TEMENOS™

Infobasic Programming - jBASE

DateOf Issue	Version	Changes	By
2001	1.0	Initial	Alagammai Palaniappan
2002	1.1	jBASE Changes	Alagammai Palaniappan
2002	1.2	Infobasic Programming changes	Alagammai Palaniappan
July 2004	1.3	TXN Mgt changes and overall update	Alagammai Palaniappan
September 2004	1.4	EB.READLIST amendment and CACHE.READ inclusion	Alagammai Palaniappan
December 2004	1.5	Minor Changes and formatting	Sara Cleur

Table of Content

Objectives

- To understand the concept of arrays
- To understand the steps to write a simple routine in Infobasic
- To understand the need to compile and catalogue subroutines
- To understand the working of the jBASE environmental variables JBCDEV_BIN, JBCDEV_LIB, JBCOBJECTLIST and PATH.
- To understand the need of the jBASE configuration file jLibDefinition
- To understand and use T24 subroutines like OPF, F.READ etc
- To understand and use Infobasic commands like LOOP, REMOVE etc.
- To understand the command that are used to access sequential files
- To understand the creation of subroutines that take in and return arguments
- To understand the creation of functions.

Introduction

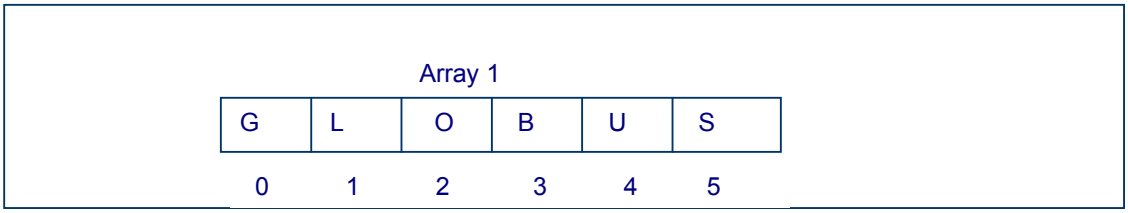
As you would be aware by now, T24 uses jBASE as the back end to store its data. All programs that make up the T24 core are written in a language called Infobasic. Infobasic is a very simple yet powerful programming language. With its English like statements, it makes programming very simple. A salient 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.

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. In case a series of characters (string) like ‘T24’ 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 continuous 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’.



Note : In case you wish to access ‘G’ in ‘GLOBUS’ , then you would usually specify Array1[0]

Types Of Arrays

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

I. Dynamic Arrays

II. Dimensioned Arrays

Dynamic Arrays

Dynamic arrays are, as the name implies, dynamic in 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

A field marker separates each field 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**FM**Field4**FM**SubValue1**SM**SubValue2**FM**Field5

Note
All variables in Infobasic are treated as dynamic arrays. Dynamic arrays do not need any explicit declaration. Initialisation would suffice.

Storage Of Data In A Dynamic Array

The following record is a part of the TEMENOS.TRG file.

1 Name	TemenosTrg
2.1 Address	India
2.2 Address	UK
2.3 Address	Geneva
3.1 Course Category	Technical
4.1.1 Course Name	JBASE
4.1.2 Course Name	T24
3.2 Course Category	Functional

4.2.1 Course Name	Lending
4.2.2 Course Name	Financials
5 Free Text	
6 Inputter	TRAINER.1

If the above record were to be stored in a dynamic array, it would be as follows

TemenosTrg**FM**India**VM**UK**VM**Geneva**FM**Technical**VM**Functional**FM**
jBASE**SM**T24**VM**Lending**SM**Financials**FMFM**Trainer.1

Please note that the FM, VM and SMs will be stored as characters but will be stored as special characters.

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

DIM ARRAY2(5,3)

5 - > Refers to the number of rows

3 - > Refers to the number of columns

You can also create single dimensioned arrays. This type of dimensioned arrays will only have a fixed number of rows. The number of columns will unlimited. In this case, each row in the dimensioned array will be a dynamic array.

DIM ARRAY3(5)

5 - > Refers to the number of rows

Columns - Unlimited

Structure Of An Infobasic Program

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

Any program that is executed from the database prompt is termed as a ‘PROGRAM’ and a program that is executed from within T24 is termed as a ‘SUBROUTINE’.

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

Usually, any program or subroutine developed by the user is stored under a directory named BP and the core T24 programs or subroutines are stored under GLOBUS.BP. Never store user written programs/subroutines in the GLOBUS.BP directory.

Compiling And Cataloguing 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.

Compiling Subroutines

When T24 is installed, a directory named “globuslib” and “lib” get installed under the home directory (run directory) of the user. The directory “globuslib” contains the object code of all core subroutines and the directory “lib” is supposed to contain the object code of all local subroutines. When a subroutine is compiled, an object code is produced. For instance when a subroutine TEMENOS whose source is under the BP directory is compiled an object code $TEMENOS is produced and is placed under the directory BP (The source directory). A subroutine also needs to be catalogued. The process of cataloguing refers an environmental variable called JBCDEV_LIB to obtain the path where it has to place the object file that is to be created. Once the path is obtained, the object code is placed under one of the library files under that path. Therefore, all object codes of all subroutines get stored under a library file under the path pointed to by the environmental variable JBCDEV_LIB. The library files mentioned above are controlled by a configuration file named jLibDefinition, which is present under the jBASE config directory (Referred by the environmental variable JBCGLOBALDIR). The jLibDefinition file specifies the naming convention of the library files and the maximum size of them as well. jBASE decides, under which library file the object code has to reside. If none of the existing library files under the directory pointed by the JBCDEV_LIB have space to store a new object code then jBASE will automatically create a new library file by referring the jLibDefinition file. jBASE will also swap object codes from one library file to another in order to utilize the existing space inside the library files to the maximum.

