

II.4. **Structured programming**

Structured programming is the idea of “divide and conquer” applied to computer programming. A program is divided and sub-divided into smaller functions or subroutines that are easier to understand, manage and re-use.

Functions/subroutines

In C/C++ there is no distinction between functions and subroutines. Each one takes 0 or more parameters and returns some type of value (which may be void). There are three aspects to using functions:

1. declaring the function – identifying it’s name and parameters,
2. defining the function – define the actual procedures of the function body, and
3. calling the function – executing the function.

A function must be declared or defined in the source code before it can be called. A function is “defined” using the syntax:

```
<return_type> <function_name>(<parameter_type> <parameter_name>, ...)  
{  
    < function_code >  
    return <return_value>;  
}
```

The return_type may be any variable type, or “void”.

The function_name must start with a letter but can contain numbers and underscores. It is case-sensitive.

Inside the brackets is a list of parameter declarations. These parameters have local scope (are valid only in the function). If a parameter is not being used, the parameter_type can be given without parameter_name.

The function_code will then be executed until the ‘}’ or until a return statement is reached. There can be more than one return statement. If the return type is “void” there may be none.

The return statement must give a value as the result of the function calculations unless the return_type is “void”.

Any variables declared in a function are local to that function.

The function is called by giving the function name, and giving variables or constants for the function parameters. E.g.

```
#include <iostream.h>

using namespace std;

void Print_Food(char food_letter)
{
    if (food_letter == 'a')
        cout << "Apple";
    else if (food_letter == 'b')
        cout <<"Banana";
    else if (food_letter == 'c')
    {
        cout <<"Carrot";
        cout << " and Corn";
    }
    else if (food_letter == 'd')
        cout <<"Donut";
    else
        cout <<"I don't know any foods that";

    cout << " start(s) with '" << food_letter << "'." << endl;
}

int main()
{
    char food_letter;

    for (food_letter='a'; food_letter <= 'e'; food_letter++)
        Print_Food(food_letter);

    return 0;
}
```

A function is “declared” using the syntax:

<return_type> <function_name>(<parameter_type> <parameter_name>, ...);

For example:

```
void Print_Food(char first_letter); // "void" means no type
```

This is known as the function “prototype” and does not actually contain any code.

recursion

Recursion is when a function executes a nested instance of itself within the body of the function. Recursive functions can sometimes provide very eloquent and simple implementations for complicated algorithms. However, when using recursion, one must be careful to provide some way of stopping the recursion, otherwise the program will never come out from the recursion.

Here is a non-recursive function for computing factorials.

```
int Factorial(int n)
{
    int i, fact = 1;
    for (i = 1; i<=n; i++)
        fact = fact * i;
    return fact;
}

int main()
{
    int i;
    for (i=1; i<10; i++)
        cout << i << "! = " << Factorial(i) << endl;
    return 0
}
```

Here is a recursive version of the function. Note that Factorial() is called from within the Factorial() function. The recursive version of this function uses fewer lines than the non-recursive version. Note also that there is an “if” statement that makes it possible to return from Factorial() without calling the nested Factorial() in order to stop the recursion.

```
int Factorial(int n)
{
    if (n<2)
        return 1;
    return n*Factorial(n-1);
}
```

Existing functions

Prototypes for existing functions are found in header files, included using the #include<> compiler directive. Standard header files that come with C++ are:

iostream – input and output functions

stdio.h – old style input/output functions (use iostream instead)

stdlib.h – old style functions for allocating memory, starting other programs, etc.

string – has the String class

string.h – old style char * manipulation (use String instead)

math.h – math functions such as sqrt()

These header files are found in the directories `/usr/include` or `/opt/SUNWspro/WS6U2/include/CC/Cstd`. Use the UNIX “man” command to find out more about the functions in these files.