synthesis. To solve a sufficiently complex problem, therefore, we may start with identifying many modules--modular design. Here occurs the process of analysis. Then, for each module, we may adopt a top-down design. In the end, we bring all modules together to make it a single program--a synthesis. Thus, the two basic approaches to structured programming are analysis and synthesis, and two basic techniques used for the implementation of this approach are the modular design and top-down structure.

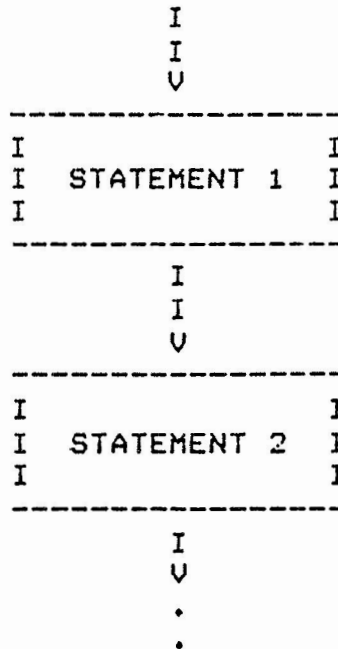
## PROGRAM STRUCTURES

After learning the two basic approaches of structured programming we can now turn our attention to the development of structured programming. A program, as we know, is a sequence of statements. The way these statements are related to one another is known as the "structure" of the program. In structured programming there are basically three program structures. They are:

1. SEQUENCE STRUCTURE
2. DECISION STRUCTURE

## SEQUENCE STRUCTURE

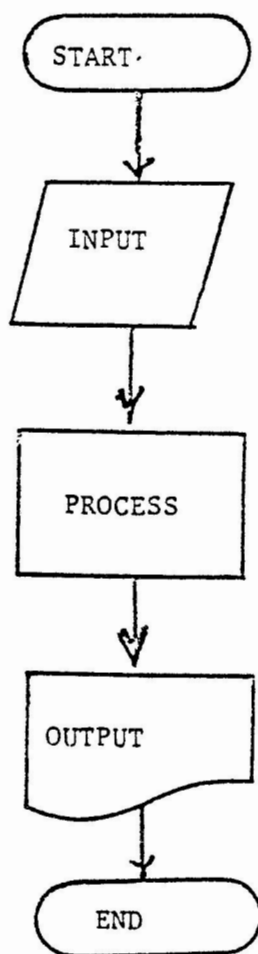
In a sequence structure, each statement is executed in a sequential flow of program logic. In a flowchart form, it may appear as follows:



In this program structure, each procedure occurs sequentially. The procedure may be a single statement, or, as described above, a module. In order to implement this sequential or step by step approach, we must carefully plan the program. We must take extreme care to avoid unnecessary branching and the use of GOTO statements. If we use many GOTO statements to branch back and forth, the program becomes difficult to understand. Often such programs are referred to as "spaghetti programs" as against structured programs. Let us suppose that we have a problem whose programming job consists of only three tasks:

1. To read one record.
2. To calculate the wage.
3. To print out the result.

These tasks are performed with statements which are linked together in a simple sequence structure. These tasks can be flowcharted as follows:



The program to accomplish this task will be as follows:

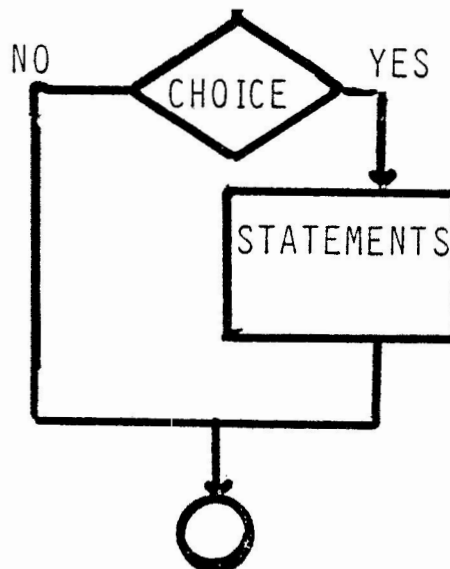
```
*****  
      ILLUSTRATION 1: SEQUENCE STRUCTURE  
*****  
  10  INPUT "ENTER NAME, HOURS, AND RATE";NAME$, &  
      HOUR,RATE  
  20  LET WAGE = HOUR * RATE  
  30  PRINT NAME$,HOUR,RATE;WAGE  
  40  END  
-----  
READY  
RUNNH  
-----  
ENTER NAME, HOURS, AND RATE? DAVID,25,3.7  
DAVID          25          3.7          92.5  
-----
```

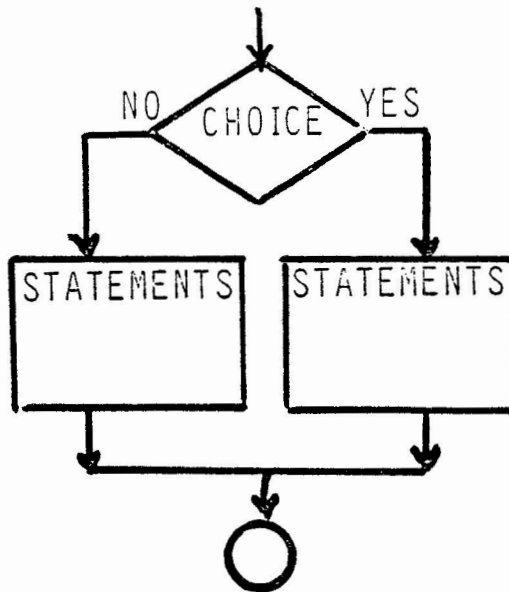
#### DECISION STRUCTURE

Decision structure refers to conditional branching. Conditional branching occurs in two distinct contexts. One is the double decision context and the other is the multiple decision context. The double decision context is handled in two ways using IF-THEN statements and IF-THEN-ELSE statements.

##### 1. DOUBLE DECISION AND IF-THEN STATEMENT

In the double decision structure, a test condition along with two possible resulting paths are presented. One of these two paths will be chosen depending on whether or not a particular condition is met. In flowchart, this situation is shown as follows:





In VAX-BASIC double decision situations can be tackled either by IF-THEN statements or by IF-THEN-ELSE statements.

Suppose in the previous example, we have to decide the wage on overtime and regular time basis. The computer, therefore, must make a check after it reads the record.

The corresponding program will be:

-----  
 ILLUSTRATION 2: IF-THEN SELECTION STRUCTURE  
 -----

```

10 INPUT 'TYPE IN YOUR NAME, HOURS, AND RATE'; &
    NAME$,HOURS,RATE
20 IF HOURS > 40 THEN 50
30 LET WAGE = HOURS * RATE
40 GO TO 60
50 WAGE = (40 * RATE) + (HOURS - 40) *(RATE * 1.5)
60 PRINT NAME$,HOURS,RATE,WAGE
70 END
  
```

-----  
 TYPE IN YOUR NAME, HOURS, AND RATE? DAVID BRENT,40,3  
 DAVID BRENT      40                      3                      120  
 -----



Here the condition is tested in line 20. If the condition is false, we have to use our unconditional branching in line 40 to continue the execution.

## 2. DOUBLE DECISION AND IF-THEN-ELSE STATEMENT

VAX-BASIC provides a convenient feature to avoid the GO TO statement in line 40 of the previous program. This feature is the IF-THEN-ELSE statement as we have seen in the chapter dealing with control statements.

