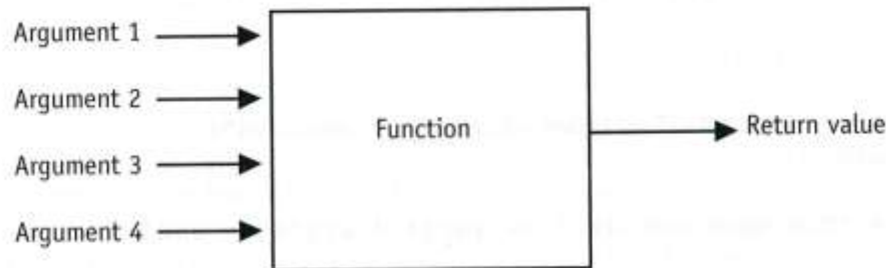
Defining Functions and their inner workings

Petar Jercic
Francisco Luro
Andrew Moss (course-resp)
5 March 2012

Functions (1)

- Collection of statements (code) that perform a specific task.
- **Breaking the problem** into a set of **smaller manageable functions (modules)**, breaking down the algorithm.
- Consider a function as a „black box”; You know how to use the function but not necessarily its method of operation.



- Functions seen and used before:
  - pow() sqrt() main() ...
    - the_root = sqrt(9.0);
- Functions are written **(defined) once** to perform the task, and **called anytime** they are needed.

Functions (2)

```
return-type function-name(argument declarations)
{
        declarations and statements;
}
```

General definition of a function

```
double total_price (int items_number, double item_price) {
        return items_number * item_price;
}
```

Example of a function

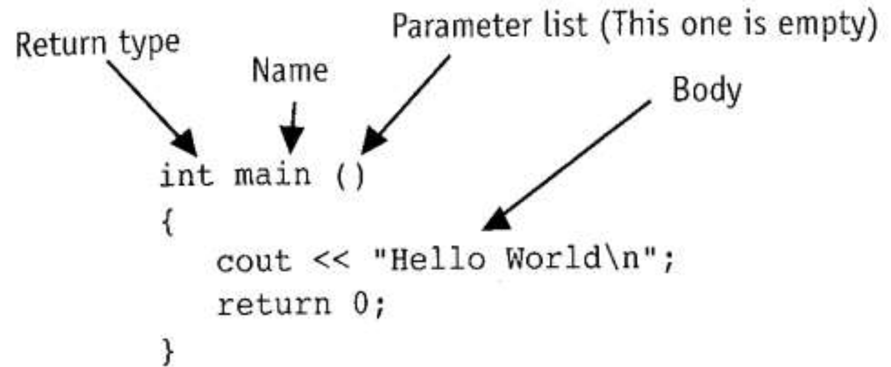
Return type
Name
Parameter list (This one is empty)
Body

```
int main ()
{
    cout << "Hello World\n";
    return 0;
}
```

Example of a function explained.
NOTE: main()

Function Definition (1)

- Every function has:



  - **Name**: same as variable, no spaces!!!
  - **Parameter list**: list the values passed into the function
  - **Body**: Set of statements (code) that will execute
  - **Return type**: Data type of the value sent back to the calling program.

Function Definition (2)

- Methods
  - Fcns without parameters, empty parameter list.
  - *iSecret = rand() % 10 + 1;*
- Void function
  - function that doesn’t return a value; returns void.
  - This fcn. Performs some statements and returns back to the calling program

```
void starline ( ) {
    for ( intj=0, j < 45; j++ )
            cout << "*" ;
    cout << endl;
}
```

Function Call (1)

