03 - Program Structure and Variables

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Maryville COLLEGE

Outline

Outline

Startup

1. Log in to your shell account
2. `cd cs1-fall2019-username`
3. `git pull`

hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "hello, world" << endl;
}
```

hello.cpp

Preprocessor →

```
#include <iostream>

using namespace std;

int main()
{
    cout << "hello, world" << endl;
}
```

hello.cpp

Preprocessor →
```
#include <iostream>
```

```
using namespace std;
```

Main Header →
```
int main()
{
    cout << "hello, world" << endl;
}
```

hello.cpp

Preprocessor →
```
#include <iostream>
```

```
using namespace std;
```

Main Header →
```
int main()
{
    cout << "hello, world" << endl;
}
```
← Begin Block

← End Block

hello.cpp

Preprocessor →
```
#include <iostream>
```

```
using namespace std;
```

Main Header →
```
int main()
```
```
{
```
← Begin Block

Main Body →
```
    cout << "hello, world" << endl;
```
```
}
```
← End Block

Preprocessor Directives

- The preprocessor runs before the main compilation takes place.

Preprocessor Directives

- The preprocessor runs before the main compilation takes place.
- Preprocessor directives begin with `#`.

Preprocessor Directives

- The preprocessor runs before the main compilation takes place.
- Preprocessor directives begin with `#`.
- The include directive copies the contents of a file to its location.

Preprocessor Directives

- The preprocessor runs before the main compilation takes place.
- Preprocessor directives begin with `#`.
- The include directive copies the contents of a file to its location.
- `iostream` is a C++ library file which contains definitions for input and output.

Preprocessor Directives

- The preprocessor runs before the main compilation takes place.
- Preprocessor directives begin with `#`.
- The include directive copies the contents of a file to its location.
- `iostream` is a C++ library file which contains definitions for input and output.
- For any program that does input and output in C++, you must therefore have the directive:
  `#include<iostream>`

using

- In order to avoid name collisions, C++ has namespaces.

using

- In order to avoid name collisions, C++ has namespaces.
- All of the C++ library is in the `std` namespace.

using

- In order to avoid name collisions, C++ has namespaces.
- All of the C++ library is in the `std` namespace.
- To access these elements, we would normally have to use the `::` operator.

using

- In order to avoid name collisions, C++ has namespaces.
- All of the C++ library is in the `std` namespace.
- To access these elements, we would normally have to use the `::` operator.
- for example, the cout line in `hello.cpp` would read:
  `std::cout « "hello, world" « std::endl;`

using

- In order to avoid name collisions, C++ has namespaces.
- All of the C++ library is in the `std` namespace.
- To access these elements, we would normally have to use the `::` operator.
- for example, the cout line in `hello.cpp` would read:
  `std::cout « "hello, world" « std::endl;`
- Needless to say, this gets tedious!

using

- In order to avoid name collisions, C++ has namespaces.
- All of the C++ library is in the `std` namespace.
- To access these elements, we would normally have to use the `::` operator.
- for example, the cout line in `hello.cpp` would read:
  `std::cout << "hello, world" << std::endl;`
- Needless to say, this gets tedious!
- The using line tells c++ to import all the objects from a namespace so we don't have to use `::` to access them.
  `using namespace std;`

The `main` Function

- Every C++ program begins its execution in the main function.

The `main` Function

- Every C++ program begins its execution in the main function.
- This is formally called **the program entry point**.

The `main` Function

- Every C++ program begins its execution in the main function.
- This is formally called **the program entry point**.
- The main function returns an integer to the operating system. A 0 means success, all other numbers are errors.

The `main` Function

- Every C++ program begins its execution in the main function.
- This is formally called **the program entry point**.
- The main function returns an integer to the operating system. A 0 means success, all other numbers are errors.
- If you do not specify a return value, the compiler will default to 0.

The `main` Function

- Every C++ program begins its execution in the main function.
- This is formally called **the program entry point**.
- The main function returns an integer to the operating system. A 0 means success, all other numbers are errors.
- If you do not specify a return value, the compiler will default to 0.
- All of your code, for now, will go in between the curly braces that mark the start and end of the main function.

Blocks and Statements

- A statement is a group of operations terminated by a semicolon `;`.

Blocks and Statements

- A statement is a group of operations terminated by a semicolon `;`.
- For example, this line is the only statement in `hello.cpp`:

```
cout « "hello, world" « endl;
```

Blocks and Statements

- A statement is a group of operations terminated by a semicolon `;`.
- For example, this line is the only statement in `hello.cpp`:
  `cout « "hello, world" « endl;`
- C++ does not care about white space, so a statement can span multiple lines.

Blocks and Statements

- A statement is a group of operations terminated by a semicolon `;`.
- For example, this line is the only statement in `hello.cpp`:
  `cout « "hello, world" « endl;`
- C++ does not care about white space, so a statement can span multiple lines.
- Groups of statements are called blocks, and they are wrapped in curly braces: { and }.

Blocks and Statements

- A statement is a group of operations terminated by a semicolon `;`.
- For example, this line is the only statement in `hello.cpp`:
  `cout « "hello, world" « endl;`
- C++ does not care about white space, so a statement can span multiple lines.
- Groups of statements are called blocks, and they are wrapped in curly braces: { and }.
- The `main` function's body is a block of code.

Blocks and Statements

- A statement is a group of operations terminated by a semicolon `;`.
- For example, this line is the only statement in `hello.cpp`:
  `cout « "hello, world" « endl;`
- C++ does not care about white space, so a statement can span multiple lines.
- Groups of statements are called blocks, and they are wrapped in curly braces: { and }.
- The `main` function's body is a block of code.
- Blocks can be nested inside each other (more on this later).

Comments

- Comments are notes which explain the code.

Comments

- Comments are notes which explain the code.
- The text of comments are ignored by the compiler.

Comments

- Comments are notes which explain the code.
- The text of comments are ignored by the compiler.
- C++ has two types of comments:

```
// Everything to the end is a comment
/* Everything within is a comment */
```

Comments

- Comments are notes which explain the code.
- The text of comments are ignored by the compiler.
- C++ has two types of comments:

```
// Everything to the end is a comment
/* Everything within is a comment */
```

- The `//` comment is new to C++, and is the preferred method.

Comments

- Comments are notes which explain the code.
- The text of comments are ignored by the compiler.
- C++ has two types of comments:

```
// Everything to the end is a comment
/* Everything within is a comment */
```

- The `//` comment is new to C++, and is the preferred method.
- `/* */` are c-style comments and can be used to make multi-line comments, but be careful!

Comments

- Comments are notes which explain the code.
- The text of comments are ignored by the compiler.
- C++ has two types of comments:

```
// Everything to the end is a comment
/* Everything within is a comment */
```

- The `//` comment is new to C++, and is the preferred method.
- `/* */` are c-style comments and can be used to make multi-line comments, but be careful!
- Every program should have comments (lest they lose points when being graded).

boilerplate.cpp

1. `cd ~/cs1-fall2019-`*`username`*
2. Create the file `boilerplate.cpp` and enter the following:

```
// File:
// Purpose:
// Author:
#include <iostream>

using namespace std;

int main()
{

}
```

The Stream Insertion Operator

- `cout` is the character output stream.

The Stream Insertion Operator

- `cout` is the character output stream.
- Inserting data into `cout` will display it on the screen.

The Stream Insertion Operator

- `cout` is the character output stream.
- Inserting data into `cout` will display it on the screen.
- The operator « is the **insertion operator**.

The Stream Insertion Operator

- `cout` is the character output stream.
- Inserting data into `cout` will display it on the screen.
- The operator « is the **insertion operator**.
- `endl` is a constant which means “end of line”.

The Stream Insertion Operator

- `cout` is the character output stream.
- Inserting data into `cout` will display it on the screen.
- The operator « is the **insertion operator**.
- `endl` is a constant which means “end of line”.
- So the line of C++:
  `cout « "hello, world" « endl;`
  means “Display the words ‘hello, world’ and then end the line”

The Stream Insertion Operator

- `cout` is the character output stream.
- Inserting data into `cout` will display it on the screen.
- The operator « is the **insertion operator**.
- `endl` is a constant which means “end of line”.
- So the line of C++:
  `cout « "hello, world" « endl;`
  means “Display the words ‘hello, world’ and then end the line”
- **Something to Try**: Remove the `« endl` portion of this line in hello.cpp. Recompile and run it. What changed?

Multiple Lines of Output

Often, we want to have multiple lines of text. This can be done in one statement!

```
cout << "Tell me, where is fancy bred?" << endl
     << "  Or in the heart, or in the head?" << endl
     << "                  --William Shakespeare" << endl
     << "                   (Merchant of Venice)" << endl;
```

Challenge: Draw a Diamond!

Challenge: Write a program `diamond.cpp` in your `labs/week2` folder which uses a single statement to print the following figure (begin by copying your `boilerplate.cpp` file!:

```
        #
       ###
      #####
     #######
    #########
  ###########
    #########
     #######
      #####
       ###
        #
```

Outline

The Basic Idea of Variables

- Variables are where programs store data.

The Basic Idea of Variables

- Variables are where programs store data.
- Variables can be assigned values, be used in operations, and can be changed.

The Basic Idea of Variables

- Variables are where programs store data.
- Variables can be assigned values, be used in operations, and can be changed.
- In C++, variables are strongly typed. That is, each variable can only store one type of information!

Types and Declarations

- C++ has the following variable types:

Types and Declarations

- C++ has the following variable types:

  bool Stores a value that is either true or false

Types and Declarations

- C++ has the following variable types:

bool	Stores a value that is either true or false
char	Stores a single character (a letter, digit, or any other symbol)

Types and Declarations

- C++ has the following variable types:

bool	Stores a value that is either true or false
char	Stores a single character (a letter, digit, or any other symbol)
int	Stores an integer

Types and Declarations

- C++ has the following variable types:

bool	Stores a value that is either true or false
char	Stores a single character (a letter, digit, or any other symbol)
int	Stores an integer
float	Stores a single precision floating point number (don't use these!)

Types and Declarations

- C++ has the following variable types:

bool	Stores a value that is either true or false
char	Stores a single character (a letter, digit, or any other symbol)
int	Stores an integer
float	Stores a single precision floating point number (don't use these!)
double	Stores a double precision floating point number.

Types and Declarations

- C++ has the following variable types:

bool	Stores a value that is either true or false
char	Stores a single character (a letter, digit, or any other symbol)
int	Stores an integer
float	Stores a single precision floating point number (don't use these!)
double	Stores a double precision floating point number.

- Variables must be declared before they are used:

```
int x;
char letter;
double num;
```

Variable Names

Variables names:

- must begin with a letter or _.

Variable Names

Variables names:

- must begin with a letter or _.
- can contain letters, numbers, or _.

Variable Names

Variables names:

- must begin with a letter or _.
- can contain letters, numbers, or _.
- are case sensitive.

Variable Names

Variables names:

- must begin with a letter or _.
- can contain letters, numbers, or _.
- are case sensitive.
- must be unique.

The `cin` Stream

- `cin` is the character input stream object.

The `cin` Stream

- `cin` is the character input stream object.
- User input can be read into a variable using the **extraction operator** ».

The `cin` Stream

- `cin` is the character input stream object.
- User input can be read into a variable using the **extraction operator** ».
- For example:
  `cin » x;`
  would allow the user to enter an integer which is then stored in `x`.

Example: multiple_choice.cpp

Compile and run this program (found in your examples folder)

```
#include <iostream>

using namespace std;

int main()
{
    char choice;  //The choice made by the user

    //Get the user's choice
    cout << "In my opinion, computer programming is _______." << endl
         << "\tA) the best part of my day" << endl
         << "\tB) what gives me a sense of purpose" << endl
         << "\tC) how I scream into the void" << endl
         << endl
         << "Your Choice? ";
    cin >> choice;

    //report the user's choice
    cout << "You chose " << choice << "." << endl;
}
```

Outline

Overview

Over the course of this class, we will develop an application which manages a stock portfolio. It will allow us to:

- Buy Stocks

This program was inspired by a project found in *Complete C Language Programming for the IBM PC* by Douglas A. Troy (1986)

Overview

Over the course of this class, we will develop an application which manages a stock portfolio. It will allow us to:

- Buy Stocks
- Sell Stocks

This program was inspired by a project found in *Complete C Language Programming for the IBM PC* by Douglas A. Troy (1986)

Overview

Over the course of this class, we will develop an application which manages a stock portfolio. It will allow us to:

- Buy Stocks
- Sell Stocks
- Run Reports

This program was inspired by a project found in *Complete C Language Programming for the IBM PC* by Douglas A. Troy (1986)

Overview

Over the course of this class, we will develop an application which manages a stock portfolio. It will allow us to:

- Buy Stocks
- Sell Stocks
- Run Reports
- Store Stock Data in a File

This program was inspired by a project found in *Complete C Language Programming for the IBM PC* by Douglas A. Troy (1986)

Main Menu

Write a program `stock.cpp` which displays the main menu of the stock portfolio system and reads a user's choice.

```
$ ./stock
        Stock Portfolio Management System
                Please Make a Selection
        1 -- Buy a Stock
        2 -- Sell a Stock
        3 -- Report Current Holdings
        4 -- Report Gains and Losses
        5 -- Remove a Current Holding
        6 -- Done!  (quit)

        Choice? 6
```

Finishing Up

- Make sure you have the following files in `cs1-fall2019-`*`username`*`/labs/week2`
  - `hello.cpp`
  - `diamond.cpp`
  - `stock.cpp`
- Make sure you have `boilerplate.cpp` in your `cs1-fall2019-`*`username`* directory.
- These programs must all be in working order to receive full credit for the week!
- `git add -A`
- `git push`
- `git commit -a -m 'Finished Week2!'`

Maryville COLLEGE