The previous program using IF-THEN-ELSE statements is as follows:

-----  
ILLUSTRATION 3: IF-THEN-ELSE SIMPLE SELECTION STRUCTURE  
-----

```
10 INPUT "TYPE IN YOUR NAME, HOURS AND RATE";  &  
      N$,HOUR,RATE  
20 IF HOUR > 40  
      THEN WAGE = (40 * RATE) + (HOUR-40)*(RATE* 1.5)  
      ELSE  
      WAGE = HOUR * RATE  
30 PRINT N$,HOUR,RATE,WAGE  
40 END
```

-----  
TYPE IN YOUR NAME, HOURS AND RATE? DAVID BRENT,42,3  
DAVID BRENT 42 3 129  
-----

We did not use any GO TO statements in this program. But when we have many tasks to accomplish in either selections a simple IF-THEN-ELSE statement would be inadequate. For example, the previous problem may be elaborated as follows:

Let us assume that there are three tasks to be accomplished in either case. Let these three tasks be:

1. determination of the number of overtime or regular time.
2. calculation of wage.
3. total wage in each category.

In a situation shown above, we may use the IF-THEN-ELSE statement as follows:

-----  
 ILLUSTRATION 4: IF-THEN-ELSE COMPLEX SELECTION STRUCTURE  
 -----

```

10 INPUT "TYPE IN YOUR NAME, HOURS AND RATE";  &
      N$,HOUR,RATE
20 IF HOUR > 40 THEN 50 ELSE 120
30 !-----
40 REM THE "THEN" PART BEGINS
45 !
50 LET WAGE = (40*RATE) + (HOUR-40)*RATE
60 LET OVER_TIME = OVER_TIME + 1
70 LET TOTAL_OVER_WAGE = TOTAL_OVER_WAGE + WAGE
80 GO TO 160
90 REM THE "THEN" PART ENDS
100 !-----
110 REM THE "ELSE" PART BEGINS
115 !
120 LET WAGE = HOUR * RATE
130 LET REG_TIME = REG_TIME + 1
140 LET TOTAL_REG_WAGE = TOTAL_REG_WAGE + WAGE
150 REM THE "ELSE" PART ENDS
160 !-----
  
```

3. MULTIPLE DECISIONS OR CASE STRUCTURE

The IF-THEN-ELSE statement is convenient if there are only two alternatives to test. When there are more than two alternatives to choose from, we enter into a case structure. Two techniques which VAX-BASIC provides for case structure are the ON-GO-TO statements and the NESTED-IF-THEN-ELSE statements.

A. THE ON-GO-TO STATEMENT

Suppose we want to determine the wages according to categories of working hours. Let us assume that the firm determines the rates according to the table given below.

HOURS WORKED	CATEGORY	COMMISSION RATE
LESS THAN 20	PART TIME	\$5.00
20 - LESS THAN 40	REG-TIME	\$6.00
40 - LESS THAN 60	OVER TIME	\$9.00
60 - LESS THAN 80	DOUBLE TIME	\$12.00

The case structure for the above situation can easily be calculated by applying a simple arithmetic expression to the ON-GO-TO expression. The ON-GO-TO statement might be as follows:

```
10 ON (HOURS/20 + 1) GO TO 100,200,300,400
```

Suppose we have Peter, John, Jacob, Tom, Dick and Harry who have worked 15, 35, 50, and 70 hours. A simple program involving case structure with ON-GO-TO statement will be as follows:

-----  
ILLUSTRATION 5: ON-GO-TO CASE STRUCTURE  
-----

```
10 !THIS PROGRAM CALCULATES WAGES ACCORDING TO
20 !FOUR DIFFERENT RATES. IT UTILIZES CASE STRUCTURE.
30 !THE TECHNIQUE IS ON-GO-TO
40 !-----
50 PRINT "NAME","HOURS","CATEGORY","WAGE"
60 PRINT "-----","-----","-----","-----"
70 READ NAME$,HOUR
80 IF NAME$ = STOP THEN 440
90 ON (HOUR/40 + 1) GO TO 100,200,300,400
95 !
100 REM CASE1: HOURS < 20
105 !
110 WAGE = HOUR * 5%
115 RATE$ = "PART TIME"
120 GO TO 420
125 !
200 REM CASE2: HOURS - 20 - LESS THAN 40
210 WAGE = HOUR * 6%
215 RATE$ = "REGULAR"
220 GO TO 420
225 !
300 REM CASE3: HOURS 40 TO < 60
305 RATE$ = "OVER TIME"
310 WAGE = HOUR * 9%
320 GO TO 420
325 !
400 REM CASE4: HOURS 60 TO < 80
410 WAGE = HOUR * 12%
415 RATE$ = "DOUBLE TIME"
420 PRINT NAME$,HOUR,RATE$,WAGE
430 GO TO 70
440 !
450 DATA "PETER",15,"JOHN",35,"JACOB",50
460 DATA "TOM",20,"DICK",70,"HARRY",55,"STOP",0
470 END
```

-----  
READY  
RUNNH  
-----

NAME	HOURS	CATEGORY	WAGE
PETER	15	PART TIME	75
JOHN	35	REGULAR	210
JACOB	50	OVER TIME	450
TOM	20	REGULAR	120
DICK	55	OVER TIME	495
HARRY	70	DOUBLE TIME	840

-----

It must be noted, that the ON-GO-TO statement necessitates the use of GO TO statements. Yet the ON-GO-TO statement is an efficient tool for structured programming because of its ability to classify into specific categories.

