PROGRAMMING IN INFOBASIC

1.1 Introduction

As you would be aware by now, Globus uses uniVerse/jBase as the bank end to store its data. All programs that make up Globus are written in a language called Infobasic. Infobasic is a very simple yet powerful programming language. With its english like statements, it makes programming very simple. A salient feature of Infobasic is that it does not support data types. All variables in Infobasic are treated as Dynamic Arrays(Refer 2.1 Arrays). Since Infobasic does not support data types, the need to declare variables does not arise.

2.1 Arrays

Before we understand the various commands and the way to write programs in Infobasic, it is very essential to understand the concept of arrays.

Every variable that we use occupies a portion of the memory. Usually character variables occupy 1 byte of memory, which have the capacity to store just one character. Incase a series of characters (string) like 'GLOBUS' has to be stored, then a character variable would not suffice. There comes the need for arrays. We now need 6 bytes of continuous memory blocks in order to store the string. Sequential storage of characters that form a string will make storage and retrieval easier and faster. Moreover all the 6 bytes should have the same name. This is exactly the functionality of an array.

To sum it up, an array is nothing but continuos memory allocation, where in all the bytes have the same name as that of the array and can be distinguished with the help of a subscript which always starts with a '0'.

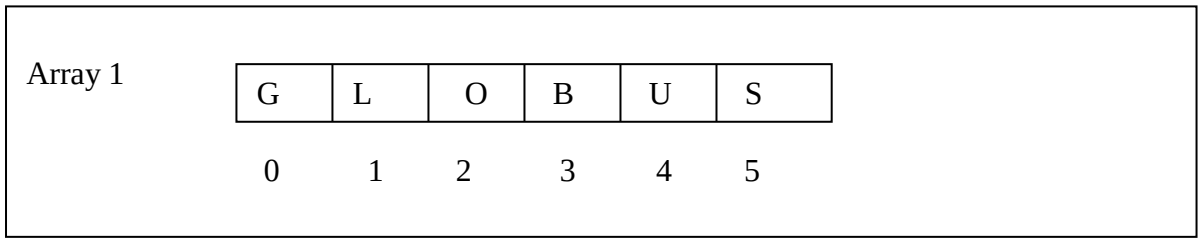

Figure 1.1 Structure Of An Array

Note :
Incase you wish to access 'G' in 'GLOBUS' then specify Array1[0]

2.1.1 Types Of Arrays

There are two different types of arrays that are supported by Infobasic. They are

I. Dynamic Arrays
II. Dimensioned Arrays

I. Dynamic Arrays

Dynamic arrays are, as the name implies, dynamic in both the number, dimensions and their extents. Dynamic arrays are especially useful in providing the ability to manipulate variable length records with a variable length of fields and/or values within fields etc. A dynamic array is just a string of characters that contain one or more delimiter characters. The delimiter characters are :

ASCII Decimal	Description
254	Field Marker
253	Value Marker
252	Sub-Value Marker

Each field is separated by a field marker and a field may contain more than one value separated by a value marker. Any value may have more than one sub-value separated by a sub-value marker.

Filed1**FM**Field2**FM** Value1**VM**Value2**VM**Value3**VM**Value4 **M**Field4**FM**SubValue1**SM**SubValue2**FM**Field5

Figure 2.1 Structure Of A Dynamic Array

Note :
All variables in Infobasic are treated as dynamic arrays. Dynamic arrays do not need any explicit declaration. Initialisation would suffice.
ARRAY = '' → A dynamic array being initialised. Incase the array needs to store a numeric value

I. Dimensioned Arrays

Dimensioned array provide more efficient means of creating and manipulating tables of data elements where the number of dimensions and the extent (number of elements) of each dimension is known and is not likely to change. Dimensioned arrays have to be declared using the DIMENSION statement.

Example:
To declare a dimensioned array use DIMENSION Array2[5,3]
5 - > Refers to the number of rows
3 - > Refers to the number of columns
A customer record is a dimensioned array. All the fields that form the customer record are dynamic arrays.

3.1 Structure Of An Infobasic Program

There are two different types of programs that we can write in Infobasic. One is 'PROGRAM' itself and the other is 'SUBROUTINE'.

Any program that is executed from the uniVerse prompt is termed as a 'PROGRAM' and a program that is executed from within Globus is termed as a subroutine.

*Comments	*Comments
PROGRAM ProgramName	SUBROUTINE SubroutineName
Statement1	Statement1
Statement 2	Statement 2
Statement 3	Statement 3
	RETURN
END	END

Figure 3.1 Structure of a program and subroutine

Usually, any program or subroutine developed by the user is stored under a directory named BP and the core Globus programs or subroutines are stored under GLOBUS.BP. Never store user written programs/subroutines in the GLOBUS.BP directory.

4.1 Compiling And Cataloguing Infobasic Programs And Subroutines

Just like programs written in any programming language need to be compiled, Infobasic programs also need to be compiled. Compilation is the process of converting the code into assembly language that the machine can understand. Once programs/subroutines are compiled, object codes get produced. These object codes get stored in specific directories.

If the source(program/subroutine) is in BP then the object code gets stored in BP.O. If the source is in GLOBUS.BP then the object code gets stored in GLOBUS.BP.O. Apart from compiling Infobasic programs, we also need to catalogue them. As you would be aware by now, VOC is the back bone of our database uniVerse. Cataloguing is the process of making an entry in the VOC. When a program or a subroutine is catalogued, a VOC entry with a type 'V' gets created.