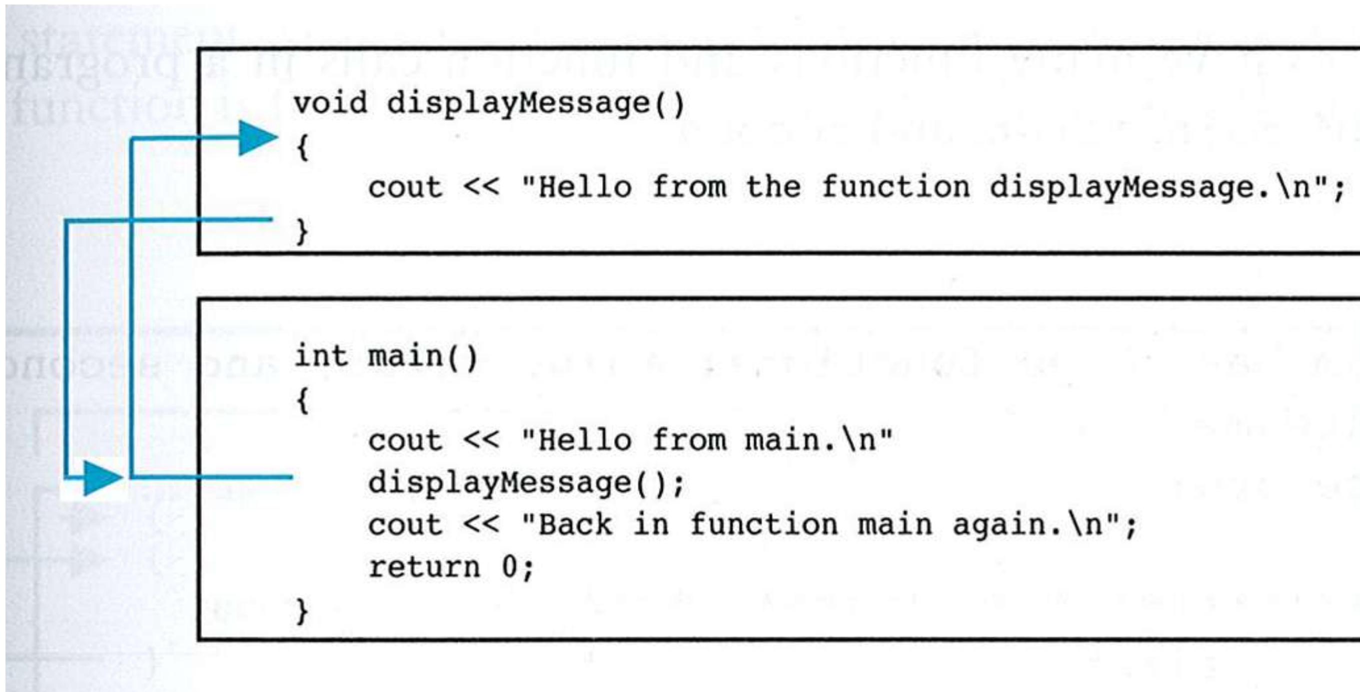
- A function call is a statement that causes function to execute. Function is executed when it is called.
- It is an expression consisting of function name followed by arguments enclosed in parentheses.

```
fcn_name(arg, arg, arg, ...)
side = sqrt(area);
pow(2.5, 3.0);
```

- A function has to be defined before we call the function to execute.
- Function is like a small program, calling a function is like running a program
  - **cin** as **arguments**
  - **cout** as **return value**

Function Call (2)

- When the function is executed a program branches to that function and executes statements in the body
- A function has to be defined before we call the function to execute.



Declaring functions (1)

- Functions have to be declared before calling them since the compiler has to know certain things about the fcn when the call is made.
- Description how to call a fcn (fcn prototype)
  - **return_type name (parameter_type(s));**
  - int sum (int, int);
- Names of the variables are not needed.

Declaring functions (2)

```
double total_price (int items_number, double item_price) {
		return items_number * item_price;
}

int main () {
		cout << "Total price for 12 items, 2.3$ each is " << total_price(12, 2.3) << "$" << endl;
}
```

Definition only

```
double total_price (int items_number, double item_price);

int main () {
		cout << "Total price for 12 items, 2.3$ each is " << total_price(12, 2.3) << "$" << endl;
}

double total_price (int items_number, double item_price) {
		return items_number * item_price;
}
```

Declaration and definition

```
double total_price (int, double);
```

Alternative declaration

```
> Total price for 12 items, 2.3$ each is 27.6$
```

Output

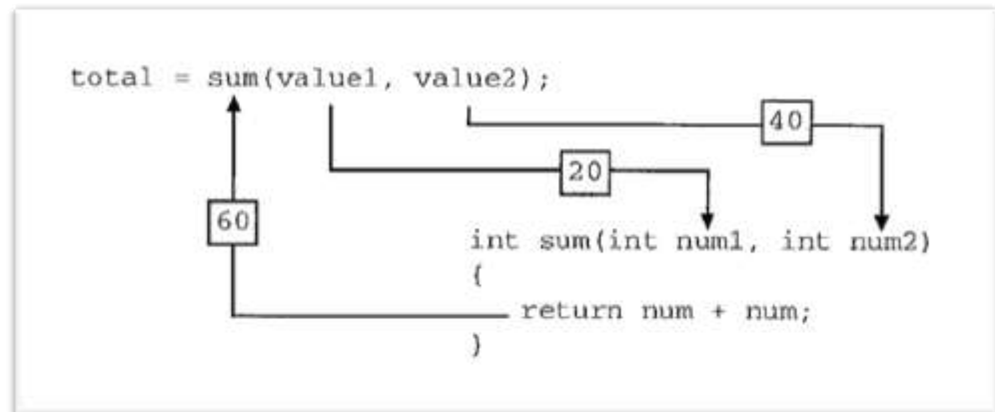
Recap

```
#include <iostream>

using namespace std;

int mult ( int x, int y );

int main()
{
  int x;
  int y;

  cout<<"Please input two numbers to be multiplied: ";
  cin>> x >> y;
  cout<<"The product of your two numbers is "<< mult ( x, y ) <<"\n";
}

int mult ( int x, int y )
{
  return x * y;
}
```