#### A. NESTED IF-THEN-ELSE STATEMENT

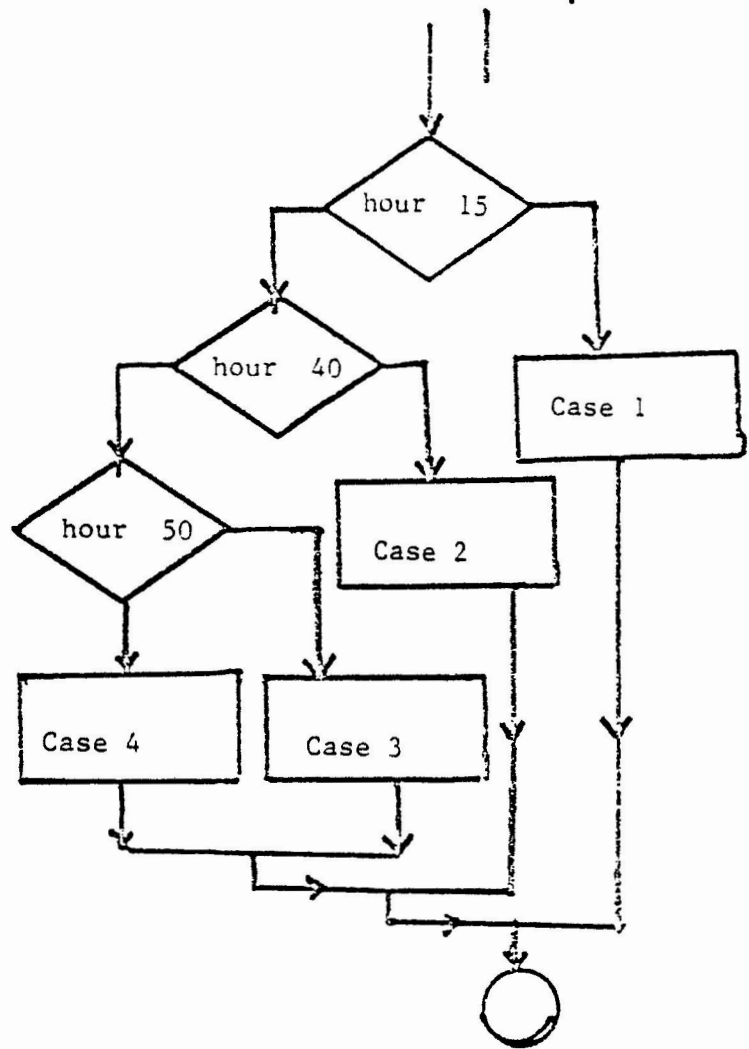
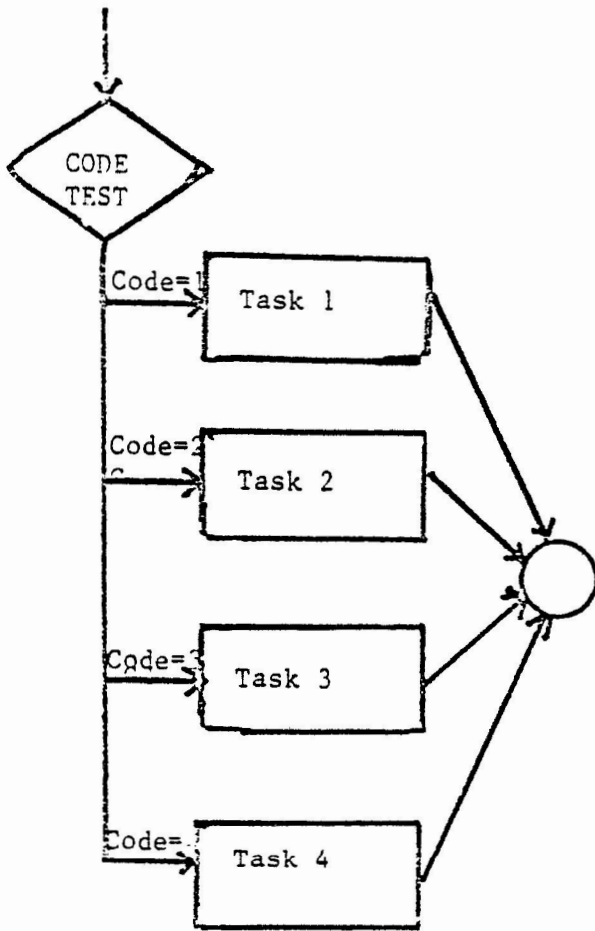
The ON-GO-TO statement is convenient to use as long as there is a relationship between the case number of the expression in the ON-GO-TO statement and the breakdown of the data to analyze. If the relationship does not exist, we have to use NESTED IF-THEN-ELSE statements to handle case structure. Let us try to find wages for a different set of rates as described below.

HOURS WORKED	CATEGORY	RATE
LESS THAN 15	PART TIME	\$5.00
15 - LESS THAN 40	REGULAR	\$6.00
40 - LESS THAN 50	OVER TIME	\$9.00
50 - LESS THAN 70	DOUBLE TIME	\$12.00

-----

The NESTED IF-THEN ELSE statements in this case may be represented as follows:

Flowcharts



The program on commissions can be rewritten with NESTED IF-THEN-ELSE statements with one line change. The line number 90 might be replaced as follows:

-----  
 ILLUSTRATION 6: NESTED IF-THEN-ELSE CASE STRUCTURE  
 -----

```

10 !THIS PROGRAM CALCULATES WAGES ACCORDING TO
20 !FOUR DIFFERENT RATES. IT UTILIZES CASE STRUCTURE.
30 !THE TECHNIQUE IS ON-GO-TO
40 !-----
50 PRINT "NAME", "HOURS", "CATEGORY", "WAGES"
60 PRINT "-----", "-----", "-----", "-----"
70 READ NAME$, HOUR
80 IF NAME$ = STOP THEN 440
90 IF HOUR < 15                                &
    THEN 100                                  &
    ELSE IF HOUR < 40                          &
        THEN 200                              &
        ELSE IF HOUR < 50                      &
            THEN 300                          &
            ELSE 400
100 REM CASE1: HOURS < 15
105 !
110 WAGE = HOUR * 5%
115 RATE$ = "PART TIME"
120 GO TO 420
125 !
200 REM CASE2: HOURS - 20 TO < 40
210 WAGE = HOUR * 6%
215 RATE$ = "REGULAR"
220 GO TO 420
225 !
300 REM CASE3: HOURS 40 TO < 50
305 RATE$ = "OVER TIME"
310 WAGE = HOUR * 9%
320 GO TO 420
325 !
400 REM CASE4: HOURS 50 TO < 70
410 WAGE = HOUR * 12%
415 RATE$ = "DOUBLE TIME"
420 PRINT NAME$, HOUR, RATE$, WAGE
430 GO TO 70
440 !
450 DATA "PETER", 15, "JOHN", 35, "JACOB", 50
460 DATA "TOM", 20, "DICK", 45, "HARRY", 55, "STOP", 0
470 END
  
```

READY  
 RUNNH

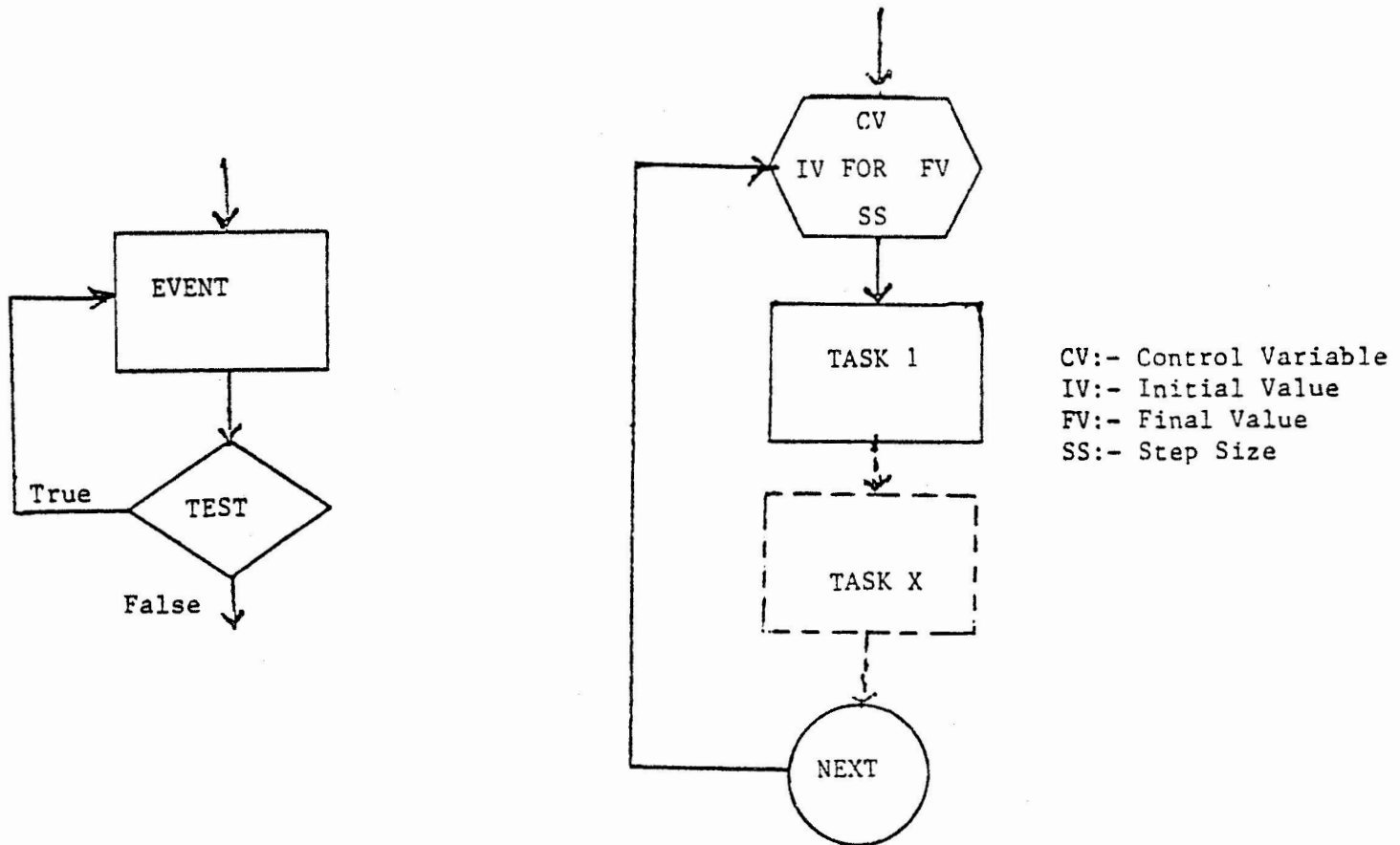
```

-----
NAME           HOURS           CATEGORY           WAGE
-----
PETER          15           PART TIME          75
JOHN           35           REGULAR            210
JACOB          50           OVER TIME          450
TOM            20           REGULAR            120
DICK           55           OVER TIME          495
HARRY          70           DOUBLE TIME        840
-----
  