Note :

In jBase VOC entries do not get created for programs and subroutines. In JBase, when a subroutine is compiled, an object code gets created and is stored under the current directory. Cataloguing subroutines in jBase is the process of making the object code a part of a library file under the path specified by the jBase global variable JBCDEV_LIB. This library file under the lib directory gets created automatically inorder to store object codes produced as a result of compiling subroutines. The size and the name of the library files is determined by the configuration file jLibDefinition under the jBase 'config' directory.

Incase of a program, when catalogued, the object code is placed under the directory pointed by the jBase global variable JBCDEV_BIN. Usually this variable points to the 'bin' directory, which is under the 'run' directory. Unlike subroutines, library files do not get created here. The object files get stored straight away under the path specified by JBCDEV_BIN.

5.1 Writing Infobasic Programs

Example 1

Program to display "Hello World"

Step 1

Write a program to display the string "HELLO WORLD" and store it under the BP directory.

Consolidated Solution 1

```
>ED BP HELLO
New record.

----: I
0001= PROGRAM HELLO
0002= CRT "HELLO WORLD"
0003= END
0004=
Bottom at line 3.
----: FI
"HELLO" filed in file "BP".
```

ED is the editor used by Infobasic. Please refer to 'Using ED Editor' notes that has been attached to this course material.

Step 2:

Compile the program

```
BASIC BP HELLO
Compiling:Source='BP/HELLO',
Object = 'BP.O/HELLO'
Compilation Complete.
```

BASIC is the command used by Infobasic to compile programs/subroutines. It converts the code into assembly language and creates the object file.Note that the object code has been placed under the directory BP.O automatically.

Step 3:

Catalogue the program

```
>CATALOG BP HELLO
"*mbdemo.run*HELLO" cataloged.
```

CATALOGUE is the command in Infobasic that makes an entry in the VOC for a compiled program/subroutine.

VOC Entry for the program

```
CT VOC HELLO
HELLO
0001 V
0002 *mbdemo.run*HELLO
0003 B
0004 BN
```

Note :

Instead of using BASIC to compile and CATALOG to catalogue programs/subroutines we can use EB.COMPILE to compile and catalogue programs/subroutines. This command can be used in jBase as well to compile and catalogue programs/subroutines.

Step 4:

Execute the program by typing the following statement at the uniVerse prompt**.**

>RUN HELLO

HELLO WORLD ⟶ Output of the program

6.1 Control Structures In Infobasic

Just like any other programming language, Infobasic also supports a number of control structures namely

I. If Then Else
II. Begin Case End Case
III. For Loop
IV. Open Loop

I. If Then Else

The If clause is used to determine the operations to be run following to be run following either the true or false (successful or unsuccessful) result of the statement. If the statement evaluates to a 'true' then the statements following the THEN clause will get executed. If the statement evaluates to a 'false' then the set of statements following the 'ELSE' clause would get executed. In most cases, either the THEN or the ELSE must be specified; optionally both may be. In certain specific cases the ELSE clause only is available.

For each of these statements the format of the THEN and ELSE clauses is the same. If the THEN or ELSE clause is restricted to one statement, on the same line as the test statement, the THEN or ELSE can be specified in the simple format.

If the THEN or ELSE clause contains more than one statement, or you wish to place it on a separate line, you must use the multiline format which encloses the statements and terminates them with an END.

Example :

```
IF AGE <= 17 THEN
      PRINT "AGE IS LESSER THAN OR EQUAL TO 17"
      PRINT "MINOR"
END
ELSE
      PRINT "MAJOR"
END
```

II. Begin Case End Case

Use the CASE statement to alter the sequence of instruction execution based on the value of one or more expressions. If expression in the first CASE statement is true, the following statements up to the next CASE statement are executed. Execution continues with the statement following the END CASE statement. If the expression in a CASE statement is false, execution continues by testing the expression in the next CASE statement. If it is true, the statements following the CASE statement up to the next CASE or END CASE statement are executed. Execution continues with the statement following the END CASE statement. If more than one CASE statement contains a true expression, only the statements following the first such CASE statement are executed. If no CASE statements are true, none of the statements between the BEGIN CASE and END CASE statements are executed.

Example:

```
USERNAME = @LOGNAME
BEGIN CASE
     CASE USERNAME = "TOM"
          DEPARTMENT = "HR"
      CASE USERNAME = "DICK"
           DEPARTMENT = "ADMIN"
      CASE 1 ------------------------> if none of the Case statements match
                                       then this statement would get
                                       executed
      "DEPARTMENT NOT FOUND"
 END CASE
```

III. For Loop

Use the For Loop to execute a set of statements repeatedly until a specific condition is met or for specific number of times. The counted loop uses a variable to hold the iteration count. This commences at the start value for the loop is automatically incremented by a step value at each iteration. Once it has passed the end value, the loop terminates.

Example :

```
      FOR COUNTER = 1 TO 10
            CRT "TEMENOS GLOBUS"  ------> The string TEMENOS GLOBUS will
                                          get
      NEXT COUNTER                        printed 10 times
```

IV. Open Loop

The open loop specifies a more powerful loop construction which will continue to iterate until a condition is met to terminate this. The condition is held in the WHILE clause. The REPEAT statement takes the control back to the first line after the LOOP statement.