When a subroutine is executed, the environmental variable JBCOBJECTLIST is referred as it contains the search path for all T24 subroutines. It is similar to that of the Unix PATH variable that contains the search path of all Unix executables.

JBCDEV_LIB=$HOME/lib

JBCOBJECTLIST=$HOME/lib.$HOME/globuslib

Compiling Programs

When T24 is installed, two directories namely “globusbin” and “bin” get installed under the home directory (run directory) of the user. The directory “globusbin” contains the core T24 executables and the directory “bin” is supposed to contain the non-core/local executables. When a program is compiled an executable is produced. For instance when a program TEMENOS whose source is in the BP directory is compiled, an executable with the name $TEMENOS gets created under the BP (Source directory). The process of cataloguing refers an environmental variable JBCDEV_BIN to obtain the directory into which this executable needs to be placed.

When a program is executed, jBASE refers an environmental variable PATH, which contains the search path of jBASE executables as well. PATH, as you would be aware of by now, is not a jBASE variable but a UNIX variable and contains the search path of UNIX executables.

JBCDEV_BIN=$HOME/bin

PATH=$HOME/bin:$HOME/globusbin:$PATH

The values of JBCDEV_BIN, JBCDEV_LIB, JBCOBJECTLIST and PATH can be changed accordingly depending upon the requirement.

Writing Infobasic Programs

Example 1

Program to display “Hello World”

Solution 1

Step 1

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

Consolidated Solution 1

```
JED BP HELLO
New record.

PROGRAM HELLO
CRT "HELLO WORLD"
END

"HELLO" filed in file "BP".
```

JED is the jBASE editor. Please refer to 'Using JED Editor' notes that have been attached to this course material.

Step 2

Compile and catalog the program. Since this is a local program, the executable needs to go to "bin" and not "globusbin". Therefore check the value of JBCDEV_BIN. If it is pointing to any other directory other than bin change it to point to bin.

```
echo $JBCDEV_BIN
```

Note the output. If it is anything other than the bin directory then change the value of JBCDEV_BIN as follows

```
export JBCDEV_BIN=$HOME/bin
```

Please note that the above statement will only change the value of JBCDEV_BIN for the current session. If you want this change to be permanent, then make the change in the .profile file, logout and login for the change the take effect.

Please ensure that PATH first points to $HOME/bin, as, if there is a program with the same name and its executable resides in "globusbin" or any other directory that is specified first in PATH, then that program only would get executed.

EB.COMPILE BP HELLO ⟶ Command to compile and catalog a program. BP is the source directory name where the source code of the program resides.

Step 3

Execute the program by typing the following statement at the database prompt.

```
Jsh-->HELLO

HELLO WORLD ⟶ Output of the program
```

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

If Then Else

The IF clause is used to determine the flow to be executed depending on 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 that 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
```

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 (or OTHERWISE)
      "DEPARTMENT NOT FOUND"
 END CASE
```

If none of the Case statements match then this statement would get executed

For Loop

Use the For Loop to execute a set of statements repeatedly 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 for each iteration. Once it has passed the end value, the loop terminates.

Example

```
FOR COUNTER = 1 TO 10
   CRT "TEMENOS GLOBUS"
NEXT COUNTER
```

→ The string TEMENOS GLOBUS will get printer 10 times"

Open Loop

The open loop specifies a more powerful loop construction that 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
  CRT "Total " : Y.NUM1 + Y.NUM2
REPEAT
```

→ 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.

Built In Infobasic Functions

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

LEN

COUNT

DCOUNT

UPCASE

DOWNCASE

CHANGE

OCONV

LEN

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

Example

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

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,",")  ──►
Var2  = 2
```

The COUNT function is used to count the number of "," in the string held in the variable var1

DCOUNT

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

Example

```
Var1  = "abc,def,ghi"
Var2  = DCOUNT(Var1,",")  ──►
Var2  = 3
```

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

Note

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

UPCASE

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

Example

```
Var1    = UPCASE("temenos")  ──►
```

Var1 is now TEMENOS

DOWNCASE

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

Example

```
Var1  = DOWNCASE("TEMENOS")  ──►
```

Var1 is now temenos

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 is now TEMENOS

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

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 T24.

```
SUBROTUINE SubroutineName
$INSERT I_COMMON
$INSERT I_EQUATE

Actual Statements
Actual Statements

RETURN
END
```

→ Insert Files

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 must have the same name.

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 contains some inbuilt functionality that 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 T24. 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.

It has to be noted that common variables get loaded when a user signs on into T24. Many of the common variables get loaded when a user signs on and many of them get loaded when certain applications are started.

Example

R.USER – Will contain the currently signed on user's USER record. If any changes are made to the user record, these changes will take effect only if he logs off and logs in again.

