Subroutines and Gosubs

by Alyce Watson

Table of Contents

What is a Subroutine?

A subroutine is also called a subprogram. It is a self-contained section of code that can be called from other parts of the program. It can be called many times. If code that is to be reused many times is placed in a subroutine, the program has fewer lines of code. It is easier to read and to modify. Liberty BASIC has two kinds of subroutines, the GOSUB and the SUB.

Gosubs

A subroutine that is called by *GOSUB [branchLabel]* is part of the main program and shares its memory space. It begins with a branch label and ends with the keyword *RETURN*. When the RETURN keyword is encountered, the program execution returns to the next line after the GOSUB command that called the routine. Here is an example:

```
Print "Starting program!"
gosub [printHello]
print "Ending program!"
END

[printHello]
print "Hello, you are in a gosub!"
RETURN
```

GOSUB Rules

- A subroutine called by GOSUB must begin with a branch label and end in RETURN.
- The program must not fall into a GOSUB subroutine. The subroutine must be called explicitly. A WAIT, END, or GOTO statement must be used in the code before the branch label that begins the subroutine. The WAIT, END, or GOTO statement stops the main program from flowing into the subroutine.

```
'WRONG!
[begin]
print "Hello World"

[myGosub]
print "I've visited a gosub!"
RETURN
'
'Right:
print "Hello World"
WAIT

[myGosub]
print "I've visited a gosub!"
RETURN
```

- The code must not attempt to exit the GOSUB subroutine with a GOTO [branchLabel]
- It is not possible to pass arguments (values) into a GOSUB subroutine.
- GOSUB subroutines can "see" all other variables and branch labels in the main program.
- GOSUB subroutines cannot "see" variables and branch labels in called subroutines or functions.
- GOSUB subroutines can call other GOSUB subroutines, called subroutines, or functions.

Called Subroutines

Called subroutines (subs) differ in many respects from GOSUB subroutines. The general form of a called subroutine is as follows:

```
Sub SubName arg1, arg2,... argN
    'some code here
End Sub
```

The sub definition is signalled by the keyword *SUB*. It is followed by the sub's name, then a list of arguments, separated by commas. The argument list must not be placed inside of parentheses. The next lines contain whatever code routine is performed by the sub. The sub definition must end with *END SUB*.

The Argument List

The arguments sent to a called SUB may be string, numeric, or a mixture of both. They may be literal values, variables, or expressions. The arguments must be separated by commas. These arguments are actually variables that are local to the SUB. They can be used in code contained in the SUB. When the SUB ends, the variables no longer have any value. A SUB can exist without arguments, in which case the SUB name stands alone in the SUB statement line of code.

The names given to the argument list in the sub definition represent variables that are local to the SUB. When the SUB is called, whatever literal value, expression or variable is contained within the commas will be evaluated and given the corresponding name in the argument list. See the following examples for a demonstration of the argument list. Notice in the last example the variable names used to call the SUB have exactly the same names used in the SUB, but in a different order. (They are name1$ and name2$.) This doesn't matter, because the argument variables are local to the sub and are not the same variables as the ones with the exact same names used in the main program.

```
call LongestName "Sam","Patricia"

first$="Carl"
last$="Gundel"
call LongestName last$,first$

name1$="Liberty"
name2$="BASIC"
call LongestName name2$,name1$
end
```

```
Sub LongestName name1$,name2$
    if len(name1$)>len(name2$) then
        print name1$ + " is the longest."
    else
        print name2$ + " is the longest."
    end if
End Sub
```

Results in:

Patricia is the longest.

Gundel is the longest.

Liberty is the longest.

Beginning with Liberty BASIC 4, arguments can be passed ByRef as well as by value. See:
Passing Arguments into Subroutines and Functions - by value and BYREF

Scope

Each SUB has its own memory space that is separate from the memory of the main portion of the program. Variables and branch labels that exist inside of the SUB do not exist in the main program. The main program cannot "see" them. Likewise, SUBs cannot "see" variables and branch labels that exist in the main program, or in other SUBs, or functions. The part of the program that can "see" a variable is called its "scope."

Some entities have global scope. That is, they can be seen both inside of SUBs and in the main program. These things are global in scope: arrays, handles (window and control handles, file handles, DLL handles, communications ports), and structs. In Liberty BASIC 4 variables can be declared as GLOBAL, which means that they can be seen from all parts of a program.

Special global status is given to certain default variables used for sizing, positioning, and coloring windows and controls. These include variables WindowWidth, WindowHeight, UpperLeftX, UpperLeftY, ForegroundColor$, BackgroundColor$, ListboxColor$, TextboxColor$, ComboboxColor$, TexteditorColor$. The values of these variables can be seen in any SUB.

Calling SUBs

SUBs are called by using the keyword *CALL* followed by the name of the sub, then a comma separated list of arguments. When the SUB exits, program execution returns to the line after the CALL command.

SUB Rules

1. SUBs don't need input arguments.
2. SUB names are case sensitive. mysub does not refer to the same routine as MySub.
3. SUBs may contain code that calls Liberty BASIC functions or other user-defined functions or subs.

SUBs as Event Handlers

Liberty BASIC 4 allows us to use Subs as event handlers. In previous versions of Liberty BASIC, branch labels were used as event handlers. When a Sub is used as an event handler, the argument list contains things like the handle of the control or window that fired the event, the MouseX and MouseY values, and the value of Inkey$. See the article by Janet Terra called Coding with Sub Event Handlers. Here is a small example of a button event handler Sub.

```
nomainwin
button #main.exit, "Exit", exitClicked, UL, 10, 10
open "Button Example" for window as #main

[loop]
    wait

sub exitClicked buttonhandle$
    notice "The button handle is ";buttonhandle$;"  Goodbye."
    close #main
    end
    end sub
```

Subroutines in Action

Here is a demo that changes the case of all words in a string so that the first letter is uppercase and the remaining letters are lowercase. Run the little demo to see how it works. Notice that the code inside the SUB uses several native Liberty BASIC string functions.

```
call NameCase "hello world"
call NameCase "CARL GUNDEL"
call NameCase "tiTaNiC"
end
```

```
Sub NameCase name$
    nm$=word$(name$,1)
    while nm$<>""
        i = i + 1
        nm$=word$(name$,i)
        first$=upper$(mid$(nm$,1,1))
        rest$=lower$(mid$(nm$,2))
        new$=new$+" "+first$+rest$
    wend
    print "Name is now ";trim$(new$)
    End Sub
```

Here is a simple numeric SUB that prints the average of three values.

```
call Average 15,88,20
call Average 10,20,30
end
Sub Average num1,num2,num3
    print "Average is ";(num1+num2+num3)/3
    End Sub
```

Here is the same routine, but this one uses a GOSUB subroutine. GOSUB subroutines do not accept input arguments, but they can "see" values in the main program, because they are actually part of the main program and share its memory space. Note the differences between the two methods:

```
num1=15
num2=88
num3=20
gosub [Average]

num1=10
num2=20
num3=30
gosub [Average]

end

[Average]
    print "Average is ";(num1+num2+num3)/3
    RETURN
```

GOSUB or SUB?

Use a GOSUB routine when:

- The routine must be able to see variables and branch labels in the main program.
- There is no need for input arguments.

Use a called SUB when:

- The routine requires input arguments.
- The routine requires variables and/or branch labels that cannot be viewed by the main program.
- In Liberty BASIC 4, the routine is to be used as an event handler.