Return

- The return statement is the mechanism for returning a value and control back from the called function to its caller.
- Return causes the function to end **immediately**.
  - Illustrate example with few returns across
- Return can return **ONE** value of a specified data type

```
int max(int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}
```



Sending data into a function (1)

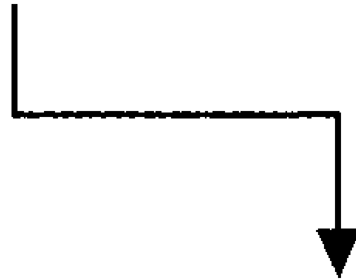
- Values sent into the function are called arguments.
  - result = pow(**2.0, 4.0**);
  - displayValue(**2**);
- A parameter is a variable that hold the value being passed as an argument into the function

```
Void displayValue(int num) {
	cout << „Value is ” << num;
}
```

- What happens if we make a call like this?
  - displayValue(2.3);

Sending data into a function (2)

```
displayValue(5);                // Function call
```

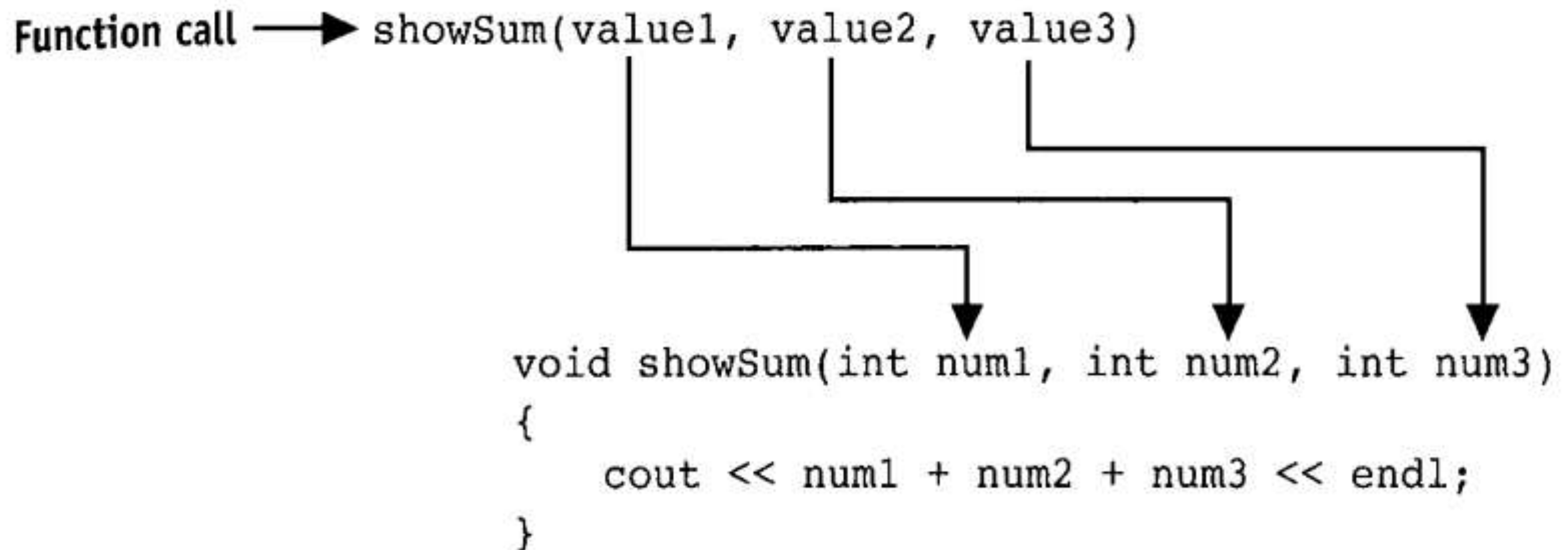


```
void displayValue(int num)    // Function header
{
     cout << "The value is " << num << endl;
}
```

- Arguments can be variable, a constant or an expression result (anything that results in a value)

Sending data into a function (3)

- Arguments are copied in the same order the fcn. parameters are specified.