```

Care must be taken in a case structure for proper indentation and the continuation symbol (&) required for the multiline statement.

### LOOP STRUCTURE

The loop structure is also known as the ITERATION STRUCTURE. In flowchart one can depict a loop structure as follows:



In VAX-BASIC we have learned two ways of constructing loop structures:

1. Use of counter, and test for exit.
2. Use of FOR-NEXT statements.

For example, consider the following two program statements:

-----  
 ILLUSTRATION 7: LOOP STRUCTURE WITHOUT FOR-NEXT  
 -----

```

10 LET C = 0
20 LET C = C + 1
30 IF C > 20 THEN 60
40 PRINT C, C^2
50 GO TO 20
60 END
  
```

-----

-----  
 ILLUSTRATION 8: LOOP STRUCTURE WITH FOR-NEXT  
 -----

```

10 FOR C = 1 TO 20
20 PRINT C, C^2
30 NEXT C
40 END
  
```

-----

It may be noted that the segment 2 is more easy to understand, and has less number of statements than segment 1. In segment 1, we need to initialize the counter, make a test for exit with the IF-THEN statement, and often we may have to use GO TO statements too. Thus, counters and tests tend to clutter a program and in a way obscure the purpose of the loop. Hence, it is a good practice to use FOR-NEXT statements for loop-structure whenever possible.

In some cases, knowing the exact number of loopings will be difficult or impractical. In such contexts, modified versions of FOR-NEXT provided by VAX-BASIC may be used. These modifiers for loop-structure are FOR-UNTIL and FOR-WHILE as discussed in the chapter on CONTROL STATEMENTS.

FOR/NEXT	I	UNTIL/NEXT
10 FOR X% = 1% TO 5 %	I 5	X% = 1
20 PRINT X%;	I 10	UNTIL X% = 5%
30 NEXT X%;	I 20	PRINT X%
40 END	I 30	X% = X% + 1%
	I 40	NEXT
	I 50	END
READY	I	READY
RUNNH	I	RUNNH
1 2 3 4 5		1 2 3 4

-----



## CASE STRUCTURE USING SELECT STATEMENT

The SELECT statement is perhaps more useful than any of the previous techniques discussed earlier in multiple decision contexts. The general format of the SELECT statement is as follows:

```

SELECT                               Select-expression

      CASE                            Case-values
      :
      :
      STATEMENTS
      :
      :
      CASE                            Case-values
      :
      :
      STATEMENTS
      :
      :
      CASE                            ELSE
      :
      :
      STATEMENTS
      :
      :

END SELECT

```

The SELECT expression can be numeric or string. For Example, if we desire to categorize people according to age groups in many irregular combinations, SELECT statement can be conveniently used as follows:

```

*****
                ILLUSTRATION:                SELECT/CASE
*****

```

```

SELECT                               AGE_OF_PERSON

      CASE                            15
      PRINT "YOU ARE TOO YOUNG TO GET MARRIED"
      PRINT "PLEASE BE PATIENT"
      CASE                            16 TO 20
      PRINT "YOU ARE IN RIGHT AGE TO MAKE EXPLORATIONS"
      CASE                            21 TO 30
      PRINT "THIS IS THE MOST OPPORTUNE TIME"
      CASE                            31 TO 55
      PRINT "TIME FOR SECONDARY AND TERTIARY EXPERIMENTS"
      CASE                            ELSE
      PRINT "YOU ARE GETTING OLD, BETTER TO GIVE UP"

END SELECT

```

```

*****

```

The modifiers are key words that qualify a statement. By qualifying a statement, we are able to execute a statement conditionally or to create an implied loop. The IF and UNLESS modifiers enable us to list a conditional expression and FOR, UNTIL, WHILE modifiers enable us to create loops.

For instance,

1. 10 PRINT "YES" IF K = 0 is equivalent to 10 IF K = 0 THEN PRINT "YES"
  
2. 10 PRINT "\*" FOR X% = 1% TO 5% """" 10 FOR X% = 1% TO 5%  
20 PRINT "\*" 30 NEXT X%
  
3. 10 PRINT "\*" UNLESS K = 0 """" 10 IF K = 0 THEN PRINT "\*"

In short, gone are the days of BASIC spaghetti programs with innumerable GO TO statements going up and down. With the use of the selection structures and loop structures mentioned in the paper, we can develop reasonably sound structured programs in VAX-BASIC. We would not be far from truth when we say that the revised VAX-BASIC versions are as good as PASCAL to implement structured programs.

## IMPLEMENTATION OF STRUCTURED PROGRAMMING

For the sake of simplicity, the program we choose is very small and less complicated. The idea is to illustrate most of the concepts and techniques discussed in this chapter in the development of a structured program.

The development of a structured program can be viewed as undergoing what is sometimes known as the cycle of birth, death, and resurrection. The birth of the structured program takes place through the process of the input, output specifications of the problem--the STEP 1. This is the FIRST LOOK at structured programming. At this stage, we do not have a clear perception of its parts. We simply see the problem as a whole, and as such it can not be tackled. Hence, we want to have a closer look at it by dividing it into well defined parts. Thus, in structured programming, the gradual death process occurs through progressive and systematic breakdowns of the problem. This breakdown begins with an ANALYTIC VIEW of the problem--the STEP 2. Here we examine the complexity of the problem and we try to adopt the "divide and conquer" principle. We delineate the major tasks involved in the problem. Once we delineate the tasks, we introduce the technique of modularity, namely, we assign each task to functions or subroutines. This marks the first-level breakdown, the SUBROUTINE BREAKDOWN--the STEP 3. Once each module has been defined, it is easy to introduce the TOP-DOWN DESIGN to each module. In this stage, usually there is a general partitioning of each unit into three major units. These three units in each module can usually be identified as Preparation, Process, and Conclusion.

The preparatory unit introduces into the specified task. The process unit does the necessary calculations and computations. In the conclusion unit, the task is wrapped-up. This process may be called the UNITS BREAKDOWN--the STEP 4.

