CMPT 102
Introduction to Scientific Computer Programming

Introduction to simple functions

© Janice Regan, CMPT 102, Sept. 2006

Modularization

- When solving a real scientific (or other) programming problem it will be necessary to break the problem into pieces or modules.
- Each module can be separately developed and debugged
- Modules can then be assembled and tested to build the solution to the entire problem
- When programming in C it is often useful to write each module as a function.
- A function is an independent program that solves a problem. That problem is often a component of a larger problem

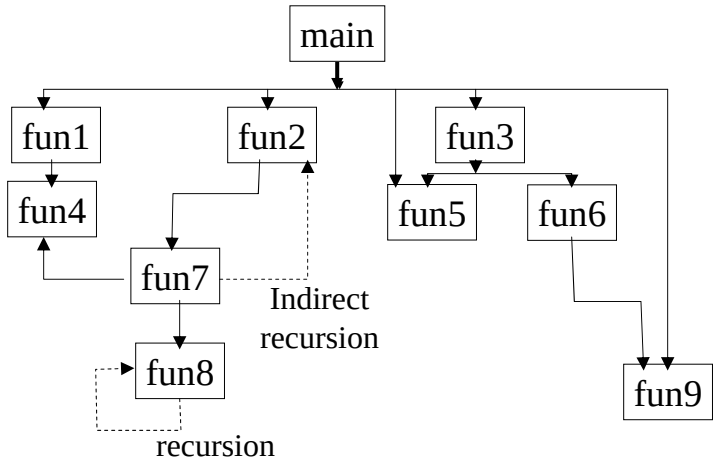
© Janice Regan, CMPT 102, Sept. 2006

Modularization

- Any module or function can use (call) any other function
- When designing a solution for a larger problem it is often useful to draw a map of which functions call which other functions ***(structure chart or module chart)***
- A function can even call itself. Such a function is called a recursive function and requires special care to design
- Beware of unintended recursion (most commonly function1 calling function2 which calls function1, this type of recursion is also called indirect recursion). This can cause serious problems.

© Janice Regan, CMPT 102, Sept. 2006

Module Chart



© Janice Regan, CMPT 102, Sept. 2006

Types of functions

- Functions can be user defined
  - Written by the user for applications specific to the developers needs
  - Part of libraries associated with particular tools being used by the developer to built more complex applications
    - Graphics libraries
    - Statistical analysis libraries

Types of functions

- Functions can be built in functions
  - Functions provided by the C language (or other programming language) to accomplish common tasks
    - Trigonometric and mathematical functions
    - Input and output functions

Defining a function

- The first line of a function is the ***function definition***
- The function has a type
  1. ***double*** sinc(double x)
  - If the type of a function is ***void*** it returns no value
  - Otherwise the function returns a value of the same type as the function
  - The value of the expression in 1. will be the value returned from the function
- The function has 0 or more parameters (arguments)
  - double sinc(double ***x***) /*function with 1 argument*/
  - int printmsg(void); /*function with no arguments*/

© Janice Regan, CMPT 102, Sept. 2006

Defining a function

- Each of a function's parameters have types
  - A function may have parameters of more than one type
  - In the function definition each parameter must be given its own type
  - double sinc(***double*** x)
- The function has a unique name or identifier
  - double ***sinc***(double x)
  - Beware: if you write a function with the same name as a built in function it will replace that built in function
- There is no ; at the end of a function definition

© Janice Regan, CMPT 102, Sept. 2006

Sample Function

```
double sinc(double x)
{
    if (fabs(x) < 0.0001)
    {
        return(1.0);
    }
    else
    {
        return( sin(x)/x);
    }
}
```

Function definition

Function body

Declaring a user's function

- If we wish to use a user supplied function in our main program we must declare that function in our main program
- If we wish to use another user supplied function within a function we are writing we must declare that function within the function we are writing
- A function prototype (what we call a function declaration) usually follows the variable declarations within the main program or function
- A function prototype tells us the name and type of the function, and the types and order of the parameters.
- A function prototype looks like the definition of a function followed by a ; parameter identifiers may be omitted

  double sinc(double x); or double sinc(double);

The body of a function

- After the function definition the body of the function is enclosed in { }

```
double sinc(double x)
{
   variable declarations
   function declarations
      calculations to determine y
   return(y);
}
```

- When we use the function the expression
  sinc(myDoubleVariable)
  appearing in the program calling the function will have value y where y is the sinc of the value of myDoubleVariable

Returning a function's value:1

- In general a simple function will take the supplied values of our parameters, calculate a result, then return that result to the calling program
- The function has a type. The type of the function is the type of the value returned by that function to the calling program
- A function can return the value of a single variable of a specified type
- Later we will look at how to use the parameter list to provide the calling program access to more than one value from the function

Returning a function's value:2

- Our sample function determines the value of sin(x)/x when we supply a value for the parameter x
- The function sinc(x) will take the value of x, calculate the value of sin(x)/x and return the resulting value to the calling program
- To return the value of the function to the calling program the command

  return(ValueToBeReturned);

  is used. The type of ValueToBeReturned should match the type of the function returning the value
- A function of any type other than void must contain at least one return statement. It may contain more.

© Janice Regan, CMPT 102, Sept. 2006

Using a function

- Once we have declared a function we can then use it in the body of our function or main program
- If we are using a built in function from a C library we need not declare it (the declaration is inside the libname.h file)
- To use a function we write an expression of the form
  - FunctionName(myintvariable, myfloatvariable, myintvariable2);
  - sinc(MyValue);
- The value of the functional expression has the same type as the function and will contain the value returned by the function.
- The identifier for a variable that is used in place of a parameter must identify a variable with the same type as that parameter.
- It is not necessary to use the same identifier that was used in the function declaration when you use the function, any variable of the right type will do

© Janice Regan, CMPT 102, Sept. 2006

Example of calling a function

```
#include <stdio.h>
int main( )
{
   /* calculate the number of combinations of r objects */
   /* chosen from m objects */

   /* declare local variables */
   int n =12, r= 4, c= 0;

/* declare functions by giving prototypes */
   int Combination(int m, int r);

   c = Combination( n, r );
   printf("The number of combinations is %d\n", c);
   return 0;
}
```


Analysis of example (1)

- Combination is a function that determine the number of ways r objects can be chosen from m distinct objects (order not significant)
- The number of combinations is an integer so the type (return value) of the function is an integer
- The numbers of objects in the whole group and the numbers of objects chosen to form a combination are also integers so the arguments of the function are both integers

Analysis of example (2)

- The value of the expression
  Combination(n, r)
  is the number of combinations. This type of expression is referred to as a call to a function.
  - In this example this is a call to the function whose identifier (name) is Combination
  - This call to function Combination will cause the function Combination to be executed to determine the number of combinations of r objects chosen from n objects. The resulting number of Combinations will be assigned to be the value of this expression
- The assignment statement below is used to put the value (the number of combinations) into variable c
  c = Combination (n, r);

© Janice Regan, CMPT 102, Sept. 2006

Summary

- This quick summary of how functions work is enough to help you use the built in functions of the C language or user defined functions supplied to you.
- Before you write your own functions we will return to this topic for a more complete discussion

© Janice Regan, CMPT 102, Sept. 2006