ID.NEW – Will contain the ID of an application (either when F3 is pressed or when the ID is input manually).

Example 2

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

Solution 2

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

In order to open the Customer file we can use the command OPEN.

OPEN FBNK.CUSTOMER

When we use the 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 identify 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.

In order to overcome this problem or program portability, we need to use the core T24 subroutine OPF instead of Open.

OPF

OPF is a core T24 subroutine that is used to open files.

Syntax

CALL OPF(Parameter1,Parameter2)

Example

FN.CUS = 'F.CUSTOMER'

F.CUS = ''

CALL OPF(FN.CUS,F.CUS) ⟶ Code to open the Customer file

Working Of OPF

OPF takes in 2 parameters:

Parameter 1 → The name of the file to be opened prefixed with an F.

Parameter 2 → Path of the file to be opened. This is usually specified as ' '

Both the 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 in order 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

“F<BankMnemonic>” - 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

In order to read the Customer file use the T24 subroutine F.READ.

F.READ

Syntax

F.READ(<filename>,<id of record>,<dynamic array that contains the record that is read>,<file path>,<error variable>)

Example

Y.CUSID = “100069”

CALL F.READ(FN.CUS,Y.CUSID,R.CUSTOMER,F.CUS,CUS.ERR)

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(ʕ) and multi values have been delimited using the value marker(**ÿ**). There aren’t any sub values in this customer record.

```
DAOHENGBKʕDAO HENG BANK INCʕDAO HENG BANK INCʕʕ119 ASIAN MANSION 209 DELA ROSA S
TʕLEGASPI VILLAGE MAKATI CITY MAN PHʕʕʕʕ1111ʕ90ʕʕ8100ʕ999ʕPHʕ4ʕPHʕ20000101ʕʕʕʕʕ2
0000101ʕʕʕ1ʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕ1ʕ18_RICKBANAT1ÿ28_ANDREABARNES1ʕ0006121042ʕ18_RICKB
ANAT1ʕUS0010001ʕ1ʕʕ
```

Step 3

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 because field positions could change from one release of T24 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 T24 have insert files that begin with 'I_F.' followed by the name of the file. They will be available under the GLOBUS.BP directory. 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. In case 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.

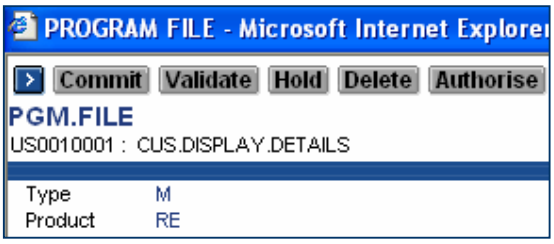
We need to compile and catalogue this subroutine now. Use

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

Compile and catalogue. Check variables JBCDEV_LIB and JBCOBJECTLIST and change if necessary.

How Do We Execute This Subroutine From T24?

As you would be aware by now, anything that needs to be executed from the 'Awaiting Application' prompt in T24 needs to have an entry in the PGM.FILE. In order to execute out subroutine from within T24, 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 that stores the subroutine.



Note

If we type CUS.DISPLAY.DETAILS in the Awaiting Application prompt we would see the output.

Debug Statement

The DEBUG statement shows the execution of a subroutine line by line.

Let us add the DEBUG statement to the subroutine and see the display (Add it at the point from where you wish to see the execution of the subroutine line by line)

```
SUBROUTINE CUST.DISPLAY.DETAILS
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER
GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN
INIT:
DEBUG
```

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 subroutine line by line.

```
Source changed to BP/CUS.DISPLAY.DETAILS
0011 DEBUG
jBASE debugger->S
0012         FN.CUS = 'F.CUSTOMER'
jBASE debugger->S
0013         F.CUS = ''
jBASE debugger->S
0014         Y.CUS.ID = 1038
jBASE debugger->S
0015         Y.MNEMONIC = ''
jBASE debugger->S
0016         Y.NATIONALITY = ''
jBASE debugger->S
0017         R.CUSTOMER = ''
jBASE debugger->S
0018         CUS.ERR1 = ''
jBASE debugger->S
0019 RETURN
jBASE debugger->S
```

```
0020 OPENFILES:
jBASE debugger->S
0021         CALL OPF(FN.CUS,F.CUS)
jBASE debugger->S
0022 RETURN
jBASE debugger->S
0023 PROCESS:
jBASE debugger->V FN.CUS
  FN.CUS                    : FBNK.CUSTOMER
jBASE debugger->V F.CUS
  F.CUS                     : File '../mbdemo.data/st/FBNK.CUST000'
jBASE debugger->S
0024         CALL F.READ(FN.CUS,Y.CUS.ID,R.CUSTOMER,F.CUS,CUS.ERR1)
jBASE debugger->S
0025         Y.MNEMONIC = R.CUSTOMER<EB.CUS.MNEMONIC>
jBASE debugger->S
0026         Y.NATIONALITY = R.CUSTOMER<EB.CUS.NATIONALITY>
jBASE debugger->S
0027         CRT "Customer Id: ":Y.CUS.ID
jBASE debugger->V Y.MNEMONIC
  Y.MNEMONIC                : TAKIZAKIT
jBASE debugger->V Y.NATIONALITY
  Y.NATIONALITY             : JP
jBASE debugger->S
Customer Id: 1038
0028         CRT "Customer Mnemonic: ":Y.MNEMONIC
jBASE debugger->S
Customer Mnemonic: TAKIZAKIT
0029         CRT "Customer Nationality: ":Y.NATIONALITY
jBASE debugger->S
Customer Nationality: JP
0030 RETURN
jBASE debugger->Q
Are you sure ?Y
jBASE debugger , QUIT from program 'EX'
jsh geneva ~ -->
```

Note

Some DEBUGger commands

S - To execute the line

V variablename - To see the contents of a variable

Q or QUIT- Quit out of the subroutine and return to the Database prompt.