Each of this unit is further broken down into procedures--the STEP 5. This process can be called the PROCEDURE BREAKDOWN. At this stage, care must be taken to choose appropriate program structures such as sequence, selection, or looping. In STEP 6, the procedures are further broken down into subprocedures if necessary. These procedures or subprocedures, in turn, are broken down into activities. This may be called the ACTIVITY BREAKDOWN. The death processes ends with it.

Thus, the activities are translated into the particular codes. Obviously, in BASIC, these activities are translated into BASIC statements. After this process, we make sure that each of the modules works as desired through testing and debugging. Comments, documentation, and indentations are also inserted as deemed appropriate. These are the cosmetic processes for the funeral. Finally, we combine each of these finished modules together and make it one single program. This is the SYNTHESIS. This synthesis brings about resurrection--structured program. This is STEP 7.

## STEPS IN STRUCTURED PROGRAMMING

STEP -----	EVENT -----	DESCRIPTION -----	DIAGRAM -----
1	FIRST-LOOK	A whole view of the problem without knowing what the parts are.	
2	ANALYSIS	Examine what the major tasks are.	
3	SUBROUTINES BREAKDOWN	Assign each task into each module (modularization).	
4	UNIT BREAKDOWN	Each module is broken down into major units. (top down design begins)	
5	PROCEDURE BREAKDOWN	Each unit is broken down into major procedures or sub-procedures. (selection of program structures).	
6	ACTIVITY BREAKDOWN	Each procedure or subprocedure is broken down into activities translatable to language codes.	
7	SYNTHESIS	Combining all modules together. (Appropriate program structures, comments, documentation, indentation, and remarks are necessary).	

## ILLUSTRATION OF A STRUCTURED PROGRAM

### STEP 1: FIRST LOOK

The problem is to generate a multiple choice quiz program which will allow the user to answer the questions and will give out the result of the quiz.

### STEP 2: ANALYSIS

Obviously, the program must contain the set of multiple choice questions, it must receive the answers as input from the user, it must examine its rectitude and validity if necessary, it must assess the number of right and wrong answers and finally, it should print out the result. It is also desirable to explain to the user the nature and purpose of the program in the beginning.

### STEP 3: FIRST-LEVEL BREAKDOWN: SUBROUTINES

In this stage we assign the major tasks delineated in the analysis stage into different modules in the proper sequence. Thus, we might arrive at:

MODULE 1: Subroutine explaining the nature and purpose of the program.

MODULE 2: Subroutine to present the current question.

MODULE 3: Subroutine to answer the current question, to make a validity check.

MODULE 4: Subroutine to verify the answer.

MODULE 5: Subroutine to print out the results.

### STEP 4, STEP 5, AND STEP 6 (UNIT PROCEDURE AND ACTIVITY BREAKDOWNS)

Step 4, Step 5, and Step 6 are combined in one table shown below. After the modules dealing with different levels of breakdown and coding are well defined, each module is tackled individually.

MODULE 1  
-----

	STEP 4 PROCEDURE -----	STEP 5 ACTIVITY -----	STEP 6 BASIC STATEMENT -----
PREPARATION UNIT	Select a subroutine	Call a subroutine	100 GOSUB 1000
PROCESS UNIT	Explain purpose and nature	printout purpose	100 PRINT "PURPOSE"
		printout nature.	100 PRINT "NATURE"
CONCLUSION UNIT	End of sub- routine.	Return to main line.	1020 RETURN

MODULE 2  
-----

	PROCEDURE -----	ACTIVITY -----	BASIC STATEMENT -----
PREPARATION UNIT	Select a subroutine	Call a subroutine	120 GOSUB 2000
PROCESS UNIT	1. Present question #1.	Present the ques- tion.	120 PRINT "5X2 IS"
		Present choice 1	210 PRINT, 4
		Present choice 2.	220 PRINT, 6
		Present choice 3.	230 PRINT, 8

Present 240 PRINT, 10  
choice 4.

2. Present	.	.
question	.	.
#2.	.	.
	.	.
	.	.
3. Present	.	.
question	.	.
#3.	.	.
	.	.
	.	.
4. Present	.	.
question	.	.
#4.	.	.
	.	.
	.	.
5. Present	.	.
question	.	.
#5.	.	.
	.	.
	.	.

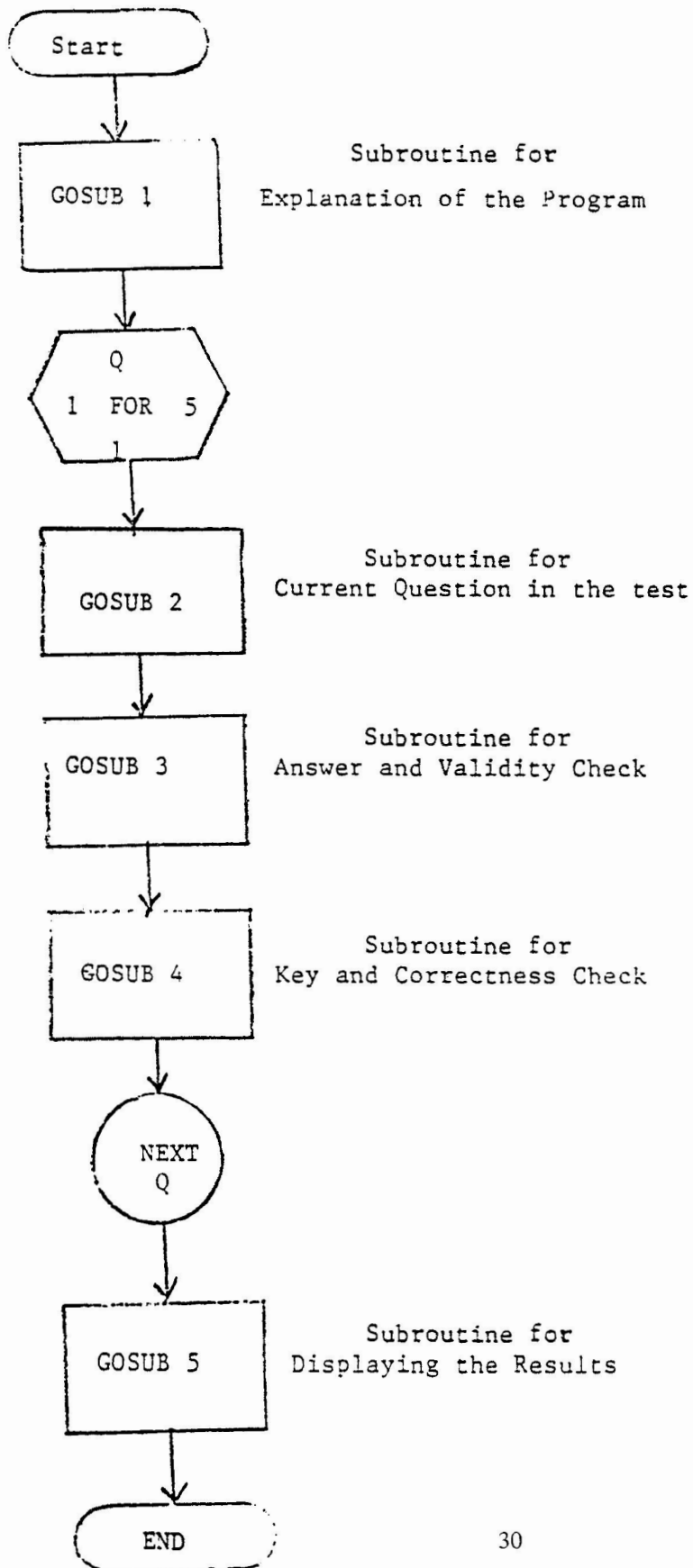
CONCLUSION  
UNIT

End the  
subroutine.

