# Chapter 8 - Introduction to Functions and Subroutines

Functions and Subroutines are also known as procedures. Procedures define a block of code which can be executed with a single statement (ie a function call). There are several reasons to use procedures

* It reduces redundancy.
* They enable the reuse of code.
* It improves readability of the program.
* Improves maintainability of the program
* Makes it easy to extend your program

## Procedures

A procedure is a block of code which may be called at any time from a program. This code may need to be executed multiple times, and procedures provide an invaluable means to simplify code by replacing these blocks of code with a single call. A procedure also serves to allow a user to extend the FreeBASIC language to provide custom commands. Many of the functions built into FreeBASIC are merely subroutines or functions that are part of a "runtime library" linked to by default.

Subroutines and functions

When it comes to defining a procedure, subroutines and functions follow the same pattern.

The Sub keyword marks the beginning of a subroutine, and its end is marked by End Sub. The "name" parameter is the name by which this subroutine is called. For instance, if the declaration is "Sub...End Sub", the user can execute the code in between "Sub foo" and "End Sub" by using "foo" as a statement. This code is executed separate from the code which calls the subroutine, so any variable names, unless they are shared, are not available to the subroutine. Values can, however, be passed using parameters.

When calling a subroutine, parentheses after the subroutine name (surrounding the argument list if any) are optional.

Parameters:

Identifier

the name of the subroutine

## calling convention

In procedure declarations, Cdecl specifies that a procedure will use the Cdecl calling convention. In the Cdecl calling convention, any parameters are to be passed (pushed onto the stack) in the reverse order in which they are listed, that is, from right to left. The procedures need not preserve the EAX, ECX or EDX registers, and must not clean up the stack (pop any parameters) before it returns - that is left to the calling code.

Cdecl is allowed to be used with variadic procedure declarations (those with the last parameter listed as "...").

Cdecl is the default calling convention on Linux, the \*BSDs, and DOS, unless another calling convention is explicitly specified or implied by one of the EXTERN blocks. Cdecl is typically the default calling convention for C compilers, and it's used almost exclusively on Unix-like systems.

In procedure declarations, Pascal specifies that a procedure will use the Pascal calling convention. In the Pascal calling convention, any parameters are to be passed (pushed onto the stack) in the same order in which they are listed, that is, from left to right. The procedures need not preserve the EAX, ECX or EDX registers, and must clean up the stack (pop any parameters) before it returns.

Pascal is not allowed to be used with variadic procedure declarations (those with the last parameter listed as "...").

Pascal is the default calling convention for procedures in Microsoft QuickBASIC, and is the standard convention used in the Windows 3.1 API.

In procedure declarations, Stdcall specifies that a procedure will use the Stdcall calling convention. In the Stdcall calling convention, any parameters are to be passed (pushed onto the stack) in the reverse order in which they are listed, that is, from right to left. The procedures need not preserve the EAX, ECX or EDX registers, and must clean up the stack (pop any parameters) before it returns.

Stdcall is not allowed to be used with variadic procedure declarations (those with the last parameter listed as "...").

Stdcall is the default calling convention on Windows, unless another calling convention is explicitly specified or implied by one of the EXTERN blocks. Stdcall is also the standard (or most common) calling convention used in BASIC languages, and the Windows API.

## Overloading

In procedure declarations, Overload allows procedure names to be overloaded, that is, other procedures (regardless of whether to be subs or functions) can then be declared with the same name if their parameter lists are unique. Two parameter lists are unique if they contain a different number of parameters, or have parameters of different types. Note that this means that two or more procedures cannot be declared with the same name if they differ in return type alone.

A variadic procedure name can never be overloaded.

Once a procedure name has been declared overloaded, further declarations using the name need not specify Overload, but it is allowed.

Overload is not necessary in member procedure declarations, as they are always implicitly overloaded.

When calling an overloaded procedure, the compiler determines the most appropriate definition to use among a set of compatible candidates, by comparing the argument types used to call the procedure with the parameter types specified in the definitions. If no match or an ambiguous match is found, the compiler generates an error at compile time.

External\_identifier

Parameter\_list

A subroutine can also specify how parameters are passed, either as "Byref" or "Byval", as shown in the syntax definition. If a parameter is "Byref", the parameter name literally becomes a reference to the original variable passed to the subroutine. Any changes made to that variable will be reflected outside of the subroutine. If a parameter is passed "Byval", however, the value of any passed variable is copied into a new variable, and any changes made to it will not affect the original.

parameters must also have a supplied type, in the form "parameter as type".

Parameters are the arguments passed to any statement. For instance, if a user executes a statement as "Print 4", the value "4" is passed to the function "Print". Parameters that need to be passed to a subroutine are supplied by one or more parameter arguments in the "Sub" keyword. Creating a subroutine with "Sub mysub(foo, bar)...End Sub", allows the code in between "Sub" and "End Sub" to refer to the first passed argument as "foo" and the second passed argument as "bar". If a parameter is given a default value, that parameter is optional.

[ByRef|ByVal] identifier [As type] [= default\_value]

Return\_type: (functions)

Return is used to return from a procedure .

Return (from procedure) is used inside a procedure to exit the procedure possibly with a return value:

• A Sub cannot specify a return return value. Return is roughly equivalent to the Exit Sub idiom.

• In a Function, Return must specify its return value. Return expression is roughly equivalent to the Function = expression : Exit Function idiom.

Static and Export

## Static and Export

The Static specifier indicates that the values of all local variables defined in the sub should be preserved between calls. To specify individual local variables as static see the Static keyword.

Statements:

Return\_value: (functions)