Press Ctrl + C to abort the subroutine.

Break: Option (A,C,L,Q,D,?) - Choose A. Will return to the Database prompt

Example 3

Modify example 1 to display the id, 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

As you would be aware by now that we need to use OPF to open any file in T24

```
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 T24

subroutine EB.READLIST. Please note that a Select statement can only return the ids from the file on which the Select statement is based.

EB.READLIST

EB.READLIST is a core T24 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 array will only hold the ids of the records that have been selected. All the ids are 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.

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

```
SEL.CMD = “SELECT  “:FN.CUS
CALL EB.READLIST(SEL.CMD,SEL.LIST,’’,NO.OF.REC,CUS.ERR)
```

Insight Into EB.READLIST

EB.READLIST behaves differently based on the type of SELECT statement supplied to it.

1. If a SELECT with conditions or a SSELECT(sorted select) is passed, then EB.READLIST
    a. Spawns a new process and does an ‘EXECUTE’ of the SELECT statement passed to it.
    b. Extracts the IDs and returns it to the user.

    This method is called ‘Executing A Select’

2. If a plain SELECT (with no conditions and not a sorted select) is passed, then EB.READLIST
    a. Does an OPF of the file whose records you require
    b. Makes the file pointer (F.) point to the beginning of the file
    c. Extracts the IDs one after the other returns the list of IDs to the user.

    This method is called ‘Executing An Internal Select’

The second method is faster when compared to the first one, as there is no select that is actually executed on the file.

Use the second method when you wish to process all or most of the records in a file. If you wish to process only a few records in a file, then the first method is ideal.

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.US)
      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 T24. In order to see the execution of the subroutine line by line, we need to add the DEBUG statement.

Example 4

Modify Example 3 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 T24. In order to see the execution of the subroutine line by line, we need to add the DEBUG statement.

Example 5

As a part of the COB process in T24, all savings accounts that have balance less than 5000 need to be charged a fee. For this purpose you are expected to create a local reference field by name CHARGE in the Account application and write a subroutine that will check the working balance of all savings accounts, and if the working balance is lesser than 5000 then set the value in the local reference field CHARGE to 'Y'. As a part of the COB process in T24, one of the COB routines will deduct a charge from all accounts which have this field set to 'Y'.

Solution 5

Algorithm

Step 1 . Create the local reference field CHARGE using the LOCAL.TABLE application and attach it to the ACCOUNT application using the LOCAL.REF.TABLE application.

Step 2. Open the ACCOUNT file

Step 3.Select all accounts with category = 6001(this category might differ from one T24 installation to another).

Step 4.For each of the accounts selected, read the corresponding record from the ACCOUNT file

Step 5.Check the working balance of the account and if the balance is less than 5000 then write the entire ACCOUNT record into the ACCOUNT file with the local reference field CHARGE set to 'Y'

Step 6.Update the F.JOURNAL file.

Repeat steps 3 to 5 for all accounts using the LOOP and REPEAT statements.

Step 1

Create the local reference filed CHARGE in the ACCOUNT application using the LOCAL.TABLE and the LOCAL.REF.TABLE applications.

Steps 2 And 3

As discussed earlier use OPF and EB.READLIST.

Step 4

Now we need to read the record from the ACCOUNT file. Normally we would use the F.READ statement to read a record from a file. But we need to understand that , when we read a record, we only obtain a shared lock on the record and hence multiple people can read the record simultaneously. Since the record that we are to read by might updated by us, we need to ensure that we have exclusive control over the record. Hence we need to use F.READU instead of F.READ.

F.READU

Parameters

FILEID file name

KEY record-id

REC record returned

F.FILEID file variable

ER returning error message

RETRY P msg - prompt user with msg to retry if record locked

R nn xx - retry xx times with a nn seconds sleep interval

I - Ignore the lock and return

E - Return immediately with an error message

‘’ (null) - Retry continuously

CALL F.READU(FN.ACC,Y.AC.ID,R.ACCOUNT,F.ACC,Y.ACC.ERR,RETRY)

The simple rule is, when you want to modify a record, use F.READU and obtain the record.

Step 5

Once the working balance has been extracted and is found to be lesser then 5000, now the local reference field CHARGE needs to be set to ‘Y’ and the entire account record needs to be written on to the ACCOUNT file.

To update a local reference field

R.ACCOUNT<AC.LOCAL.REF,1> = ‘Y’.

Note that there is just one physical field called LOCAL.REF in most of the applications in T24. By using LOCAL.TABLE and the LOCAL.REF.TABLE applications, we are just multi-valuing the field LOCAL.REF and giving the new multi value field a new name. Therefore, once a local reference field is created, it would not affect the physical layout of the file(would not affect the dict) but will only affect the STANDARD.SELECTION. All local reference fields will have an entry in the SS application with the Usr Type set to ‘I’ – I descriptor.

Use the T24 subroutine F.WRITE to write on to a file.

F.WRITE

F.WRITE is a core T24 subroutine that is used to write a record on to a file.

The routine works differently depending on mode of T24.

- If the system is in the Online mode, then F.WRITE will only write the data on to the cache.
- If the system is in the Batch mode (COB is in progress) then it will straight write to the disk and not to the cache provided the F.WRITE is not within a transaction block.

When the system is in the Online mode, as mentioned earlier, the write will only happen to the cache. It is only when a subsequent call to JOURNAL.UPDATE is encountered, the data will be flushed to the disk.

JOURNAL.UPDATE is a core T24 routine that takes care of updating the F.JOURNAL file and also ensures transaction management. Please note that JOURNA.UPADTE, if called when the system is in Batch mode, will not update the JOURNAL fie.

Transaction management can be explicitly triggered in T24 using the EB.TRANS routine.

CALL EB.TRANS(“START”, ”Start of transaction block”)

Statement 1
Statement 2
Statement 3
CALL EB.TRANS("END", 'End of transaction block")

If it any point within a transaction block the transaction needs to aborted, the T24 subroutine TRANSACTION.ABORT can be called.

CALL TRANSACTION.ABORT

The JOURNAL.UPDATE routine in T24, has the EB.TRANS embedded in it.

Syntax

F.WRITE(<filename>,<id of record>,<dynamic record variable>)

Example

CALL F.WRITE(FN.ACC,Y.AC.ID,R.ACCOUNT)

Step 6

Use the T24 subroutine JOURNAL.UPDATE to update the F.JOURNAL file.

CALL JOURNAL.UPDATE(Y.AC.ID)

Note

While writing subroutines that would get executed during the Online stage of T24, a call to F.WRITE will fail if it does not find a subsequent call to JOURNAL.UPDATE. During the batch stage, since the F.JOURNAL file is not maintained, a call to F.WRITE does not require a subsequent call to JOURNAL.UPDATE.

Consolidated Solution 5