Return  
to main  
line.

999 RETURN

FLOWCHART FOR THE MAIN LINE OF THE PROGRAM





As described above, the rest of the modules, namely, module 3, module 4, and module 5 can be developed in a similar fashion.

After we develop each module, STEP 7: SYNTHESIS, they should be combined into one single program. This constitutes the Main Line of the program. This process marks STEP 7--THE SYNTHESIS. The main line for this program may be as follows:

```
100  GOSUB 100      ! SUBROUTINE FOR EXPLANATION
110  FOR Q = 1 TO 5 ! Q = QUESTION
120    GOSUB 2000   ! SUBROUTINE FOR CURRENT
                ! QUESTION.
130    GOSUB 3000   ! SUBROUTINE FOR ANSWER AND
                ! VALIDITY CHECK.
140    GOSUB 4000   ! SUBROUTINE FOR KEY AND
                ! CORRECTNESS.
150  NEXT Q
160  GOSUB 5000    ! SUBROUTINE FOR RESULTS.
170  STOP
```

The complete program is given below.

-----  
 QUIZ PROGRAM  
 -----

```

10  ! THE PURPOSE OF THIS PROGRAM IS GO GENERATE A
20  ! FIVE QUESTION MULTIPLE CHOICE QUIZ WHICH WILL
30  ! TELL THE USER IF THE QUESTION IS ANSWERED
40  ! CORRECTLY AND WILL ALSO GIVE THE NUMBER RIGHT
50  ! AT THE END OF THE QUIZ.
60  !
70  GOSUB 200          !SUBROUTINE FOR EXPLANATION
80  FOR Q = 1 TO 5  !Q = QUESTION
90      GOSUB 300      !SUBROUTINE FOR CURRENT QUESTION
100     GOSUB 870      !SUBROUTINE FOR ANSWER AND VALIDY CHECK
110     GOSUB 990      !SUBROUTINE FOR KEY AND CORRECTNESS
120 NEXT Q
130 GOSUB 1070        !SUBROUTINE FOR RESULTS.
140 !-----
150 REM - SUBROUTINE FOR EXPLANATION
160 !-----
200 PRINT "*****"
210 PRINT
220 PRINT "THE FOLLOWING IS A BASIC MATH QUIZ.
230 PRINT "ANSWER EACH QUESTION WITH THE LETTER OF"
240 PRINT "CHOICE YOU FEEL ANSWERS THE QUESTION"
250 RETURN
260 !-----
270 REM - SUBROUTINE FOR CURRENT QUESTION.
280 !-----
300 ON Q GO TO 310,420,530,630,740
310 PRINT
320 PRINT
330 PRINT "QUESTION #1"
340 PRINT "WHICH IS THE ANSWER TO THE FOLLOWING FOR X?"
350 PRINT
355 PRINT "X = (2*3) + ((5-1)*2)"
360 PRINT
370 PRINT ",A) 36.6"
380 PRINT ",B) 9"
390 PRINT ",C) 14"
400 PRINT ",D) -14"
410 RETURN
420 PRINT
430 PRINT
440 PRINT "QUESTION #2"
450 PRINT "WHICH OF THE FOLLOWING IS THE CORRECT"
460 PRINT "SOLUTION FOR Y IN THE EQUATION BELOW?"
465 PRINT
470 PRINT "Y = (((3*2)-1)-2) + 1"
475 PRINT
480 PRINT ",A) 4"

```

```

490 PRINT , "B) -4"
500 PRINT , "C) 3"
510 PRINT , "D) 26"
520 RETURN
530 PRINT
540 PRINT
550 PRINT "QUESTION #3"
560 PRINT "WHICH OF THE FOLLOWING IS THE CORRECT SOLUTION"
570 PRINT "FOR Z IN THE EQUATION BELOW?"
580 PRINT
585 PRINT "Z = (((((3*2)*(3-1)/2)-1)/1)"
590 PRINT
595 PRINT , "A) 0"
600 PRINT , "B) 5"
605 PRINT , "C) 4"
610 PRINT , "D) 2"
620 RETURN
630 PRINT
640 PRINT
650 PRINT "QUESTION #4"
660 PRINT "WHICH OF THE BELOW IS NOT A PROPER"
670 PRINT "MATHEMATICAL EXPRESSION IN VAX BASIC?"
680 PRINT
690 PRINT , "A) (A*B-4*X+Y)*1-3/4*(1)"
700 PRINT , "B) 1*2*3*4*5*6/1*1-1"
710 PRINT , "C) 222/1+0"
720 PRINT , "D) 3*(5/1(2*3))"
730 RETURN
740 PRINT
750 PRINT
760 PRINT "QUESTION #5"
770 PRINT "WHICH OF THE BELOW IS A CORRECT VERSION"
780 PRINT "OF THE QUADRATIC FORMULA?"
790 PRINT
800 PRINT , "A) (-B + SQRT(B**2-4*A*C))/(2*A)"
810 PRINT , "B) B-4*A*C"
820 PRINT , "C) B**2-4*A/2*A"
830 PRINT , "D) SQRT(B**2-4*A*C)"
840 RETURN
850 !-----
860 REM - SUBROUTINE FOR VALIDITY CHECK AND ANSWER
865 !-----
870 PRINT
880 PRINT "WHAT IS YOUR CHOICE";
890 INPUT A$
900 IF A$ = "A" THEN 960
910 IF A$ = "B" THEN 960
920 IF A$ = "C" THEN 960
930 IF A$ = "D" THEN 960
940 PRINT "INVALID RESPONSE, PLEASE RETYPE ENTRY."
950 GO TO 880

```

```

960 RETURN
970 !-----
980 REM - SUBROUTINE FOR KEY AND CORRECTNESS
985 !-----
990 READ K$:
1000 IF A$ = K$ THEN 1025
1010 PRINT
1020 PRINT "INCORRECT, ";K$" WAS THE CORRECT ANSWER."
1022 GO TO 1040
1025 PRINT
1030 PRINT "CORRECT!! ";K$;" IS THE CORRECT ANSWER."
1035 LET C = C+1 !C=NUMBER OF QUESTIONS CORRECT
1040 RETURN
1050 !-----
1055 REM - SUBROUTINE FOR RESULTS
1065 !-----
1070 LET P = C/.05 !P=PERCENTAGE CORRECT
1080 PRINT
1090 PRINT "THAT IS THE END OF OUR FIVE QUESTION QUIZ"
1100 PRINT "YOU HAD ";C;" OUT OF FIVE QUESTIONS CORRECT."
1105 PRINT "THAT IS ";P;"%."
1110 PRINT
1115 IF C = 5 THEN 1160
1120 IF C = 4 THEN 1170
1130 IF C = 3 THEN 1180
1140 IF C < 3 THEN 1190
1150 PRINT
1160 PRINT "GREAT JOB, YOU GOT THEM ALL CORRECT!!"
1165 GO TO 1200
1170 PRINT "GOOD JOB, YOU ALMOST GOT THEM ALL!"
1175 GO TO 1200
1180 PRINT "FAIR JOB, THAT IS ABOUT AVERAGE."
1185 GO TO 1200
1190 PRINT "YOU DID NOT DO VERY WELL."
1200 RETURN
1210 !
1215 DATA "C","A","B","D","A"
1220 END
-----

```

# U.S. CHAPTER DECUS Program Library SOFTWARE ABSTRACTS

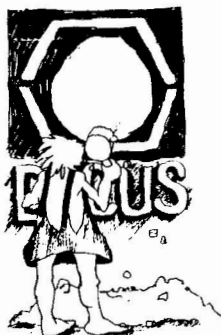
## DECUS PROCEEDINGS

For your convenience and information listed below are the current DECUS Proceedings that are available and can be ordered through the DECUS office in Marlboro, Massachusetts. As availability changes this list will be updated.

