Computer Programming

An-Najah N. University
Computer Engineering Department
Luai Malhis, Ph.D,

Functions

Introduction

- Computer programs that solve real-world problems are usually much larger than the simple programs discussed so far.
- To design, implement and maintain larger programs it is necessary to break them down into smaller, more manageable pieces or *modules*.
- Dividing the problem into parts and building the solution from simpler parts is a key concept in problem solving and programming.

Introduction Continues

- In C++ we can subdivide the program into blocks of code known as **functions**. In effect these are subprograms that can be used to avoid the repetition of similar code and allow complicated tasks to be broken down into parts
- Until now we have encountered programs where all the code (statements) has been written inside a single function called main(). Every executable C++ program has at least this function.

Function Definition

- A function Is comprised of heading and a body. Following is the syntax of function definition.

```
return type function name (data type parameter(s))
{
    statement(s);  //function body
}
```

```
void main ( )
{
    statement(s);
}
```

```
int main ( )
{  statement(s)
   return (exp);
}
```

Examples

```
void print2lines ( )
{ cout<<"**************************\n";
  cout<<"**************************\n";
}

int sum2int (int a, int b) // params are declared
{    int c;
     c = a + b;
     return (c);}
```


Function Return Type

- May be void, indicating that nothing will be returned, or any of the data types (int, float, double, char,…, or a pointer to type)
- When the return type is any thing other than void, a return statement must be part of the function body. In this case the function return a value (Value - Returning functions)
- When functions are part of a conditional expression they can not be void. Example
  If (pow(2,3) > 5) …..

Return Statement and its Placement

- Are included in value-returning functions (non-void functions).
- Terminate the execution of statements in a function. Any statements after the return statement are not executed.
- Send a single value back to the calling function may be main or another function.
- int f1(int x) { x = x *2; return x; cout << x;}
  the cout statement never executed

Function Placement

- a function may be placed before main or after main. If it defined after main, it requires function prototypes. Example:

```
int f1() { int y; ……;return();}   // impl.
double f2(int);                   // proto
void f3(char);                    // proto
void main(){…. F1…… F2…. F3};   // impl
double f2(int z) {……return();} // impl
void f3(char z) { ….};          // impl
```

Function Prototypes

- Act similar to a variable declarations.
- Occur before its called in the file (before main).
- Look similar to function heading except that it ends in a semicolon and it does not require the parameter name, just the data type.

```
void print2lines ( );
int sum2int (int, int);
```

- If the parameter name is included, it is ignored.

Function Format

```
//comments
# include library files
Global variables
function definition(s)
{
   statement(s)   //body
}
void main ( )
{
   statement(s)
}
```

```
# include library files
function prototype(s)
global variables
void main ( )
{
   statement(s)
}
function definition(s)
{
statement(s)  // body
}
```


Calling Functions

- A function call is the statement that tells the function to execute.
- When execution of the function has been completed, flow of the program returns to the statement immediately after the call.
- It has the format of
  function name (parameters);
- The parameters from the call are transferred to the function heading.
- Function calls for value-returning functions are often part of assignment statements.

Examples of Function Calls

```
#include <iostream.h>
int readint() {int x; cin >>x}
void printint(int x) {cout << x << endl;}
int sum2int( int x, int y){ return(x+y);}

void main ( ) {
   int a,b,c;
    a = readint();   b=readint();
    c = sum2int(a,b);
     printint(c);  }
```

More on Function Calls

- A function can call another function or even itself (recursive functions)
- A single C++ statement may call more than one function. Example: sqrt(x) + pow(x,y);
- Function calls can be part of a condition.

```
if (sum2ints(6, 12) > 10) {statements}
if (function()  ) {statements}
```

- Function could be part of an expression

```
X + function() ……
int y = function();
```


Parameters

- The parameters in the function heading are referred to as “formal parameters” or arguments.

```
int add(int x, int y) {    return(x+y);}
```

- The parameters in the function call are referred to as “actual parameters” or arguments.

```
int w = 10;    cout << add(4,w);
```

- When the function call is executed, the actual parameters are transferred to the formal parameters in order that they appear in the call.
- We will discuss some restrictions later.
- Formal parameters and actual parameters are different variables even if the have the same name

Passing Parameters

- **Pass by value** – a copy of the value from the actual parameter is sent to the formal parameter of the function. The function can not change the value of the actual parameter.
- **Pass by address (pointer)**- the formal parameters point to the actual parameters. The function can change the value of the actual parameter.
- **Pass by reference** – the same memory location is shared by actual parameter and the formal parameter. The function can change the value of the actual parameter.