```
*Subroutine to check if the balance in all savings accounts are less than
*5000 and if so, update a local reference field in the ACCOUNT file called
*CHARGE to 'Y'.

      SUBROUTINE  ACC.BAL.CHECK
      $INSERT I_COMMON
      $INSERT I_EQUATE
      $INSERT I_F.ACCOUNT
      GOSUB INIT
      GOSUB OPENFILES
      GOSUB PROCESS

      INIT:
      FN.ACC = 'F.ACCOUNT'
      F.ACC = ''
```

```
        Y.AC.ID = ''
        R.ACCOUNT = ''
        Y.ACC.ERR = ''
        RETRY = ''
        RETURN

        OPENFILES:
        CALL OPF(FN.ACC,F.ACC)
        RETURN

        PROCESS:
        SEL.CMD = "SELECT ":FN.ACC:" WITH CATEGORY = 6001"
        CALL EB.READLIST(SEL.CMD,SEL.LIST,'',NO.OF.REC,RET,CODE)
        LOOP
            REMOVE Y.ACC.ID FROM SEL.LIST SETTING POS
            WHILE Y.ACC.ID:POS
                    CALL
F.READU(FN.ACC,Y.AC.ID,R.ACCOUNT,F.ACC,Y.ACC.ERR,RETRY)
                    IF R.ACCOUNT<AC.WORKING.BALANCE> < 5000 THEN
                                R.ACCOUNT<AC.LOCAL.REF,1> = 'Y'
                                CALL F.WRITE(FN.ACC,Y.ACC.ID,R.ACCOUNT)
                            END
        Y.AC.ID = ''
        R.ACCOUNT = ''
        REPEAT
              CALL JOURNAL.UPDATE(Y.ACC.ID)
        RETURN
        END
```

In the above example, F.WRITE is called within a loop and JOURNAL.UPDATE is called outside the loop. In this case the system is in Online mode and hence all the writes triggered by F.WRITE would happen only to the cache. It is only when the control comes out of the loop the data from the cache will actually get flushed to the disk. Care should be taken when calling F.WRITE within a loop – it might lead to using up all the cache.

An Insight Into Reading And Writing In T24

There are a number of ways using which data can be retrieved from T24. We need to choose the right method of extracting data to achieve performance.

As we have already discussed, F.READ and F.READU can be used to read data from files in T24. It is very important for us to understand how data is read by these routines and the best way to read data so that we get the optimum performance.

Working of F.READ

1. Obtain the file name and the record id
2. Perform SMS validations
3. Check if the record is in cache. If it is in cache, will fetch it
4. If it is not in cache, read from the file in the disk and load the record into cache

Working of F.READU

1. Obtain the file name and record id
2. Perform SMS validations
3. Check if the record is in cache and if so, check if it is locked. If not locked, go to the disk and lock it
4. If the record is not in cache, read and lock the record from the disk

F.READU holds info of all record locks so that they can be released during a transaction abort. Whenever we lock a record using F.READU, the lock gets released when

a. That record is written back to the file
b. When the program terminates
c. When an explicit RELEASE statement is encountered

In T24, in many routines, even if there is no explicit RELEASE statement, the locks get releases. This is because of the RELEASE statement in JOURNAL.UPDATE.

Working of F.READV

Similar to F.READU, but fetches only one field of the record unlike F.READ that fetches the whole record.

READ: Extract data from a file. F.READ and F.READV call READ internally to extract data.