			DECUS Part No.	Media Service Codes
Europe	1980	Amsterdam, Holland	PRO-81/V07.1	YA
U.S. Fall	1980	San Diego, California	PRO-81/V07.2	YA
Canada	1981	Montreal, Quebec	PRO-81/V07.3	YA
U.S. Spring	1981	Miami, Florida	PRO-81/V07.4	YA
Australia	1981	Brisbane, Australia	PRO-81/V07.5	YA
Europe	1981	Hamburg, Germany*	PRO-82/V08.1	YA
U.S. Fall	1981	Los Angeles, California	PRO-82/V08.2	YA
Canada	1982	Toronto, Canada	PRO-82/V08.3	YA
U.S. Spring	1982	Atlanta, Georgia	PRO-82/V08.4	YA
Europe	1982	Warwick, United Kingdom	PRO-EUR-82	YA
U.S. Fall	1982	Anaheim, California	PRO-ANA-82	YA
U.S. Spring	1983	St. Louis, Missouri	PRO-STLO-83	YA

\* Available from Geneva only. None available until further notice.

PLEASE NOTE: The DECUS Proceedings are no longer grouped together in one volume; they are each listed separately. European, Canadian and Australian Proceedings will be listed by the year, date and place of the symposium. U.S. Proceedings will be listed by the year, season (Spring or Fall) and place of the symposium.



CATALOG

Version: September 1983

Author: Lars Palmer, Ph.D. AB Hassle, Molndal, Sweden

Operating System: VAX/VMS

Source Language: Datatrieve

This is a very special offering. It consists of an index of the DECUS Library program offerings in machine readable format. It has been in use within Europe now for some time and is updated regularly. The aim is to update it in the Library at least twice a year i.e., it will contain material more recent than the current catalog. This version is updated to correspond to the 1983/84 catalog.

The material is in the form of a large file that can be loaded into a datatrieve ISAM file and used with the procedures enclosed. The routines needed to load the files are on the media. If you do not have datatrieve you should note that the magtapes also contain, beside the basal files, printouts from the files sorted by the several different criteria (the datatrieve command files to do this sort are on the media).

Note: This material is produced as a private initiative of the submitter. The Library has no responsibility either for the correctness of the material or for the updating of it.

Changes and Improvements: Updated to reflect 1983/84 catalog.

Documentation on magnetic media.

Media (Service Charge Code): 600' Magtape (MA)

Format: VAX/ANSI (Blocked at 2048)

Keywords: Catalog  
Operating System Index:  
VAX/VMS

February 27, 1984

Symposium Tape from the VAX SIG, Spring 1983, St. Louis

Version: Spring 1983

Author: Various

Submitted By: Joe L. Bingham, Mantech International,  
Alexandria, VA

Operating System: VAX/VMS V3.2

Source Language: APL, VAX-11 BASIC, BLISS-32, C, VAX-11 COBOL,  
DCL, VAX-11 FORTRAN, MACRO-32, PASCAL, TECO

These programs were submitted for the Tapecopy project at the Spring '83 DECUS Symposium. This is a very large tape, over 96,000 blocks plus about 7,500 blocks of general information and indexes into the VAX SIG tapes. It is a potpourri on new and revised programs, command procedures and other interesting (even useful) material. You have to browse through the tape to appreciate it but some of the things available are: programs to facilitate communications between VAXes and other computers, EDT initializer procedures, spelling and grammar checkers, command line editors, tape manipulation routines (Need to copy a UNIX tar tape?), a touch typing tutor, a few thousand blocks of line printer pictures and many routines to make the VAX System Manager's job easier - from monitoring resources to logging off idle users.

No guarantees are made as to the completeness, usability, or quality of the programs on the tape and the material has not been checked or reviewed.

Note: Release notes (User Instructions) are distributed with the tape.

Restrictions: See individual program documentation.

Sources may or may not be included. Documentation may or may not be included on the magnetic media.

Media (Service Charge Code): 2400' Magtapes (PB)

Format: VMS Backup (Blocked at 2048)

Keywords: Symposia Tapes - VMS  
Operating System Index:  
VAX/VMS

December 12, 1983

Symposium Tape from the VAX SIG. Fall 1983, Las Vegas

Version: Fall 1983

Author: Various

Submitted By: Joe L. Bingham, Mantech International,  
Alexandria, VA

Operating System: VAX/VMS V3.X

Source Language: VAX-11 BASIC, BLISS-32, C, DCL, VAX-11 FORTRAN,  
MACRO-32, PASCAL, TECO

Other Software Required: FORTRAN Compiler. However, most FORTRAN  
and all other sources using a compiler include the compiled  
version.

This tape includes material submitted for the TapeCopy project at  
the Fall 1983 (Las Vegas) DECUS symposium. This is a large tape  
with about 73500 blocks of submitted material and 11500 blocks of  
general information and indexes into the VAX SIG tapes. It is a  
potpourri of new and revised programs, command procedures and  
other useful material. This tape contains new releases of several  
of the most asked about software packages which have appeared on  
past tapes (the LBLTOOLS Unix-like overlay to VMS, Denison's  
spelling and grammer checker, the KERMIT and VAXNET communications  
packages, the VPW poor man's all-in-one system, the ICE command  
line editor and several others) and much material appearing for  
the first time (TYPIST for those aspiring to greater keyboard  
skills, back issues of the pageswapper, an enhanced RUNOFF and a  
graphics package, to name a few) and many utilities for general  
use and for the VAX System Manager.

This tape contains the first extensive collection of games since  
the Spring 1979 tape (many old, some new) and a system for  
controlling access to them.

No guarantees are made as to the completeness, usability, or  
quality of the programs on the tape. The material has not been  
checked or reviewed and documentation may or may not be included.

Note: Release notes are distributed with each tape.

Restrictions: Complete sources are not included.

Completed sources are not included. Documentation may or may not  
be on the magnetic media.

Media (Service Charge Code): 2400' Magtapes (PB)

Format: VMS/Backup (Blocked at 7952)

Keywords: Symposia Tapes - VMS  
Operating System Index:  
VAX/VMS

February 6, 1984



**INDENT/BASIC-PLUS-2 Programming Templates**

**Version:** V1.0, October 1983

**Author:** Janet Scherer et.al., North Shore Sanitary District,  
Gurnee, IL

**Operating System:** RSTS/E V7.2

**Source Language:** BASIC-PLUS-2, INDENT

**Memory Required:** 32K

**Other Software Required:** RMS File Support

**Special Hardware Required:** VT52 or VT100 (latter preferable)

This package, which consists of both templates and utility subprograms, may be used to write data entry/edit programs in BASIC-PLUS-2 with INDENT as the screen handler. The templates include subprograms which will add, update, inquire about, or delete a single record from an RMS indexed file; update entire forms; or update an individual field (allowing the programmer to insert additional validation and/or file lookups). These templates MUST be modified to meet the needs of your own application.

Any of the above templates may call one or more standard utility subprograms. Utilities include: one subprogram for each RMS verb (e.g. GET, PUT); one subprogram for each of your common "end-of-screen" prompts; a subprogram for any FINPT call; and a subprogram for any FUPD call.

Templates for the INDENT screen definition, .CMD and .ODL files, and some programmer aids are also included.