Example :

```
LOOP
  CRT "Input 2 Numbers"
  INPUT Y.NUM1
  INPUT Y.NUM2
  WHILE Y.NUM1:Y.NUM2 ------------> Note that a condition is being checked using the While
                                    clause. ':' is the concatenation operator in Infobasic. The
                                    While statement specified here checks if Y.NUM1 and
                                    Y.NUM2 contain values.
  CRT "Total " : Y.NUM1 + Y.NUM2
REPEAT
```

Note :
Following are the boolean operators used in Infobasic

=EQ	Equality
#<>NE	Inequality
>GT	Greater Than
>=GE	Greater Than/Equal
<=LT	Less Than
<=LE	Less Than/Equal
MATCHES	Pattern Match

7.1 Built In Infobasic Functions

Infobasic has a number of built in functions that help in rapid code development. Some of the commonly used build in functions are listed below.

I. Len
II. Count
III. Dcount
IV. UpCase
V. DownCase
VI. Change
VII. Iconv
VIII. Oconv

I. Len

Use the LEN function to return the number of characters in string.

Example :

```
Var1  = Len("TEMENOS")
Var1  = 8
```

II. Count

Use the COUNT function to return the number of times a substring is repeated in a string value.

Example :

```
Var1  = "abc,def,ghi"
Var2  = COUNT(Var1,",")  ---> The COUNT function here is used to count the
                             number of "," in the string held in the variable var1
Var2  = 3
```

III. Dcount

Use the DCOUNT function to return the number of delimited fields in a data string.

Example :

```
Var1  = "abc,def,ghi"
Var2  = DCOUNT(Var1,",")
```

→ The DCOUNT function here is used to count the number of fields delimited by the delimiter "," in the string held in the variable var1

```
Var2  = 4
```

Note :

DCOUNT basically counts the number of delimiters and adds one to the result and displays. When the number of delimiters need to be obtained, use the COUNT function. When the actual number of values need to be obtained, use the DCOUNT function.

IV. UpCase

Use the UpCase function to convert the passes string to UPPER CASE.

Example :

```
Var1  = UPCASE("temenos")
Var1  = TEMENOS
```

V. DownCase

Use the DownCase function to convert the passed string to *lower case*

Example :

```
Var1  = DOWNCASE("TEMENOS")
Var1  = temenos
```

VI. Change

Use the CHANGE function to replace a substring in expression with another substring. If you do not specify occurrence, each occurrence of the substring is replaced.

Example :

```
Var1  = CHANGE("TEMENOOS","OO","O")
Var1  = TEMENOS
```

VII. Iconv

Use the ICONV function to convert string to a specified internal storage format. string is an expression that evaluates to the string to be converted. If the string evaluates to a null value, null is returned.

Example :

VIII. Oconv

Use the OCONV function to convert string to a specified format for external output. The result is always a string expression.

Example :

```
DATE=OCONV('9166',"D2") →   3 Feb 93
```

9.1 Writing Subroutines In Infobasic

You would be aware by now that Infobasic allows us to create programs as well as subroutines, which are to be executed from within Globus.

```
SUBROTUINE SubroutineName
$INSERT I_COMMON   } ──────────► Insert Files
$INSERT I_EQUATE   }

Actual Statements
Actual Statements

RETURN
END
```

Figure 4.1 Structure Of A Subroutine

All subroutines have to compulsorily begin with the line SUBROUTINE SubroutineName and end with RETURN and END. The subroutine name and the name of the file where the subroutine is to be stored need not have the same name. But as a convention, inorder to aviod unnecessary confusion the subroutine name and the file name are kept the same.

Insert files are similar to 'Include' files that you might have used in 'C' and 'C++' programs. There are number of insert files available. Each one of them contain some inbuilt functionality which can be used in our programs/subroutines. This enables re-usability of code.

I_COMMON and I_EQUATE are two main insert files available in Globus. I_COMMON defines all common global variables that can be used across subroutines and the file I_EQUATE initializes those common variables. It is a good practice to include these files in every subroutine we write irrespective of whether we are to use common global variables or not. These insert files are available under the directory GLOBUS.BP.

Example 2

Write a subroutine that will display the details(Id, Mnemonic and Nationality)of a customer whose id is 100069

Solution 2

Step 1

Algorithm:

Step 1. Open the Customer File
Step 2. Read the Customer file and extract the record with id 100069
Step 3. From the extracted record obtain the mnemonic and nationality
Step 4. Display the customer id, mnemonic and nationality.

Step 1 :

Inorder to open the Customer file we can use the Infobasic command OPEN.

```
OPEN FBNK.CUSTOMER ……..
```

When we use the UniVerse command OPEN to open a file, we need to supply the exact file name(along with the prefix). If programs are written using OPEN statements , they do not become portable across branches of a bank as each branch will have a different mnemonic to indentify itself uniquely.

For Instance

Bank XYX
In Branch1
In a subroutine we open the customer file by using UniVerse OPEN statement
`OPEN FBR1.CUSTOMER`

In Branch2
If the above subroutine with the OPEN statement were to be executed in this branch, the subroutine would return a fatal error saying that it cannot open the file. The name of the customer file in this branch is FBR2.CUSTOMER.

Inorder to overcome this problem or program portability, we need to use the core Globus subroutine OPF instead of Open.

OPF :

OPF is a core Globus subroutine that is used to open files.