```
READ <Record> FROM <File Name>, <ID> ELSE <Message>
READ REC FROM F.FL, ID ELSE GOSUB “Record Not Found”
```

READU: Extract data from a file and lock it. F.READU internally calls this to extract data.

```
READU <Record> FROM <File Name>, <ID> ELSE <Message>
READU REC FROM F.FL, ID ELSE GOSUB CRT “Record Not Found”
```

Please note, when we use READ or READU, we need to specify the actual name of the file that we are to open, as OPF is not called to open the file.

The jBASE command OPEN can be used to open files.

OPEN: Open files to file variables.

```
OPEN <File Name> TO <File Variable> ELSE ABORT
OPEN “FBNK.CUSTOMER” TO F.FL ELSE ABORT
```

Working of DBR (T24 subroutine)

The DBR routine can be used to extract the value of a single field from a record. This internally does a READ and obtains the record. From the record, it fetches the required field’s value and returns it to the user. Please note that the file from which the data needs to be retrieved need not be explicitly opened for DBR as DBR would open the file internally.

Incoming Parameters:

Field 1 = File Name (without 'F.')
Field 2 = Name of the filed as in the insert file whose value needs to be fetched
Field 3 = miscellaneous arguments (delimiter = '.')
a) ""
Or 'L' = Take value field in accordance to language
Or 'F' = Take full field
b) Not used
c) Not used
d) Delimiter argument
e) Ignore error

Outgoing Parameters:

Value of the field that has been fetched

```
CALL DBR(<FileName:FM:FldToBeFetched:FM:”LangSpecFld”>,<ID>,<RetVar>)
CALL DBR(“CUSTOMER”:FM:EB.CUS.SHORT.NAME:FM:”L”:,100069,Y.RET.VAL)
```

All the above routines are core T24 routines that can be used to extract data from T24. All these routines in-turn have to call Infobasic commands to actually obtain data. Following are the various Infobasic commands that can be used to extract data.

TRANS

This is a jBASE command that is used to fetch the value of a particular field from a file. This is similar to DBR, just that this is a jBASE function and hence is faster than DBR. The file t be accessed using TRANS need not be explicitly opened as TRANS would do it internally.

```
<ExtractedValue> = TRANS(<FileName>,<ID>,<FieldName>,”X”)
FileName : Actual name of the file to be opened
X: Default action code to return the value of the field specified.
```

```
Y.SHORT.NAME = TRANS(“FBNK.CUSTOMER”,100069,EB.CUS.SHORT.NAME,”X”)
```

CACHE.READ

CACHE.READ is a routine that is used to read a record from a file in T24. CACHE.READ will

- Check if the record required is in cache
- Is yes, check if it is not older than the number of seconds specified in the field CACHE.EXPIRY in SPF. If not older, extract the record and give it to the user
- Else will perform an OPF and F.READ to extract the record
- Load the extracted record on to cache
- Return the record to the user.

CACHE.READ

- Is faster than F.READ, we do not have to explicitly open the file using OPF. CACHE.READ, if not in cache, will perform OPF and extract the record for us.
- Should be used for extracting records that are not frequently updated
- Best used for extracting parameter records

```
CACHE.READ(FileName,ID,Record,Error)
```

FILENAME = Name of file - without the mnemonic (Example: F.CUSTOMER)

ID = Valid values are

ID of a record

'SelectIDs' (List of Ids from the 'FileName')

'SSelectIDs' (List of sorted Ids)

'SSelectARs' (List of sorted Ids in ascending order right justified)

Record = Data returned

Error = RECORD NOT FOUND for example

CACHE.READ is not used to extract a record from a file, but also for executing SELECT statements.

If the string 'SelectIDs' is passed to CACHE.READ, it will in turn call EB.READLIST which will execute an 'Internal' select.

If 'SSelectIDs' or 'SselectARs' is passed to CACHE.READ, it will in turn call EB.READLIST that will execute the select statement supplied to it.

Important T24 Routines

OVERLAY.EX

This is the routine that is called when a user signs on into T24. It clears and loads all the common variables, opens common files etc.

```
CALL OVERLAY.EX
```

FATAL.ERROR

Routine to display system errors, log them to the protocol file and exit. The text that needs to be displayed as a fatal error needs to be set in the common variable TEXT. The FATAL.ERROR routine takes in one parameter. This parameter can hold any value, but is usually set to the name of the file on which the fatal error occurred.