**Note:** Probably minor modifications needed for RSTS/E V8.0.

**Restrictions:** Should use COTREES for the overlays.

**Documentation on magnetic media.**

**Media (Service Charge Code):** 600' Magtape (MA)

**Format:** DOS-11

**Keywords:** Tools - Application  
Development, RSTS BASIC  
**Operating System Index:**  
RSTS/E

February 20, 1984

DISPLY Enhancement

Version: V8.01, October 1983

Author: Ben Ethridge

Operating System: RSTS/E V7.2

Source Language: BASIC-PLUS-2

Memory Required: 32K

Other Software Required: Digital Equipment Corporation's DISPLY Program V7.2

This program performs the following functions:

User defined keyboards are sent messages from the DISPLY program if user defined warning levels are exceeded. For example, the user has told the DISPLY program to warn keyboards 40 and 45 of any irregular system static. Also, the user has set the disk space warning level for device "DB0:" to 5000 blocks. If the DISPLY program sees that the free disk space on DB0: has dropped to 4000 blocks it sends a broadcast message to keyboards 40 and 45 giving the date, time, the message: "Disk DB0 is at 4000 Blocks" and a warning bell.

Actions are performed by the DISPLY program if certain warning levels are exceeded or certain conditions are met when the DISPLY program checks the system statistics. For example, the user has told the DISPLY program to hold shutup if account [1,50] is still online when shutup is run. If the DISPLY program sees that shutup is running and [1,50] is online it changes the priority of the shutup job to -128. It further sends all user defined keyboards a message that [1,50] is online and shutup has been suspended.

The user may enter special "@" commands to force the DISPLY program to detach and process the "@" command file. This gives the user the ability to run any program the user desires from the DISPLY program. For example, the user has predefined the "@ut" command to mean "Log into the System Account and Run the Utility Program."

Documentation on magnetic media.

Media (Service Charge Code): 600' magtape (MA)

Format: DOS-11

Keywords: Utility - System  
Management, RSTS - Utilities  
Operating System Index:  
RSTS/E

**RUNOFF for RSX-11 and RSTS/E**

**Version: S1.4, October 1983**

**Author: Charles H. Spalding III, Adept Technology, Inc.,  
Mountain View, CA**

**Operating System: IAS, RSX-11M, RSTS/E**

**Source Language: MACRO-11**

**Memory Required: 10KW to 14KW**

RUNOFF greatly aids the preparation of documents and manuals. Some of the facilities provided by the program are: automatic line fill and right margin justification, hyphenation, section labeling, pagination, positioning of tables and figures, and creation of tables of contents and an index.

This version of RUNOFF is an update and enhancement of an earlier DECUS library version. (It is not, however, derived from DECUS No. 11-530. In particular, this version does NOT run under RT-11, nor does it support the "transparent string" feature of that version.)

This program includes several features for producing documents which are to be copied on both sides of the paper. Other new features include the following: the ability to combine multiple input files; up to three tables of contents can be produced (e.g., Contents, Figures, and Tables; the Contents table can be automatically generated); subentries can be recorded in the index; the index buffer self-expands as required.

**Changes and Improvements:** Many bugs have been fixed and several existing features have been enhanced. The user manual has been extensively updated, including descriptions of all the new features.

**Documentation on magnetic media.**

**Media (Service Charge Code): Write-Up (AA), Manual (EB),  
600' Magtape (MA)**

**Format: DOS-11**

**Keywords: Text Manipulation,  
RUNOFF  
Operating System Index:  
RSX-11/IAS, RSTS/E**

**February 20, 1984**

# Digital Offers Management Seminar

Digital's Educational Services is offering a seminar titled, "Software project management for small to medium sized projects." The three day course will be of interest to anyone responsible for software purchase, coding, usage, implementation, management or maintenance.

Seminar leader John Rakos will be teaching project managers a successful method for designing and implementing software on micro and mini Digital computers. The complete seminar will be based on case studies from the instructor's thirteen years of experience in software project management and the computer training business in projects with Digital, Bell Northern Research Laboratories, and for the Canadian government. Several workshops in designing projects from start to finish will be presented. Mr. Rankos' expertise lies in bringing software project management techniques previously developed for mainframe computers to the world of mini and micro computers, where these techniques are just as necessary.

Seminars are scheduled for San Francisco, Chicago New York City and Washington D.C. For information or to register, contact Educational Services in Bedford, Massachuseetes, at (617) 276-4949.

I know  
what the experts  
with their computers  
have to say.

But just to be on  
the safe side...



# How Do You Feel Today



AGGRESSIVE



AGONIZED



ANXIOUS



APOLOGETIC



ARROGANT



BASHFUL



BLISSFUL



BORED



CAUTIOUS



COLD



CONCENTRATING



CONFIDENT



CURIOUS



DEMURE



DETERMINED



DISAPPOINTED



DISAPPROVING



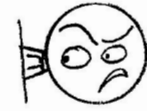
DISBELIEVING



DISGUSTED



DISTASTEFUL



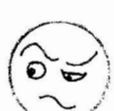
EAVESDROPPING



ECSTATIC



ENRAGED



ENVIOUS



EXASPERATED



EXHAUSTED



FRIGHTENED



FRUSTRATED



GRIEVING



GUILTY



HAPPY



HORRIED



HOT



HUNGOVER



HURT



HYSTERICAL



INDIFFERENT



IDIOTIC



INNOCENT



INTERESTED



JEALOUS



JOYFUL



LOADED



LONELY



LOVESTRUCK



MEDITATIVE



MISCHVOUS



MISERABLE



NEGATIVE



OBSTINATE



OPTIMISTIC



PAINED



PARANOID



PEEPLXED



PROUDISH



PUZZLED



REGRETFUL



RELIEVED



SAD



SATISFIED



SHOCKED



SLEEPISH



SMUG



SURLEY



SURPRISED



SUSPICIOUS



SYMPATHETIC



THOUGHTFUL



UNDECIDED



WITHDRAWN

# 'Twas the Night Before Start Up

'Twas the night before startup  
And all through the shop  
Not a program was working  
Not even a lookup.

The coders hung by their VT100s in despair,  
With hopes that a miracle soon would be there.

The users were nestled all snug in their beds,  
While visions of reports danced in their heads.

When out in the coffee room there arose such a clatter,  
I sprang from my cubicle to see what was the matter.

And what to my wondering eyes should show,  
But a super coder, in his hand a DECUS coffee cup.

His resume' showed he'd been hacking for seasons,  
He turned out clean code that used the latest version.

More rapid than eagles, the programs they came,  
With whistles and bells and 6 letter descriptive names:

RUN RECADD	RUN INQUIRY	RUN UPDATE	RUN DELETE
RUN MTHEND	RUN YEREND	RUN BATJOB	RUN COMPLT

His eyes were glassy, his body pale and lean,  
From nights and weekends in front of the screen.

A wink of his eye, and a twist of his head,  
Soon gave me to know I had nothings to dread.

He spoke not a word, but went straight to his work,  
Turning specs into code, then turned with a jerk,

And laying his finger on the <return> key,  
The system come up and ran perfectly.

UPDATE updated, and DELETE, it deleted,  
And when he ran COMPLT, the whole thing completed.

He tested each whistle, he tested each bell  
Not once using ON ERROR GOTO, the whole thing ran swell.

The testing was finished, the system concluded,  
The user's last changes were even included.

He picked up his check, and took his DECUS coffee cup,  
And when off to work for a friend at a start-up.

We signed off the system, and turned it all in,  
and waited for the comments and praise to begin.

But the user replied with new requests and the taunt,  
"It's exactly what I asked for, but not what I want."