Syntax :

`CALL OPF(Parameter1,Parameter2)`
Parameter 1 → The name of the file to be opened prefixed with a F.
Parameter 2 -> Path of the file to be opened. This is usually specified as ‘ ‘

Example :

```
FN.CUS = ‘F.CUSTOMER’
F.CUS = ‘’
CALL OPF(FN.CUS,F.CUS) ——————→ Code to open the Customer file
```

Working Of OPF :

The core Globus subroutine OPF is to be used. It takes in 2 parameters:

1 - The name of the file to be opened
2 - Path of the file

Both the above mentioned parameters are to be stored in variables and then passed to the OPF subroutine.

```
FN.CUS = ‘F.CUSTOMER’
```

The name of the variable that is to store the file name has to begin with “FN.” followed by a string that denotes the file that is to be opened. Just supply the value “F.” followed by the name of the file to open like above to the variable FN.CUS.

When the OPF subroutine gets executed, the COMPANY file is read inorder to obtain the mnemonic of the bank. Then the FILE.CONTROL record of the file is read to find out the type of file(INT,CUS or FIN). Once the file type is extracted, the ‘F.’ in the file name gets replaced with

“FBankMnemonic” - FBNK thus making subroutines portable across branches.

```
F.CUS = ‘’
```

The name of the variable that will hold the path of the file has to begin with a ‘F.’ followed by a string that denotes the file that is to be opened. This string has to be the same as that of the file name(FN) variable. This variable should be equated to a null(‘’).
When OPF gets executed, the VOC entry for the file is read and the path of the data file gets populated in this variable.

Step 2 :

Inorder to read the Customer file the Infobasic command READ can be used

```
READ FBNK.CUSTOMER ……..
```

But this will result in the same problem as the OPEN statement did – Portability of programs across branches of a Bank.

Inorder to overcome this problem, we use the core Globsu subroutine F.READ.

Syntax :

F.READ(FileName,Id of the record to be read,Dynamic array that will hold the read record,Filepath,Error variable)

Example :

```
Y.CUSID = “100069”
CALL F.READ(FN.CUS,Y.CUSID,R.CUSTOMER,F.CUS,CUS.ERR1)
```

Note R.CUSTOMER is a dynamic array that will hold the extracted customer record. It does not require declaration, but initializing it to a ‘’ would be a good programming practice. The error variable CUS.ERR1 will hold ‘null’ if the read is successful else will hold a numeric value. Note that the id of the record has been supplied using a variable.

Contents Of R.CUSTOMER

Note that the values of fields have been delimited using a field marker(®) and multi values have been delimited using the value marker(ÿ). There aren’t any sub values in this customer record.

```
DAOHENGBK®DAO HENG BANK INC®DAO HENG BANK INC®®119 ASIAN MANSION 209 DELA ROSA S
T®LEGASPI VILLAGE MAKATI CITY MAN PH®®®®1111®90®®8100®999®PH®4®PH®20000101®®®®®2
0000101®®®1®®®®®®®®®®®®®®®®®1®18_RICKBANAT1ÿ28_ANDREABARNES1®0006121042®18_RIC
KB
```

In order to obtain the mnemonic and the nationality of the customer, we need to access the dynamic array R.CUSTOMER. To extract values from a dynamic array, angular brackets “< >” need to be used.(Use ‘()’ for dimensioned arrays)

We can extract data from the dynamic array by specifying field positions as follows

```
Y.MNEMONIC = R.CUSTOMER<1>
```

or by specifying the actual name of the field.**It is always advisable to use field names ‘coz field positions could change from one release of Globus to another**. Here 1 is the field position of the field mnemonic in the CUSTOMER file.

How does one know the field numbers and the field names?

Most of the files in Globus have insert files which begin with 'I_F.' followed by the name of the file. They will be available under GLOBUS.BP.These files hold the names and the field positions of the various fields. These fields could have prefixes/suffixes.

For the customer insert file

Prefix used is : EB.CUS
Suffix used is : NIL

I_F.CUSTOMER File – Insert File For The Customer Application

```
0001: * Version 6 15/05/01  GLOBUS Release No. G12.0.00 29/06/01
0002: * File Layout for CUSTOMER Created 15 MAY 01 at 05:02pm by bhatiab
0003: *      PREFIX[EB.CUS.]      SUFFIX[]
0004:        EQU EB.CUS.MNEMONIC TO 1,          EB.CUS.SHORT.NAME TO 2,
0005:              EB.CUS.NAME.1 TO 3,              EB.CUS.NAME.2 TO 4,
0006:              EB.CUS.STREET TO 5,        EB.CUS.TOWN.COUNTRY TO 6,
0007:       EB.CUS.RELATION.CODE TO 7,        EB.CUS.REL.CUSTOMER TO 8,
0008:     EB.CUS.REVERS.REL.CODE TO 9,             EB.CUS.SECTOR TO 10,
0009:    EB.CUS.ACCOUNT.OFFICER TO 11,      EB.CUS.OTHER.OFFICER TO 12,
0010:           EB.CUS.INDUSTRY TO 13,             EB.CUS.TARGET TO 14,
0011:        EB.CUS.NATIONALITY TO 15,    EB.CUS.CUSTOMER.STATUS TO 16,
0012:          EB.CUS.RESIDENCE TO 17,       EB.CUS.CONTACT.DATE TO 18,
0013:         EB.CUS.INTRODUCER TO 19,               EB.CUS.TEXT TO 20,
0014:           EB.CUS.LEGAL.ID TO 21,   EB.CUS.REVIEW.FREQUENCY TO 22,
0015:  EB.CUS.BIRTH.INCORP.DATE TO 23,    EB.CUS.GLOBAL.CUSTOMER TO 24,
0016: EB.CUS.CUSTOMER.LIABILITY TO 25,           EB.CUS.LANGUAGE TO 26,
0017:   EB.CUS.POSTING.RESTRICT TO 27,      EB.CUS.DISPO.OFFICER TO 28,
0018:           EB.CUS.POST.CODE TO 29,           EB.CUS.COUNTRY TO 30,
0019:                EB.CUS.BOOK TO 31,         EB.CUS.CONFID.TXT TO 32,
0020:         EB.CUS.RESERVED07 TO 33,         EB.CUS.RESERVED06 TO 34,
0021:         EB.CUS.RESERVED05 TO 35,         EB.CUS.RESERVED04 TO 36,
0022:         EB.CUS.RESERVED03 TO 37,         EB.CUS.RESERVED02 TO 38,
0023:         EB.CUS.RESERVED01 TO 39,          EB.CUS.LOCAL.REF TO 40,
0024:           EB.CUS.OVERRIDE TO 41,      EB.CUS.RECORD.STATUS TO 42,
0025:             EB.CUS.CURR.NO TO 43,          EB.CUS.INPUTTER TO 44,
0026:          EB.CUS.DATE.TIME TO 45,         EB.CUS.AUTHORISER TO 46,
0027:             EB.CUS.CO.CODE TO 47,          EB.CUS.DEPT.CODE TO 48,
0028:       EB.CUS.AUDITOR.CODE TO 49,    EB.CUS.AUDIT.DATE.TIME TO 50
```