Pass by Value

- Only the value of the actual parameter stored in the formal parameter.
- The actual parameters and the formal parameters are separate memory locations
- Example:

```
int f1(int x) {  x = x/2; cout << x;  return(x);}
void main() { int y =10;
                cout << f1(y) << endl;   // 5;
                cout << y;            //  10
}
```

Pass by Value 2

Pass by can be generalized to include passing the value of any expression to the function.

```
int f2(int x) { x = x/2; return x;}
```

In main() we can call the function as follows:

```
void main() {  int y =10;
                cout << f2(y);                 // 5
                cout <<  f2(30);               // 15;
                cout << f2(25 * 2 + y);        // 30
                cout << f2(f2(y)/2 + 15);      // 10
              }
```


Passing by Address

- The address of a variable is passed to the function and the function access the variable using a pointer.
- Pass address of the variable using **&** operator
- Define the formal parameter as pointer in the function
- **Use** ***** operator to access and make changes to the value of the variable from inside the function.

```
void f3(int *x) { *x =5};
void main{ int y =10; f3(&y); cout << y;}
```

// it prints 5 because the function call changes the content of location y in main using pointer x.

Pass by Reference

- Reference is defining another name to previously defined variable. Reference declaration syntax:

  type & name = variable;

  int x =5, y=10; int & z = x;
- Now both x and z are names for the same location.

  x → [5] ← z // the location can be accessed either using the name x or the name z.

  cout << z; //5 z= 20; cout << x; // 20
- Int & z = y; // syntax error already defined z.
- Int & w; // syntax error must assign to variable;

Pass by Reference 2

- We can use references to pass variables to functions by defining another name in the function to the variable passed.
- Use reference name in the function to access and make changes to the value of the variable from inside the function.

```
void f3(int &x) { x =5};
void main{ int y =10; f3(y); cout << y;}
```

// it prints 5 because the function call changes the content of location y in main using reference x.

Function call Example

- Suppose your know the length of the sides of a rectangle and you want a single function to calculate both the perimeter and the area of the rectangle.
- Your function would need 4 parameters, rect_length, rect_width, perimeter, and area.
- The parameters rect_length and rect_width would be passed by value because you do not want the function to change them. The parameters perimeter and area would be passed by reference or by address because the function calculates them and store changes.

Example (cont.)

- The Function Definition would be

```
void calcAreaPeri(float rect_length,
   float rect_width, float &perimeter, float * area)
   {   perimeter = 2 * rect_length + 2 * rect_width;
     *area = rect_length * rect_width; }
```

- The function prototype would be

```
void calcAreaPeri(float, float, float&, float *);
```

- The function call would be: float p, a;

```
calcAreaPeri(4.6, 8.5, p, &a);
```


Example (cont.)

- Write main that calls the function

```
void main ( ) {       float len, width, p = 0, a = 0;
cout<<"Please enter the length and width. \n";
cin >>len >> width;    // len = 4   and width = 6
calcAreaPeri(len, width, p, &a);
cout<<"A rectangle of length "<< len    << endl;   //  4
cout <<  "and width of "          <<width << endl;   // 6
cout << " and has a perimeter of "<< p    << endl;   // 20
cout  <<" and has an area of"<< a<<   endl;         // 24
}
```


Summary of Passing Parameters

- With pass by value, the actual parameter may be a variable, constant , or an expression.
- Pass by reference the actual parameter must be variable name. use & in the function heading.

```
void Func(int & a , float b , char & c ) {    }
main ()   { int x, float y, char z;  Func(x,y,z);}
```

- With pass by address, the actual parameter must be the address of a variable and defined as a pointer in the function heading.

```
Func (int *a,  float b, char *c)  {   }
main ()   { int x, float y, char z;  Func(&x,y,&z);}
```

Passing Arrays to Functions

- An entire array can be passed to a function.
- In the function call, just use the array name without any indices or subscripts.
  - Often the number of elements in the array is passed as another argument of the function call.
  - Printarray(myarray, 30);
- In the function heading, the size of the array should be left blank by using empty brackets
  - void Printarray(float numbarray[ ], int size)

Some Details of Array Passing

- Previously in functions, we discussed that we could pass parameters "pass by value or "pass by reference" or pass by address.
- However, we cannot pass an entire array as pass by value to a function.
- By default an array is passed in a similar manner (to pass by value) as pass by reference.
- Any changes to array content in the function will be permanent.

