FUNCTION Statement

Steve White

Sunbelt Computer Systems

OVERVIEW

- Requests
- Solution
- Implementation
- Examples

Requests

- Variables local to a subroutine
- Reusable variables
- “Stack” type variables so a subroutine could be called by itself or a subroutine it had called
- Variables automatically initialized to a known state
- Return a variable to the calling program

Solution

- New Program structure called:

  FUNCTION / FUNCTIONEND
  LFUNCTION / FUNCTIONEND

- Uses new ENTRY statement
- Implements stack type variables
- Called like ROUTINE
- Referenced using EXTERNAL

Solution

- Each FUNCTION local variable starts out each call in it’s initial defined state.
- No more left over values from previous calls
- No more having to pre-initialize variables to a known state
- Can be called recursively
- Can return a variable to the calling routine

Implementation

- Syntax

```
{label} FUNCTION
[vara   {datatype} {info}]
        …
        ENTRY
[lvara  {datatype} {info}]
        …
        [RETURN   [using {varx}]]
        FUNCTIONEND [using {varx}]
```

Implementation

- Simplest construct with no arguments

```
{label}   FUNCTION
          ...
          ENTRY
          ...
          FUNCTIONEND
```

Implementation

- FUNCTION with a single argument

```
FUNC1       FUNCTION
PARM1       DIM          10
            ENTRY
...
            RETURN
            FUNCTIONEND
            ...
            CALL         FUNC1 USING DIMFIELD
```

Implementation

- This form not much different from a ROUTINE or LROUTINE structure with several exceptions.
  - Program cannot fall into FUNCTION where it can with a ROUTINE.
  - PARM1 is NOT destroyed if FUNC1 is called recursively! Upon return it remembers it's proper value!

Implementation

- FUNCTION with local variables

```
FUNC1     FUNCTION
PARM1     DIM          10
          ENTRY
LOCALVAR  FORM         5.2
FILE      FILE
          ...
          RETURN
          FUNCTIONEND
          ...
          CALL         FUNC1 USING DIMFIELD
```

Implementation

- This example has two local variables.
  - The FORM field is set to the initial value of ‘0’.
  - The FILE variable is always a closed file on entry, as it is closed when exiting the FUNCTION.

Implementation

- FUNCTION with return variable

```
COUNT     FORM        5
.----------------------------------
FUNC1     FUNCTION
PARM1     DIM         10
          ENTRY
LOCALVAR  FORM        5.2
          ...
          FUNCTIONEND USING LOCALVAR
.----------------------------------
          CALL        FUNC1 GIVING COUNT:
                            USING DIMFIELD
```

Implementation

- Can return a single parameter
  - Can be any variable type or object
  - If returning an OBJECT, then the returned-to OBJECT is destroyed first
  - If returning a xFILE, then the returned-to xFILE is closed first ???

Implementation

- Certain verbs are not allowed in FUNCTION
  - CALL to label in FUNCTION
  - GOTO to label outside of FUNCTION
  - NORETURN
  - TRAP (use EXCEPTION instead)
  - ROLLOUT
  - ROUTINE / LROUTINE

Implementation

- Certain verbs are not allowed in FUNCTION
  - BRANCH / BRANCHF to a label outside of FUNCTION
  - PERFORM / PERFORMF to a label in FUNCTION

Implementation

- Local Variables
  - Defined after ENTRY statement
  - Initialized upon entry to FUNCTION each time FUNCTION is called
  - Can be used as sending arguments on CALL statement
  - Can be used to receive value from another FUNCTION call
  - Must be defined before first executable statement

Implementation

- Local Variables
  - Any local variable created during FUNCTION is automatically destroyed at FUNCTIONEND or RETURN statement
  - Any local file is closed on exit
  - Any local object is destroyed on exit

Sample Program

- Sample Program
- Execute Program

QUESTIONS?

That's All!!