Note the field number and the field name

Therefore to extract the mnemonic and nationality of the customer we need to use the following code

```
Y.MNEMONIC = R.CUSTOMER<EB.CUS.MNEMONIC>
Y.NATIONALITY = R.CUSTOMER<EB.CUS.NATIONALITY>
```

Step 4 :

To display the Id, Mnemonic and Nationality values extracted we could use the Infobasic command CRT.

Syntax :

```
CRT VariableName/"String"
```

Example :

```
CRT "Customer Id : ":Y.CUSID
CRT "Customer Mnemonic : ":Y.MNEMONIC
CRT "Customer Nationality : ":Y.NATIONALITY
```

Consolidated Solution 2

```
*Subroutine to display the details of customer 100069
SUBROUTINE CUS.DISPLAY.DETAILS
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
      FN.CUS = 'F.CUSTOMER'
      F.CUS = ''
      Y.CUS.ID = 100069
      Y.MNEMONIC = ''
      Y.NATIONALITY = ''
      R.CUSTOMER = ''
      CUS.ERR1 = ''
RETURN
OPENFILES:
      CALL OPF(FN.CUS,F.CUS)
RETURN
PROCESS:
      CALL F.READ(FN.CUS,Y.CUS.ID,R.CUSTOMER,F.CUS,CUS.ERR1)
      Y.MNEMONIC = R.CUSTOMER<EB.CUS.MNEMONIC>
      Y.NATIONALITY = R.CUSTOMER,EB.CUS.NATIONALITY>
      CRT "Customer Id: ":Y.CUS.ID
      CRT "Customer Mnemonic: ":Y.MNEMONIC
      CRT 'Customer Nationality: ":Y.NATIONALITY
RETURN
END
```

Note :

In the above subroutine, the code has been split and made part of 3 different paragraphs. In order to achieve modularity and to make maintenance of code easier, it is advisable to make use of paragraphs. Every paragraph has to have a name and has to end with a RETURN statement.A *GOSUB ParagraphName* statement takes the control to that specific paragraph. Once the statements inside the paragraph get executed, the *RETURN* statement takes the control back to the line after the GOSUB statement that actually invoked this paragraph. This type of modular programming needs to be used for a lengthy subroutine. Incase the number of lines that constitute the subroutine is very less, the programmer could choose to write code using the Top Down approach of programming where there will be no paragraphs at all.

Note :

We need to compile and catalogue this subroutine now. Use

```
EB.COMPILE BP CUS.DISPLAY.DETAILS
```

Compile and catalogue.

How Do We Execute This Subroutine From Globus?

As you would be aware by now, anything that needs to be executed from the 'Awaiting Application' prompt in Globus needs to have an entry in the PGM.FILE. Inorder to execute out subroutine from within Globus, we need to make an entry in the PGM.FILE. Ensure that you set the type in the PGM.FILE to 'M' (Mainline program). The ID of the PGM.FILE entry should be the name of the file which stores the subroutine.

```
TEMENOS MODEL BANK V5   PROGRAM FILE SEE

    PROGRAM            CUS.DISPLAY.DETAILS
-------------------------------------------------------
  1 TYPE.............. M
  2. 1 GB SCREEN.TITLE CUSTOEMR DETAILS
  5 PRODUCT........... RE
 14 CURR.NO........... 1
 15. 1 INPUTTER....... 1668_TRAINEE01
 16. 1 DATE.TIME...... 25 JAN 02 11:09
 17 AUTHORISER........ 1668_TRAINEE01
 18 CO.CODE........... US-001-0001
 19 DEPT.CODE......... 1
```

Note :

If we type CUS.DISPLAY.DETAILS in the Awaiting Application prompt we would see no output.

What happened? No results!!!!!!! Don't panic.

Let us add the DEBUG statement to the subroutine and see the display (Add it just after the insert files)

```
SUBROUTINE CUST.DISPLAY.DETAILS
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER
DEBUG
GOSUB INIT
```

Ensure that you compile and catalogue after making any changes to the subroutine.

