

## Functions and variables

### Our First User Defined Function

Examine the file SUMSQRES.C (appendix 1) for an example of a C program with functions. Actually this is not the first function we have encountered because the "main" program we have been using all along is technically a function, as is the "printf" function. The "printf" function is a library function that was supplied with your compiler. Notice the executable part of this program. It begins with a line that simply says "Header()", which is the way to call any function. The parentheses are required because the C compiler uses them to determine that it is a function call and not simply a misplaced variable. When the program comes to this line of code, the function named "Header" is called, its statements are executed, and control returns to the statement following this call. Continuing on we come to a "for" loop which will be executed 7 times and which calls another function named "Square" each time through the loop, and finally a function named "Ending" will be called and executed. For the moment ignore the "index" in the parentheses of the call to "Square". We have seen that this program therefore calls a header, 7 square calls, and an ending. Now we need to define the functions.

### Declaring Functions

Before a function can be used by the compiler it must be told of its existence. In PASCAL the 'main' function was always the last function thus allowing the compiler to become aware of the code of the 'user' functions first. In 'C', 'main' is the first function, in order for the compiler to recognise the names of the function called throughout the program these are given in the section above 'main' we have called Function Prototypes. In this section the function definition statement is reproduced but ends with a semicolon (;).

### Defining Functions

Following the main program you will see another program that follows all of the rules set forth so far for a 'main' program except that it is named "void Header(void)". This is the function which is called from within the main program. Each of these statements are executed, and when they are all complete, control returns to the main program. The first statement sets the variable "sum" equal to zero because we will use it to accumulate a sum of squares. Since the variable "sum" is defined as an integer type variable prior to the main program, it is available to be used in any of the following functions. It is called a "global" variable, and its scope or life is the entire program and all functions. More will be said about the scope of variables later. The next statement outputs a message to the monitor. Program control then returns to the main program since there are no additional statements to execute in this function. It should be clear that the two executable lines from this function could be moved to the main program, replacing the Header call, and the program would do exactly the same thing that it does as it is now written. This does not minimize the value of functions, it merely illustrates the operation of this simple function in a simple way. You will find functions to be very valuable in C programming.

## Passing A Value to A Function

In the call to the function "Square", we have an added feature, namely the variable "index" within the parentheses. Note this is the variable used as a counter in the 'for' loop. This is an indication to the compiler that when it calls the function, it takes along the value of index to use in the execution of that function. Looking ahead at the function "Square", we find that another variable name is enclosed in its parentheses, namely the variable "number". This is the name we prefer to call the variable passed to the function when we are in the function. We can call it anything we wish as long as it follows the rules of naming an identifier. The variable does not need declaring within the function as this is done by the function definition statement. With all of that out of the way, we now have the value of index from the main program passed to the function "square", but renamed "number", and available for use within the function. Following the opening brace of the function, we define another variable "numsq" for use only within the function itself, (more about that later) and proceed with the required calculations. We set "numsq" equal to the square of number, then add numsq to the current total stored in "sum". Remember that "sum += numsq" is the same as "sum = sum + numsq" from the last lesson. We print the number and its square, and return to the main program.

## Passing a Value to a Function

When we passed the value of "index" to the function, a little more happened than is obvious. We did not actually pass the value of index to the function, we actually passed a copy of the value. In this way the original value is protected from accidental corruption by a called function. We could have modified the variable "number" in any way we wished in the function "square", and when we returned to the main program, "index" would not have been modified. We thus protect the value of a variable in the main program from being accidentally corrupted, but we cannot return a value to the main program from a function using this technique. Continuing in the main program, we come to the last function call, the call to "ending". This call simply calls the last function which has no local variables defined. It prints out a message with the value of "sum" contained in it to end the program. The program ends by returning to the main program and finding nothing else to do. Compile and run this program and observe the output.

## Returning a Value from a Function

Single values can be returned from a function by indicating the data type to be returned. For example our function 'void Square (int Number)' could be changed to return the value of 'sum' instead of using a global variable.