```
TEXT = "Customer File Not Accessible"
CALL FATAL.ERROR("CUSTOMER")
```

The above command will result in updating a record in the PROTOCOL file with the ID : <UserName> . This user name is the name of the user for whom the fatal error was generated.

Writing Data

We have already learnt that F.WRITE is used to write data on to T24. F.WRITE internally calls WRITE to actually write.

WRITE

```
WRITE <Record> ON <Filename>,<ID> ON ERROR <Message>
```

Please note that when a WRITE is executed on a record that has been locked, the WRITE statement will release the lock.

Accessing Sequential Files

All this while we have been writing subroutines that read from and write into hashed files. As you would be aware by now, all data files in T24 are hashed files and are of type J4. WE could also read and write into sequential files (non hashed files). These sequential files are mainly used to send data to third party systems from T24 or receive data from third party systems in T24. There are separate sets of commands that need to be used to access these sequential files. This section deals with reading and writing into such types of files. Let us understand these with an example.

Example 6

Write a program that will write a string "Infobasic programming" onto a sequential file.

Algorithm

Step1 . Create a non-hashed file with the name "TEMENOS.SEQ"

CREATE.FILE TEMENOS.SEQ TYPE=UD

Step 2. Open the non-hashed file

Store the name of the file on to a variable.

All non-hashed files also have records and hence every record needs to have a record id.

SEQ.FILE.NAME = 'TEMENOS.SEQ'

RECORD.NAME = '1'

Use the command OPENSEQ to open a sequential file.

OPENSEQ

OPENSEQ is used to open a non-hashed file. While using OPENSEQ to open non-hashed file, the record id needs to be supplied as one of the parameters. OPENSEQ will open that record to a pointer.

Syntax

OPENSEQ filename,recordid to pointer

Example

OPENSEQ SEQ.FILE.NAME,RECORD.NAME TO SEQ.FILE.POINTER ELSE

*The else condition in the OPENSEQ statement will be met when there is no record with the *id mentioned in the variable RECIORD.NAME and the following CREATE statement *would create the pointer in order for us to create a new record.

CREATE SEQ.FILE.POINTER ELSE

The else clause in the CREATE statement would be met when the file name specified in the variable SEQ.FILE.NAME is an invalid file name. When the file is invalid, there no other option other than to display an error message and abort.

Step 3

Write the message "Infobasic Programming" on to the file.

Use the command WRITESEQ to write the message on to the file.

WRITESEQ

WRITESEQ is the command that is used to write the data on to a non hashed(sequential file)

Syntax

WRITESEQ Message TO FilePointer ELSE ErrorMessage

The else clause in the WRITESEQ would be met when the file pointer has not been initialized properly (does not point to a record in a file)

Example

WRITESEQ "Infobasic programming" TO SEQ.FILE.POINTER ELSE

CRT "Unable to perform WRITESEQ"

Step 4

Close the file once the operations are complete .Use the CLOSESEQ command to close a sequential file.

CLOSESEQ

The CLOSESEQ command is used to close a sequential file after all read write operations for that file are complete.

Syntax

CLOSESEQ FilePointer

Example

CLOSESEQ SEQ.FILE.POINTER

Consolidated Solution 6

```
      PROGRAM SEQFILE.ACCESS.WRITE
      SEQ.FILE.NAME = 'TEMENOS.SEQ'
      RECORD.NAME = '1'
      OPENSEQ SEQ.FILE.NAME,RECORD.NAME TO SEQ.FILE.POINTER ELSE
            CREATE SEQ.FILE.POINTER ELSE
                  CRT   "Unable   to   create   file   pointer   to   file
":SEQ.FILE.NAME
                  STOP
            END
      END
      CRT "Openseq was successful on file ":SEQ.FILE.NAME
      WRITESEQ "Infobasic programming" TO SEQ.FILE.POINTER ELSE
      CRT  "Unable to perform WRITESEQ"
      CLOSESEQ SEQ.FILE.POINTER

      END
```

Example 7

Write a program that will read the data that has been written on to the sequential file TEMENOS.SEQ .

Algorithm

Step 1 : Open the sequential file

Step 2 : Read the data from the sequential file and display it

Step 3 : Close the sequential file

Step 1 and 3

As discussed earlier, use the OPENSEQ command to open the sequential file and the command CLOSESEQ to close a sequential file.

Step 2

To read the data from the Sequential file, use the command READSEQ

READSEQ

READSEQ is the command that is used to read the data from a sequential file.

Syntax

READSEQ variablename FROM filepointer THEN …… ELSE…..

Variablename : Name of the variable that will hold the data that has been retrieved from the file.

THEN : The THEN clause will be met when the read is successful.

ELSE : The ELSE clause would be met when the read is unsuccessful. The read would fail if the file pointer is not initialized properly.

Example

```
READSEQ Y.MSG FROM SEQ.FILE.POINTER THEN
    CRT “Message Extracted :”:Y.MSG
END
ELSE
      CRT “Unable to read from file “
END
```

Consolidated Solution 7

```
PROGRAM SEQFILE.ACCESS.READ
SEQ.FILE.NAME = 'TEMENOS.SEQ'
RECORD.NAME = '1'
OPENSEQ SEQ.FILE.NAME,RECORD.NAME TO SEQ.FILE.POINTER ELSE
      CREATE SEQ.FILE.POINTER ELSE
        CRT "Unable to create file pointer to file ":SEQ.FILE.NAME
            STOP
      END
END
CRT "Openseq was successful on file ":SEQ.FILE.NAME
READSEQ Y.MSG FROM SEQ.FILE.POINTER THEN
      CRT "Message Extracted :":Y.MSG
END
ELSE
      CRT "Unable to read from file "
END
CLOSESEQ SEQ.FILE.POINTER
END
```

Note

1. When a read and write operations are performed on a sequential file in the same program, after WRITESEQ is performed, to use the same file pointer again for a subsequent READSEQ, the file pointer must be initialized, meaning, do a CLOSESEQ and an OPENSEQ. The same rule applies when a WRITESEQ is to follow READSEQ in the same program.
2. An OPENSEQ command can also be used to open a file provided along with the path of the file and the id.