Execute The Subroutine

Type the name of the subroutine in the 'Awaiting Application' prompt and see the execution of the program line by line.

```
 ACTION  CUS.DISPLAY.DETAILS:   7:       GOSUB INIT
:: S
 CUS.DISPLAY.DETAILS:  13:       FN.CUS = 'F.CUSTOMER'
:: S
 CUS.DISPLAY.DETAILS:  14:       F.CUS = ''
:: S
 CUS.DISPLAY.DETAILS:  15:       Y.CUS.ID = 100069
:: S
 CUS.DISPLAY.DETAILS:  16:       R.CUSTOMER = ''
:: S
 CUS.DISPLAY.DETAILS:  17:       CUS.ERR1 = ''
:: S
 CUS.DISPLAY.DETAILS:   8:       GOSUB OPENFILES
:: S
 CUS.DISPLAY.DETAILS:  23:       CALL OPF(FN.CUS,F.CUS)
:: S
 CUS.DISPLAY.DETAILS:  24:       RETURN
:: FN.CUS/
STRING: T r L=13 `FBNK.CUSTOMER'
:: F.CUS/
DBFILE: A ..\mbdemo.data\st\FBNK.CUST000 Mod=31 Sep=4 Type=2
::
```

```
 CUS.DISPLAY.DETAILS:   9:       GOSUB PROCESS
:: S
 CUS.DISPLAY.DETAILS:  27:       CALL F.READ(FN.CUS,Y.CUS.ID,R.CUSTOMER,F.CUS
5.ERR1)
:: S
 CUS.DISPLAY.DETAILS:  28:       Y.MNEMONIC = R.CUSTOMER<EB.CUS.MNEMONIC>
:: S
 CUS.DISPLAY.DETAILS:  29:       Y.NATIONALITY = R.CUSTOMER<EB.CUS.NATIONALIT
:: S
 CUS.DISPLAY.DETAILS:  30: CRT "Customer Id: ":Y.CUS.ID
:: R.CUSTOMER/
5TRING: T r L=259 `DAOHENGBK§DAO HENG BANK INC§DAO HENG BANK INC§§119 ASIAN M
ION 209 DELA ROSA ST§LEGASPI VILLAGE MAKATI CITY MAN PH§§§§1111§90§§8100§999§
4§PH§20000101§§§§§20000101§§§1§§§§§§§§§§§§§§§§§1§18_RICKBANAT1ÿ28_ANDREABARNE
D0061210425§18_RICKBANAT1§US0010001§1§§'
:: Y.MNEMONIC/
5TRING: T r L=9 `DAOHENGBK'
:: Y.NATIONALITY/
5TRING: T r L=2 `PH'
::
Customer Id: 100069
 CUS.DISPLAY.DETAILS:  31:       CRT "Customer Mnemonic: ":Y.MNEMONIC
:: S
Customer Mnemonic: DAOHENGBK
 CUS.DISPLAY.DETAILS:  32:       CRT "Customer Nationality: ":Y.NATIONALITY
:: S
Customer Nationality: PH
 CUS.DISPLAY.DETAILS:  33:       RETURN
:: S
 CUS.DISPLAY.DETAILS:  10:       RETURN
:: S
```

Note :

Some DEBUGger commands

S - To execute the line
Variablename/ - To see the contents of a variable
Q - Quit out of the subroutine and return to the UniVerse prompt.

Press Ctrl + C to abort the subroutine.
Break: Option (A,C,L,Q,D,?) - Choose A. Will return to the UniVerse prompt

Example 3

Modify example 1 to display theid, mnemonic and nationality of all customers.

Solution 3

Algorithm

Step 1. Open the Customer File
Step 2. Select all the customer ids
Step 3. Remove one customer id from the selected list
Step 4. For the extracted customer id extract the corresponding record from the customer file
Step 5. From the extracted record extract the mnemonic and nationality
Step 6. Display the customer id, mnemonic and the nationality
Repeat Steps 3 to 6 for all customers

Step 1

You would be aware by now that we need to use OPF to open any file in Globus.

```
FN.CUS = 'F.CUSTOMER'
F.CUS = ''
CALL OPF(FN.CUS,F.CUS)
```

Step 2

We need to select all the customer ids from the Customer file. In order to achieve this we need to execute a Select statement that will pick up all the Customer ids. Select statements can be executed within subroutines. In order to execute select statements within a subroutine, we need to first assign the select statement to a variable and then execute the contents of the variable using the core Globus subroutine EB.READLIST. **Please note that a Select statement executed from within a subroutine can only return the ids from the file on which the Select statement is based.**

EB.READLIST

EB.READLIST is a core Globus subroutine that is used to execute a Select statement within a subroutine

Syntax :

EB.READLIST takes in 5 parameters.