The function would now look like this:

```
int Square(int Number)    /* This is the square function */
{
    int Numsq, NewSum;

    Numsq = Number * Number; /* This produces the square */
    NewSum += Numsq;
    printf("The square of %d is %d\n",Number,Numsq);

    return (NewSum);
}
```

The function prototype would have to be changed to: `int Square(int Number);` and the function call would also change as shown below. Finally the Ending function would not be needed. The whole program is repeated here with the additions and changes highlighted by a bold comment.

```

/* Function Prototypes */
void Header(void);
int Square(int Number); /* a return type added */
void Ending(void); /* no longer required */

/* Global Variables */
int Sum; /* This is a global variable */ /* no longer required */

/* ===== Main Block ===== */
void main(void)
{
int Index, Total; /* variable added to hold result */

    Header(); /* This calls the function named header */
    for (Index = 1;Index <= 7;Index++)
    {
        Total = Square(Index); /* This calls the square function */
/* and now takes a return value in Total*/
    }
/* Ending(); /* This call no longer need */
    printf("\nThe sum of the squares is %d\n",Total); /* new print line*/
}

/* ===== User Defined Function ===== */
void Header(void) /* This is the function named header */
{
    Sum = 0; /* Initialize the variable "sum" */
    printf("This is the header for the square program\n\n");
}
/* ===== User Defined Function ===== */
int Square(int Number) /* This is the square function with return type */
{
int Numsq;
static int NewSum=0; /* variable to hold result and return it to main */

    Numsq = Number * Number; /* This produces the square */
    NewSum += Numsq; /* uses new local variable instead of global */
    printf("The square of %d is %d\n",Number,Numsq);
    return(NewSum); /* returns value to main and in to Total */
}
/* ===== User Defined Function ===== */
void Ending(void) /* This function no longer needed */
{
    printf("\nThe sum of the squares is %d\n",Sum);
}

```

## Scope of Variables

Simply scope refers to the ‘life’ of a variable. Here we will consider only three types of scope and how it affects the life and initialisation of the variable.

Firstly as used in the sample program of Sumsqres.C there is the global variable of ‘Sum’. Global variables are seen or live throughout the life or execution of the program, retaining their value and space in memory. Any variable declared above ‘main’ is a global variable.

Global variables are automatically initialised to zero.

Variables in our program Sumsqres.c like 'Index' and 'Numsq' are known as local variables and are declared at the top of a block or function. This category or scope of variable only takes up memory storage for the duration that the block or function is run. Local variables are not initialised and must be assigned an initial value by the programmer unless used as a result of a calculation.

The third scope we have in our sample program is a local static variable. In the modified version of the Sumsqres program we have a static integer called NewSum. This allows us to use the same variable when we call the function again and for that variable to retain its value from the last time it was used. We would not want this variable to continually reset to zero each time the function is called in the loop, but as it is local it must be initialised when first used.

## Appendix 1 - SumSqres.c

```
/* Function Prototypes */
void Header(void);
void Square(int Number);
void Ending(void);

/* Global Variables */
int Sum; /* This is a global variable */

/* ===== Main Block ===== */
void main(void)
{
    int Index;

    Header(); /* This calls the function named header */
    for (Index = 1; Index <= 7; Index++)
    {
        Square(Index); /* This calls the square function */
    }
    Ending(); /* This calls the ending function */
}

/* ===== User Defined Function ===== */
void Header(void) /* This is the function named header */
{
    Sum = 0; /* Initialize the variable "sum" */
    printf("This is the header for the square program\n\n");
}
/* ===== User Defined Function ===== */
void Square(int Number) /* This is the square function */
{
    int Numsq;

    Numsq = Number * Number; /* This produces the square */
    Sum += Numsq;
    printf("The square of %d is %d\n",Number,Numsq);
}
/* ===== User Defined Function ===== */
void Ending(void) /* This is the ending function */
{
    printf("\nThe sum of the squares is %d\n",Sum);
}
```