MA122 - Computer Programming and Applications

Indian Institute of Space Science and Technology

January 13, 2017
Lecture 2

1. My first program

2. Variables

3. Input

4. Function 1
My first program

```cpp
#include <iostream> // a PREPROCESSOR directive
int main() // function header
{
    // start of function body
    using namespace std;
    // make definitions visible

cout << "Come up and C++ me some time."; // message

cout << endl; // start a new line

cout << "You wont regret it!" << endl; // more output

return 0; // terminate main()
} // end of function body
```
1. My first program

2. Variables

3. Input

4. Function 1
My first variable

```cpp
#include <iostream>

int main()
{
    using namespace std;
    int carrots; // declare an integer variable
    carrots = 25; // assign a value to the variable
    cout << "I have ";
    cout << carrots; //display the value of the variable
    cout << " carrots."
    cout << endl;
    carrots = carrots - 1; // modify the variable
    cout << "Crunch, crunch. Now I have " << carrots
    << " carrots." << endl;
    return 0;
}
```
Lecture 2

1. My first program
2. Variables
3. Input
4. Function 1
#include <iostream>

int main()
{
    using namespace std;

    int carrots;

    cout << "How many carrots do you have?" << endl;

    cin >> carrots; // C++ input

    cout << "Here are two more. ";
    carrots = carrots + 2;

    // the next line concatenates output
    cout << "Now you have " << carrots << " carrots." << endl;

    return 0;
}
cin and cout: A touch of class

```cpp
#include <iostream>
using namespace std;

int main()
{
   ...
   ...
   cout << "Trust me";
   ...
   ...
}
```

Figure 2.5  Sending a message to an object.
Lecture 2

1. My first program
2. Variables
3. Input
4. Function 1
// sqrt.cpp -- using the sqrt() function

#include <iostream>
#include <cmath> // or math.h

int main()
{
    using namespace std;
    double area;
    cout << "Enter the floor area, in square feet, of your home: ";
    cin >> area;
    double side;
    side = sqrt(area);
    cout << " Thats the equivalent of a square " << side << " feet to the side." << endl;
    cout << "How fascinating!" << endl;
    return 0;
}
However, if we deal now with some basic characteristics of functions, you'll be more at ease and more practiced with functions later. The rest of this chapter introduces you to these function basics.

C++ functions come in two varieties: those with return values and those without them. You can find examples of each kind in the standard C++ library of functions, and you can create your own functions of each type.

Let's look at a library function that has a return value and then examine how you can write your own simple functions.

Using a Function That Has a Return Value

A function that has a return value produces a value that you can assign to a variable or use in some other expression. For example, the standard C/C++ library includes a function called `sqrt()` that returns the square root of a number. Suppose you want to calculate the square root of 6.25 and assign it to the variable `x`. You can use the following statement in your program:

```c
x = sqrt(6.25); // returns the value 2.5 and assigns it to x
```

The expression `sqrt(6.25)` invokes, or calls, the `sqrt()` function. The expression `sqrt(6.25)` is termed a function call, the invoked function is termed the called function, and the function containing the function call is termed the calling function (see Figure 2.6).

The value in the parentheses (6.25, in this example) is information that is sent to the function; it is said to be passed to the function. A value that is sent to a function this way is called an argument or parameter (see Figure 2.7). The `sqrt()` function calculates the answer to be 2.5 and sends that value back to the calling function; the value sent back is termed the return value of the function. Think of the return value as what is substituted for the function call in the statement after the function finishes its job. Thus, this example assigns the return value to the variable `x`.

In short, an argument is information sent to the function, and the return value is a value sent back from the function.

---

**Figure 2.6** Calling a function.
That's practically all there is to it, except that before the C++ compiler uses a function, it must know what kind of arguments the function uses and what kind of return value it has. That is, does the function return an integer? a character? a number with a decimal fraction? a guilty verdict? or something else? If it lacks this information, the compiler won't know how to interpret the return value. The C++ way to convey this information is to use a function prototype statement.

Note: A C++ program should provide a prototype for each function used in the program. A function prototype does for functions what a variable declaration does for variables: It tells what types are involved.

For example, the C++ library defines the `sqrt()` function to take a number with (potentially) a fractional part (like 6.25) as an argument and to return a number of the same type. Some languages refer to such numbers as real numbers, but the name C++ uses for this type is `double`. (You'll see more of `double` in Chapter 3.)

The function prototype for `sqrt()` looks like this:

```
double sqrt(double);   // function prototype
```

The initial `double` means `sqrt()` returns a type `double` value. The `double` in the parentheses means `sqrt()` requires a `double` argument. So this prototype describes `sqrt()` exactly as used in the following code:

```
x = sqrt(6.25);
```

The terminating semicolon in the prototype identifies it as a statement and thus makes it a prototype instead of a function header. If you omit the semicolon, the compiler interprets the line as a function header and expects you to follow it with a function body that defines the function.

Figure 2.7 Function call syntax.