1 - The select statement to be executed. Give the name of the variable that holds the select statement here.
2 - The name of a dynamic array that will hold the result of the select statement. Please note that a select statement here can only select ids from the respective file. Therefore this dynamic arrays will only hold the ids of the records that have been selected. All the ids will be delimited by a field marker(FM).
3 - This is an optional parameter. This is the name of a file in the hard disk that can hold the result of the select statement. Usually this is set to NULL ('')
4 - A variable that will hold the number of records selected.
5 - A variable to hold the return code. Will contain null if the select statement was successful else will contain 1 or 2.

```
SEL.CMD = "SELECT  ":FN.CUS
```

Note the space. If this space is not given then SEL.CMD will contain "SELECTFBNK.CUSTOMER" thus resulting in an error in EB.READLIST

```
CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,CUS.ERR)
```

Step 3 And 4

Use LOOP and REMOVE(Discussed Earlier) to repeat Steps 3 to 6

Consolidated Solution 3

```
*Subroutine to display the mnemonic and nationality of all customers
SUBROUTINE CUS.DISPLAY.DETAILS
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER
      DEBUG
      GOSUB INIT
      GOSUB OPENFILES
      GOSUB PROCESS
```

```
      RETURN
INIT:
      FN.CUS = 'F.CUSTOMER'
      F.CUS = ''
      Y.CUS.ID = ''
      R.CUSTOMER = ''
CUS.ERR1 = ''
      Y.MNEMONIC = ''
      Y.NATIONALITY = ''
      SEL.CMD = ''
      SEL.LIST = ''
      NO.OF.REC = 0
      RET.CODE = ''
      RETURN
OPENFILES:
      CALL OPF(FN.CUS,F.CUS)
      RETURN
PROCESS:
     SEL.CMD = "SELECT ":FN.CUS
     CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET.CODE)
      LOOP
          REMOVE Y.CUS.ID FROM SEL.LIST SETTING POS
          WHILE Y.CUS.ID:POS
               CALL  F.READ(FN.CUS,Y.CUS.ID,R.CUSTOMER,F.CUS,CUS.ERR1)
               Y.MNEMONIC = R.CUSTOMER<EB.CUS.MNEMONIC>
               Y.NATIONALITY =  R.CUSTOMER<EB.CUS.NATIONALITY>
               CRT "Customer Id: ":Y.CUS.ID
               CRT "Customer Mnemonic: ":Y.MNEMONIC
               CRT "Customer Nationality: ":Y.NATIONALITY
      REPEAT
      RETURN
   END
```

Note :

Use the REMOVE statement to successively extract dynamic array elements that are separated by system delimiters. When a system delimiter is encountered, the extracted element is assigned to variable.

In order to execute the above subroutine, we need to compile and catalogue it. An entry in the PGM.FILE has to be made to execute it from within Globus. In order to see the execution of the subroutine line by line, we need to add the DEBUG

Example 4

Modify Example 2 to store the extracted mnemonic and nationality of all customers in an array(do not display them) delimited by a '*'. The array should contain data as follows

CusId*Mnemonic*NationalityFMCusId*Mnemoic*Nationality

Solution 4

Algorithm :

Step 1. Open the Customer File
Step 2. Select all the customer ids
Step 3. Remove one customer id from the selected list
Step 4. For the extracted customer id extract the corresponding record from the customer file
Step 5. From the extracted record extract the mnemonic and nationality
Step 6. Store the customer id, mnemonic and the nationality in a dynamic array
Repeat Steps 3 to 6 for all customers

Step 1, 2 ,3 ,4 And 5

As discussed earlier we could go ahead and use OPF, F.READ, LOOP, REMOVE and REPEAT to accomplish the above mentioned steps.

Step 6

In order to append the extracted values into an array we could use the following method.

```
ArrayName<-1> = Value
```

In our case, once we extract the mnemonic and the nationality of the customer we could concatenate the id, mnemonic and the nationality of the customer delimited with a '*' and then store it in a dynamic array.

Every time a new value comes in, the existing values get pushed down by one position. This is achieved by the '-1' that we specify along with the array name. All values get appended, delimited by a field marker 'FM'.

```
MAINARRAY<-1> = Y.CUSID:'*':Y.MENMONIC:'*':Y.NATIONALITY
```

The array will look like this after all values have been concatenated

*11111*AAA*INFM22222*BBB*INFM 33333*CCC*INFM 44444*DDD*IN*

Note -
To have values in an array delimited by value markers use
ArrayName<**1,-1**> = Value1:'*':Vale2:'*':Value3:'*':Value4
To have values in an array delimited by sub value markers use
ArrayName<**1,1,-1**> = Value1:'*':Vale2:'*':Value3:'*':Value4

Consolidated Solution 4

```
*Subroutine to store the id, mnemonic and nationality of all
*customers in an array
SUBROUTINE CUS.DISPLAY.DETAILS
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER

      GOSUB INIT
      GOSUB OPENFILES
      GOSUB PROCESS
      RETURN

INIT:
      FN.CUS = 'F.CUSTOMER'
      F.CUS = ''
      Y.CUS.ID = ''
      R.CUSTOMER = ''
      CUS.ERR1 = ''
      Y.MNEMONIC = ''
      Y.NATIONALITY = ''
      SEL.CMD = ''
      SEL.LIST = ''
      NO.OF.REC = 0
      RET.CODE = ''
      CUS.DETAILS.ARRAY = ''
      RETURN
```

```
OPENFILES:
      CALL OPF(FN.CUS,F.CUS)
      RETURN
PROCESS:
      SEL.CMD = "SELECT ":FN.CUS
CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET.CODE)
      LOOP
         REMOVE Y.CUS.ID FROM SEL.LIST SETTING POS
      WHILE Y.CUS.ID:POS
         CALL F.READ(FN.CUS,Y.CUS.ID,R.CUSTOMER,F.CUS,CUS.ERR1)
         Y.MNEMONIC = R.CUSTOMER<EB.CUS.MNEMONIC>
         Y.NATIONALITY = R.CUSTOMER<EB.CUS.NATIONALITY>
         CUS.DETAILS.ARRAY<-1> = Y.CUS.ID:'*':Y.MNEMONIC:'*':Y.NATIONALITY
      REPEAT
      RETURN
   END
```

Note :

In order to execute the above subroutine, we need to compile and catalogue it. An entry in the PGM.FILE has to be made to execute it from within Globus. In order to see the execution of the subroutine line by line, we need to add the DEBUG statement.

