Functions

Lecture 8

Outline

1. Functions
2. Predefined Functions
3. Programmer-defined Functions
4. Return Values & Arguments

Functions

- Programs can be logically broken down into a set of tasks
- Example from horoscope lab:
  - Get input (month, day) from user
  - Determine astrological sign based on input
  - Output sign and horoscope
- Individual tasks can be separated out from the main program into *functions*
- A function is simply a mini-program that completes a specific task

Predefined Functions

- C++ includes many predefined functions for common programming tasks
- Each predefined function is part of a library, which you need to include if you use that predefined function

```
double root, input;
cout << "Enter a number: \n";
cin >> input;

root = sqrt( input );

cout << "The square root is "
     << root << endl;
```

Function Call

Returned Value

Argument

General Form

```
FUNCTION_NAME( ARGUMENT_1, ARGUMENT_2, …, ARGUMENT_N )
```

- A function can have any number of arguments
  - Each argument has a specified type (`int`, `double`, `string`, etc.)
- A function has either zero or one return values (the result of the function)
  - If it has a return value, the value has a specified type
  - The function can be used in any expression in place of the specified return type

Example Math Functions

```
#include <cmath>
```

Square Root	`double sqrt ( double x )`
Power	`double pow ( double base, double exponent )`
Absolute value	`double fabs ( double x )`
Round up	`double ceil ( double x )`
Round down	`double floor ( double x )`

```
#include <cstdlib>
```

Absolute value	`int abs ( int x )`

Example Usage

```
double number, cube;
cout << "Enter a number: " << endl;
cin >> number;

cout << "The square root is "
     << sqrt( number ) << endl;

cube = pow( number, 3.0 );
cout << number << "^3 is " << cube << endl;

cout << number << " rounded up over "
     << number << " rounded down is "
     << ceil( number ) / floor( number ) << endl;
```

Exercise

- Write a program that prints out the value of 2^x for x = 1, 2, 3, … 32

- Use the **`pow()`** function and a loop

Answer

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

int main()
{
	int x = 1;
	double power;

	cout.setf( ios::fixed );
	cout.precision( 0 );
	while ( x <= 32 )
	{
		power = pow( 2.0, x );
		cout << "2^" << x << "=" << power << endl;
		// or: cout << "2^" << x << "=" << pow( 2.0, x ) << endl;
		x++;
	}

	return 0;
}
```

Programmer-Defined Functions

- C++ allows you to define your own functions to meet the needs of your specific program

- To define your own function, you need a *function declaration* and a *function definition*
  - The declaration and definition tell C++ how many arguments your function needs, what the type of the return value is, and the actual statements that make up your function
  - Similar to declaring and initializing variables

No Arguments, No Return Value

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

void say_hello();

int main()
{
    say_hello();
    return 0;
}

void say_hello()
{
    cout << "Hello!" << endl;
    return;
}
```

void means no return value

Empty () means no parameters

Function Declaration

Call to Execute

Function Definition

Function Statements

Function Execution

- When a program executes a function, it temporarily stops where it is, goes to the lines of code in the function, and executes those lines like normal

- Then, when you get to the end of the function (or a `return` statement) it goes back to where it was and resumes executing after the function call

No Arguments, One Return Value

int means returns an integer

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

int get_integer();

int main()
{
	int val;
	val = get_integer();
	cout << "Got " << val << endl;
	return 0;
}

int get_integer()
{
	int input;
	cout << "Please enter an integer: " << endl;
	cin >> input;
	return input;
}
```

Has to return an integer

Return Values

- Functions have zero or one return values
- If a function has a return value, it is of a specific type (`int`, `double`, `char`, `bool`, …)
  - Type is defined as part of the function declaration and definition
  - Else `void`
- Use the `return` statement to return a value of the specified type
  - Can be a constant, variable, expression, function, or anything that is evaluated as the type

Another Example

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

char get_character();

int main()
{
	char val;
	val = get_character();
	cout << "Got " << val << endl;
	return 0;
}

char get_character()
{
	char input;
	cout << "Please enter a character: " << endl;
	cin >> input;
	return input;
}
```

Exercise

Write a function named **`get_double()`** that reads a value from the user and returns it, then print that value in **`main()`**

Answer

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

double get_double();

int main()
{
	double val;
	val = get_double();
	cout << "Got " << val << endl;
	return 0;
}

double get_double()
{
	double input;
	cout << "Please enter a double value: ";
	cin >> input;
	return input;
}
```

Functions with Arguments

- Functions can take any number of arguments
- Each argument has a set type (`int`, `double`, `char`, `bool`, …), defined as part of the function declaration
- When called, the VALUE of the argument is passed to the function
  - In other words, the values of the arguments are plugged in to the function, just like in a normal expression

Example with Two Arguments

Returns a double value

First argument is a double

Second argument is a double

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

double do_calculation(double x, double y);

int main()
{
    double input1, input2, result;
    cout << "Please enter two numbers: ";
    cin >> input1 >> input2;
    result = do_calculation( input1, input2 );
    cout << "Answer is " << result << endl;
    return 0;
}

double do_calculation(double x, double y)
{
    return ( x*x + y*y );
}
```

input1 is passed as x

input2 is passed as y

Behavior of Arguments

- Each time a function is called, you can pass it different arguments
- The arguments are plugged in separately each time, so you can call a function with many times with different arguments to get a different return value

```
int main()
{
    double result1, result2;
    result1 = do_calculation( 2, 4 );  // in function, x=2 and y=4
    result2 = do_calculation( 1, 9 );  // in function, x=1 and y=9
    // result1=20 and result2=82
}

double do_calculation(double x, double y)
{
    return ( x*x + y*y );
}
```

Another Example

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

bool is_even(int number);

int main()
{
     int input;
     cout << "Please enter a number: ";
     cin >> input;
     if ( is_even( input ) )
     {
          cout << input
               << " is even" << endl;
     }
     else
     {
          cout << input
               << " is odd" << endl;
     }
     return 0;
}
```

```
bool is_even(int number)
{
     if ( ( number % 2 ) == 0 )
     {
          return true;
     }
     else
     {
          return false;
     }
}
```

Functions that return boolean values are often used in boolean expressions

Functions can have multiple return statements

Multiple `return` Statements

- Functions (including the **`main()`** function) can have multiple **`return`** statements in them

- When a **`return`** statement is executed, the function stops and the stated value is returned to the caller immediately
  - No more of the function is executed (unless it is called again, in which case it starts over)
  - In the case of **`main`**, the program quits and returns the value to the operating system

Exercise

Write a program that calculates the area of a rectangle given the two side lengths which are provided by the user. You must write a function that is passed the two side lengths and returns the area.

Answer

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

double area(double l, double w);

int main()
{
     double length, width;
     cout << "Enter the rectangle's length: ";
     cin >> length;
     cout << "Enter the rectangle's width: ";
     cin >> width;
     cout << endl << "The area is " << area( length, width ) << endl;
     return 0;
}

double area(double l, double w)
{
     return ( l * w );
}
```

Wrap Up

- Functions have either zero or one return values
  - If it has one, the value is a specified type
- Functions have zero or more arguments
  - Each argument (if any) has a specified type
  - When called, the current values of the arguments are plugged in and passed to the function
- Functions can have multiple `return` statements