```
SEQ.FILE.NAME = './TEMENOS.SEQ/1'
OPENSEQ SEQ.FILE.NAME TO SEQ.FILE.POINTER
```

   All the other WRITESEQ, READSEQ, CLOSESEQ statements will remain the same.

Creating Subroutines With Arguments

All this while we have been learning about subroutines that just display the value on the screen or store the value on to an array. This section will now introduce you to the process of creating subroutines that can take in arguments or parameters and return values as well. Let us understand it with a simple example.

Example 8

Step 1

Create a subroutine that will accept 2 integer values, multiply them and return the result in a variable.

```
SUBROUTINE DEMO.CALLED.RTN(ARG.1,ARG.2,ARG,3)
ARG.3 = ARG.1 * ARG.2
RETURN
END
```

While defining subroutines in Infobasic, we cannot specify which are the incoming and which are the return parameters. Therefore just specify the arguments one after the other along with the subroutine definition as done above. Since the subroutine is storing the result in the variable ARG.3, the system would understand that ARG.3 is the return parameter.

Step 2

Create another subroutine that would supply the values and call the DEMO.CALLELD.RTN.

```
SUBROUTINE DEMO.CALLING.RTN
VAR.1 = 10
VAR.2 = 20
VAR.3 = ‘’
CALL DEMO.CALLED.RTN(VAR.1,VAR.2,VAR.3)
PRINT ‘Result “:VAR.3
RETURN
END
```

Defining Functions In Infobasic

Functions can also be defined in Infobasic. These functions can take in any number of parameters but return only one value. Let us understand it with a simple example.

Example 9

Step 1

Create a function that will accept 2 integer values, multiply them and return the result in a variable.

```
FUNCTION DEMO.FUNCTION(ARG.1,ARG.2)
```

```
RET.VALUE = ARG.1 * ARG.2
RETURN(RET.VALUE)
END
```

The RETURN statement is used to return a value in a function.

Step 2

Now the function defined above can be called from any program/subroutine. Now create a subroutine that would supply values and call the above-defined function. When calling a function, the function needs to be defined first using the DEFFUN command and only then the function should be called.

```
SUBROUTINE DEMO.SUB.CALLING.RTN
VAR.1 = 10
VAR.2 = 20
DEFFUN DEMO.FUNCTION(VAL.1,VAL.2)
VAR.3 = DEMO.FUNCTION(VAR.1,VAR.2)
CRT "Result :":VAR.3
RETURN
END
```

Incase of a function, only the incoming parameters need to be defined. There can and will be only one return value and that will directly get assigned to the variable on the right side of the equation as defined above.

Un-initialized variables VAL.1 and VAL.2 have been given to demonstrate that any variable can be given at the time of defining a function.

Summary

- Infobasic does not support data types. Variables need not be declared in Infobasic.
- All variables in Infobasic are treated as dynamic arrays
- Dynamic arrays expand or reduce in size depending on the amount of data
- Dimensioned arrays need to be declared using the DIM statement
- FM, VM and SM are delimiters to separate fields, multi values in a field and sub values in a field respectively
- JBCDEV_LIB is the variable that holds the path where the object code of subroutines need to be stored
- JBCOBJECTLIST contains the search path of subroutines
- JBCDEV_BIN contains the path where the executables of programs need to be stored

- PATH contains the search path of jBASE executables in addition to the search path of Unix executables.
- globusbin contains all the core T24 executables
- globuslib contains the object code of all core T24 subroutines
- lib and bin are used to store object codes and executables of local subroutines and programs respectively.
- jLibDefinition is the jBASE configuration file that controls the creation of library files in jBASE.
- OPF is a T24 routine that is used to open files
- F.READ is a T24 routine that is used to read a record from a file
- EB.READLIST is a T24 routine that is used to execute a select statement
- EB.COMPILE is used to compile and catalog subroutines
- To append data into an array with FM as delimiter use Arrayname<-1>
- OPENSEQ is the command that is used to open a sequential file
- READSEQ is the command that is used to read a sequential file
- WRITESEQ is the command that is used to write data into a sequential file
- CLOSESEQ is the command that is used to close a sequential file
- When and read and a write operation on a sequential file are performed in the same program, then the file needs to be closed and opened again before it can be read from or written into.
- Functions can take in any number of arguments but can return only 1 value.

Additional Information

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

F.DELETE

F.DELETE is also a core T24 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)
```

Infobasic Commands

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.

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 TEM1,101 ELSE
      MAT ARRAY1 = ‘’
END
MATWRITE ARRAY1 ON TEM1,100
```

CONVERT

Use the CONVERT command to convert characters to other characters.

```
Y.STRING = “TEMENOSFMT24”
CONVERT FM TO VM IN Y.STRING
CRT Y.STRING
TEMENOSVMT24
```

DELETE

Use the DELETE command to delete a record. The F.DELETE T24 routine uses this statement to delete a record.

```
DELETE <FileName>,<ID>
```