Additional Information :

F.WRITE

F.WRITE is a core Globus subrotutine that is used to write a record on to a file.

Syntax:

```
F.WRITE(FileName,Id of the record to be written,Actual record to be
written)

CALL F.WRITE(FN.CUS,Y.CUS.ID,R.CUSTOMER)
```

F.DELETE

F.DELETE is also a core Globus subroutine that is used to delete a record from a file.

Syntax :

```
F.DELETE(FileName,Id of the record to be deleted)

CALL F.DELETE(FN.CUS,Y.CUSID)
```

8.1 Infobasic Commands

Infobasic has a number of built in commands as well that enable rapid code development. Find below some very commonly used Infobasic commands,

I. OPENSEQ
II. READSEQ
III. WRITESEQ
IV. CLOSESEQ
V. MATREAD
VI. MATWRITE
VII. LOCATE

I. OPENSEQ

OPENSEQ is used open a sequential file. If the file that you are trying to open does not exist, and you wish to create it you could create it by specifying CREATE statement in the ELSE clause, OPENSEQ has the capability to do it(Achieved by the ELSE clause)

Syntax

OPENSEQ "path of the file and the file name" TO filevariable ON ERROR
STOP "message"
END ELSE
CREATE filevariable ELSE STOP "message"
END

Example

OPENSEQ "/globus/temenos.txt" TO TEM1 ON ERROR
STOP "Unable to open temenos.txt"
END ELSE
CREATE TEM1 ELSE STOP "Unable to create temenos.txt"
END

II. READSEQ

READSEQ is used to read data from a sequential file. While reading data from a afile, READSEQ uses the new line character CHAR(10) as the delimiter. Once the end of the file is reached the ELSE clause statements are executed.

Syntax

```
READSEQ variable FROM file.variable
THEN statements ELSE statements
```

Example

```
READSEQ  data1  FROM TEM1
THEN CRT “Read operation complete” ELSE CRT “Cannot read from file temenos.txt”
```

III. WRITESEQ

WRITESEQ is used to write to a sequential file. It writes the expression as the next line to the file opened to file.variable using a new line character CHAR(10) as the delimiter.

Syntax

```
WRITESEQ expression TO file.variable
ON ERROR statements
THEN statements
ELSE statements
```

Example

```
WRITESEQ “Temenos Globus” TO TEM1
THEN CRT “Data Written” ELSE CRT “Unable to write data into temenos.txt”
```

IV. CLOSESEQ

CLOSESEQ is used to write an end-of-file mark and to make the file available to other users. It is very important that you use CLOSESEQ on any file opened with OPENSEQ, because CLOSESEQ releases the READU lock that was taken by the OPENSEQ statement.

Syntax

```
CLOSESEQ file.variable
ON ERROR statements
```

Example

```
CLOSESEQ TEM1
ON ERROR CRT “Unable to write an end-of-file mark on temenos.txt”
```

V. LOCATE

LOCATE statement is used to locate the position of a string or determine the position to insert in to maintain a specific sequence.

Syntax

```
LOCATE expr IN dynamic.array<FIELD,VALUE>,STARTPOS
BY sort.expr SETTING variable
THEN statements ELSE statements
```

Additional Information

Sort.Expr :

AL Ascending left(Alpha sort)
AR Ascending right(Numeric sort)
DL Descending left(High-low alpha sort)
DR Descending right(High-low numeric sort)

Example

```
DAYS = “MON:”FM:”TUE”:FM:”WED”:FM:”THU”:FM:”FRI”
LOCATE “WED” IN DAYS SETTING FOUND ELSE FOUND = 0
CRT “Position of WED in DAYS dynamic array :”:FOUND
LOCATE “SAT” IN DAYS BY “AL” SETTING POS ELSE
        INS “SAT” BEFORE DAYS<POS>
END
CRT “Position where SAT has been inserted :”:POS
CRT “Days dynamic array after inserting SAT :”:DAYS
```

Output

```
Position of WED in DAYS dynamic array : 3
Position where SAT has been inserted : 2
Days dynamic array after inserting SAT : MON®SAT®TUE®WED®THU®FRI
```

VI. MATREAD

MATREAD is a command that is used to read the contents of a dimensioned/dynamic array. You can specify the id of the record to be picked up from the array. Incase the read is successful, then the statements following the ‘THEN’ statements are executed else the statements following the ‘ELSE’ statement are executed.

Syntax

MATREAD array FROM file.variable, record.ID THEN statements ELSE statements

Example

MATRED Array1 from F.REGISTER.DETAILS,ID1 THEN ….. ELSE …..

The above statement will search for a record with id specified in the variable ID1, if found, it will transfer the record to the array Array1.

VII. MATWRITE

MATWRITE is used to build a dynamic array from a specified dimensioned array and write it to the file opened to file.variable using a key of record.id.

Syntax

MATWRITE matrix ON file.variable,KEY

Example

```
DIM ARRAY1(5)
MATREAD ARRAY1 FROM TEM1,101ELSE
```

```
        MAT ARRAY1 = ‘’
END
MATWRITE ARRAY1 ON TEM1,100
```

Note :

Use HELP BASIC functionname/commandname at the uniVerse prompt to get help on any of the Infobasic commands or functions.