Example of passing Arrays to Functions

```
void printarray(float B[ ] , int size)
 { for (int i = 0; i < size; i++) cout << B[i] << " ";}

void  mult2array(float B[], int size)
{  for (int i = 0; i < size; i++) B[i] *= 2; }
// all changes to array content will be present  to main,
void main ( ) {
   float A[10] = {2.5,6.9,7.2,13.1,26.5};
   printarray(A, 10); // 2.5   6.9  7.2  13.1  26.5
   mult2array(A,10);
    printarray(A,10);  // 5.2  13.8  14.4  16.2  53
}
```


Returning a pointer to memory

- Instead of returning a value from a function a pointer to a memory location may be returned provided that the memory location allocated dynamically (using new).

```
int * f(int a ) { int b; int * p = new int[10];  return(p)}
void main ()  { int *x; x = f();}
```

Now x points to new memory pointed to by p.

- Never return a pointer to a (temporarily) variable declared in the function heading or body. Because the memory location disappears when the function end.
- In the example above the function should not include the statement: return (&a) or return(&b).

Designing Functions

- When we design a function we should consider the following points:
- What do I want this function to do?
- What parameters do I need and their type?
- Should this function return a single value?
- if so what is the data type of the returned value?
- What local variables need to be defined within this function?
- Etc.

Example 1

- Write a function to print "Hello" on the screen n times where n is passed as parameters. And write main to call the function .

```
void printhello (int n) {
      for ( int i = 0; I < n ; i++)  cout << "hello"; }
void main() {    int x;
                 printhello(5);
                  cin >> x;  printhello(x);
                 printhello(x*2+19);
               }
```


Example 2

- Write a function that takes an int passed by value, a double passed by address and char passed by reference. In your function multiply the double by the int value and store result in the double value. Add the int to the char value and store the result in the char value. Also return the sum of all three variables. write main to call the function;

```
double comp (int a, double *p, char *c) {
   *p = *p * a;    c  = c + a;  return(a + *p + c); }
void main() { int x =5; double y = 12.5;  char z = 'A';
                double w = comp(x, &y, z); }
```


Example 3

- Write a function that takes two characters c1 and c2. Print all characters between them. Your function return the count of printed chacaters. Example if c1 = f; and c2 = p; prints aghijklmnop. returns 11

```
int printchar (char ch1, char ch2) {
  int count =0;
  for (char ch = ch1; ch <= ch2; ch++,count++)
       cout << ch;
 return(count); }
```

Example 4

- Write a function that takes an array of characters and array size. Your function returns a pointer to new dynamically allocated array that contains all small letters in the passed array.

```
char * getsmall (char A, int size) { int scount = 0;
 for (int i =0; I < size; i++)
     if (A[i] > =‘a’ &&  A[i] <= ‘z’)  scount++;
 char *p = new char[scount];  char *t =p;
 for (int i =0; I < size; i++)
     if (A[i] > =‘a’ &&  A[i] <= ‘z’)  *t++ = A[i];
return(p);   }
```


Example 5

Given: double **M[20][10].** Write a function that takes the matrix **M** as parameter. Your function computes and returns the smallest value in the matrix.

```
double small ( double A[20][10) { //must define # columns
    double small = A[0][0];
    for (int r= 0; r < 20; r++)
        for (int c = 0; c < 10; c++)
            if (A[r][c] < small )  small = A[r][c];
   return(small); }
void main() { double M[20][10]  = {{…},{….},{….}};
            cout << small(M);}
```


Global vs. Local Variables

- A variable that is declared within a block is “local” to that block and may only be accessed within that block. Block is enclosed within { }
- Therefore, a variable declared in a function definition (either heading or body) is local to that function.
- Several functions may use the same identifiers as variable names, but each is stored in a different memory space.

  F1() { int x;} F2() { int x}   main() { int x}
- Global variables are declared outside all functions and may be accessed from anywhere.

  int x; f1 ( ) { x =5;}   f2 () { x +=2;}   main() {x ..

Variable Scope

The variable is known from the point it is defined in a block and any sub block in that block.

A block is the code between { …….}

- Example 1: if (1) { int x; x =5;} cout << x; // syntax error

  The cout statement accesses x outside the block
- Example 2 : { int x =2 ;

```
        { int y =3;  cout << x;  cout << y;
            { int z = 12;  cout << z << x << y}
        }  // all good
        cout << y;  cout << x;  // syntax error on y
    }  cout << x << y << z;  // x,y and z not known
```