Arguments - Call by Value

- Values passed inside of the function are called arguments, local variables used inside of the fcn. Are called formal parameters.
- When a fcn. Is called the formal parameters are initialized to the values of arguments.
- Changes on formal parameters inside of the fcn. Will have no effect on arguments in the main body.

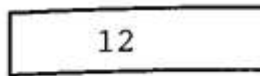
```
void swap(int x, int y) /* WRONG */
{
int temp;
temp = x;
x = y;
y = temp;
}
```

- Because of call by value, swap
- can't affect the arguments x and
- y in the routine that called it.

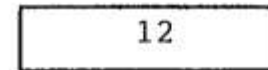
```
– main () {
– int x=1, y=2;
– swap(x,y);
– cout << x << y;
– }
```

- The function above swaps *copies* of x and y.

Original argument
(in its memory location)



Function parameter
(in its own memory location)



Sending an array to the function (1)

- Since C passes arguments to functions by value, there is no direct way for the called function to alter a variable in the calling function.
- Sending an array passes the argument of that array **name** to a fcn. Array’s name refers (address) to the values stored in the memory; thus by changing the array inside of a function, we are changing the original values of data.

Sending an array to the function (2)

```
double total_price (int, double[]);
void sale (int, double[]);
```

```
int main () {

double prices[10] = {2.3, 1.5, 2.3, 3.5, 2.6, 1.5, 2.2, 5.5, 1.8, 3.5 };
cout << "Total price for 10 is " << total_price(10, prices) << "$" << endl;
sale(10, prices);
cout << "Total price for 10 is " << total_price(10, prices) << "$" << endl;
}
```

```
double total_price (int items_number, double item_price[]) {

        double sum_prices = 0;

        for(int i = 0; i < items_number; i++)
                sum_prices += item_price[i];

        return sum_prices;
}
```

```
void sale (int items_number, double item_price[]) {

        for(int i = 0; i < items_number; i++)
        item_price[i] *= 0.5;
}
```

```
OUTPUT:

Total price for 10 is 26.7$
Total price for 10 is 13.35$
```

Sending an array to the function (3)

- Important to know the size of the array inside of the function.

```
void sale (int items_number, double item_price[]) {

        for(int i = 0; i < items_number; i++)
        item_price[i] *= 0.5;
}
```

- Fcn. Patameter is without the index in brackets.

```
double total_price (int items_number, double item_price[])
```

- Fcn. Argument is without brackets.

```
cout << "Total price for 10 is " << total_price(10, prices);
```

Scope (1)

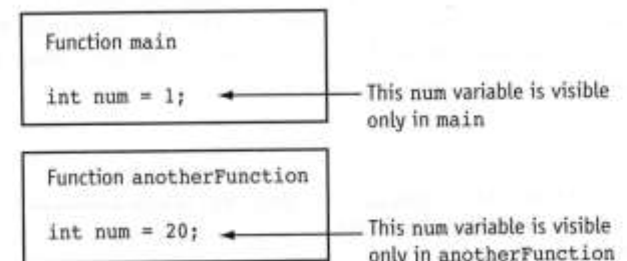
- The *scope* of a name is the part of the program within which the name can be used. For an
- Fcn parameters declared at the beginning of a function, the scope is the function in which the name is declared.

Scope (2)

```
int main () {

string msg = "Total price is ";
	double prices[10] = {2.3, 1.5, 2.3, 3.5, 2.6, 1.5, 2.2, 5.5, 1.8, 3.5 };
	cout << msg << total_price(10, prices) << "$" << endl;
	sale(10, prices);
	cout << msg << total_price(10, prices) << "$" << endl;
}

double total_price (int items_number, double item_price[]) {
	cout << msg; // ???
	double sum_prices = 0;

	for(int i = 0; i < items_number; i++)
	sum_prices += item_price[i];

	return sum_prices;
}

void sale (int items_number, double item_price[]) {

	for(int i = 0; i < items_number; i++)
	item_price[i] *= 0.5;
}
```

Local vs. Global variables (1)

- Communication between the functions is by arguments and values returned by the functions, and through external variables.
- Fcn parameters declared at the beginning of a function, the scope is the function in which the name is declared. They behave as local variables.
- The scope of an global variable or a function lasts from the point at which it is declared to the end of the file being compiled.
- Global variables are defined outside of any function, and are thus potentially available to many functions.
  - Local variable is defined inside of the function
  - Global variable is outside of the function
- Same name in loval/global variables is fine, but local variables overshadow global ones. Try to avoid!!!



Local vs. Global variables (2)

- Global variables are also useful because of their greater scope and lifetime.
- Local variables are internal to a function; they come into existence when the function is entered, and disappear when it is left.
- Global variables, on the other hand, are permanent, so they can retain values from one function invocation to the next.