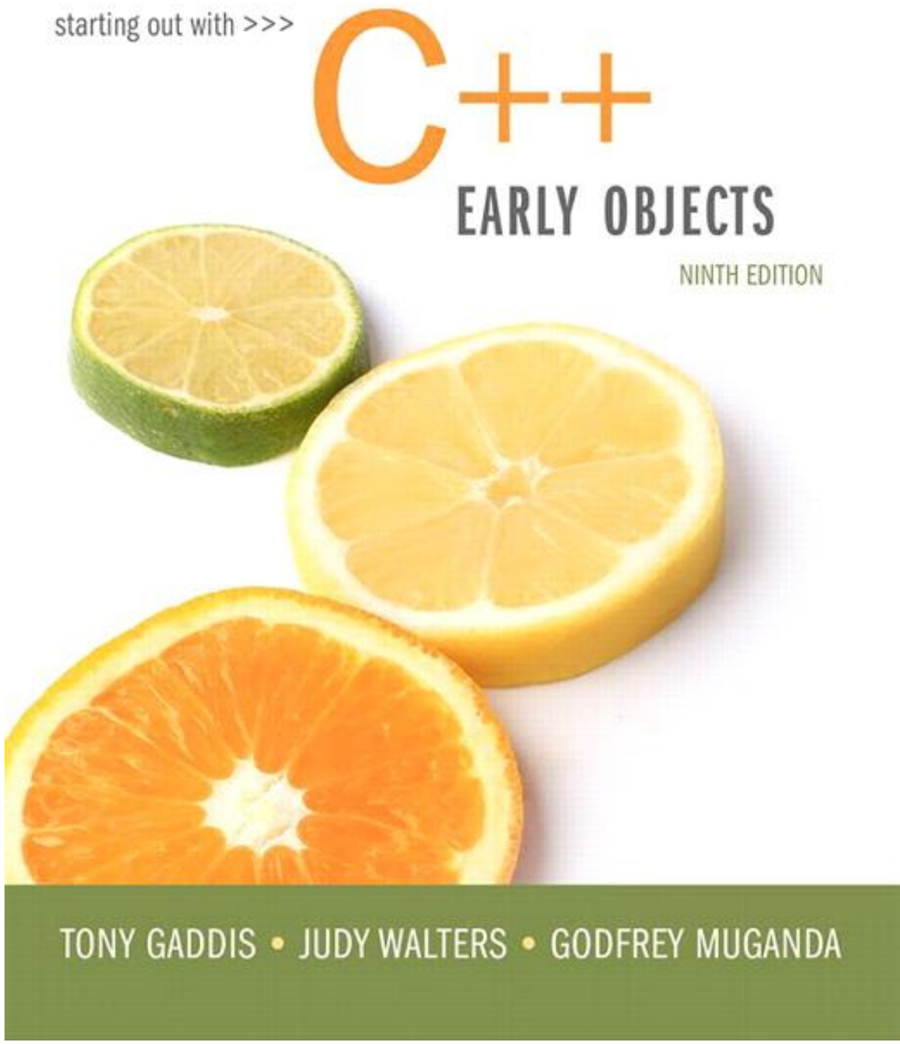
Week 1 Slides:

Introduction

to

C++



Topics

2.1 The Parts of a C++ Program

```
// sample C++ program          ←——  comment
#include <iostream>            ←——  preprocessor directive
using namespace std;           ←——  which namespace to use
int main()                     ←——  beginning of function named main
{                              ←——  beginning of block for main
    cout << "Hello, there!";   ←——  output statement
    return 0;                  ←——  send 0 back to operating system
}                              ←——  end of block for main
```

2.1 The Parts of a C++ Program

Statement	Purpose
`// sample C++ program`	comment
`#include <iostream>`	preprocessor directive
`using namespace std;`	which namespace (set of names) to use
`int main()`	beginning of function named `main`
`{`	beginning of block for `main`
`    cout << "Hello, there!";`	output statement
`    return 0;`	send 0 back to the operating system
`}`	end of block for `main`

Special Characters

Character	Name	Description
`//`	Double Slash	Begins a comment
`#`	Pound Sign	Begins preprocessor directive
`< >`	Open, Close Brackets	Encloses filename used in `#include` directive
`( )`	Open, Close Parentheses	Used when naming a function
`{ }`	Open, Close Braces	Encloses a group of statements
`" "`	Open, Close Double Quote Marks	Encloses a string of characters
`;`	Semicolon	Ends a programming statement

Important Details

- C++ is case-sensitive. Uppercase and lowercase characters are different characters. '`Main`' is not the same as '`main`'.
- Every `{` must have a corresponding `}`, and vice-versa.

2.2 The **`cout`** Object

- Displays information on computer screen
- Use **<<** to send information to cout

```
cout << "Hello, there!";
```

- You can use **<<** to send multiple items to cout

```
cout << "Hello, " << "there!";
```

Or

```
cout << "Hello, ";
cout << "there!";
```

Starting a New Line

- To get multiple lines of output on screen
  - Use **`endl`**

    ```
    cout << "Hello, there!" << endl;
    ```

  - Use **`\n`** in an output string

    ```
    cout << "Hello, there!\n";
    ```

Escape Sequences – More Control Over Output

Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing.
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop.
`\a`	Alarm	Causes the computer to beep.
`\b`	Backspace	Causes the cursor to back up, or move left one position.
`\r`	Return	Causes the cursor to go to the beginning of the current line, not the next line.
`\\`	Backslash	Causes a backslash to be printed.
`\'`	Single quote	Causes a single quotation mark to be printed.
`\"`	Double quote	Causes a double quotation mark to be printed.

Common Escape Sequence Mistakes

1) Don’t confuse `"\"` (a back slash) and `"/"` (a forward slash)
2) Remember to put **`\n`** in double quotation marks

2.3 The **`#include`** Directive

- Inserts the contents of another file into the program
- It is a preprocessor directive
  - Not part of the C++ language
  - Not seen by compiler
- Example:

```
#include <iostream>
```

No ; goes here

2.4 Variables and the Assignment Statement

A Variable

- Is used to refer to a location in memory where a value can be stored.
- An assignment statement is used to store a value.
- The value that is stored can be changed, *i.e.*, it can “vary”.
- You must define the variable (indicate the name and the type of value that it can hold) before you can use it to store a value.

Variables

- If a new value is stored in the variable, it replaces the previous value
- The previous value is overwritten and can no longer be retrieved

```
int age;
age = 17;      // Assigns 17 to age
cout << age;   // Displays 17
age = 18;      // Now age is 18
cout << age;   // Displays 18
```

Assignment Statement

- Uses the **=** operator
- Has a single variable on the left side and a value on the right side
- Copies the value on the right into the location in memory that is associated with the variable on the left

```
    item = 12;
```

2.5 Literals

A Literal is a piece of data that is written directly in the source code of the program.

```
'A'        // character literal
"Hello"    // string literal
12         // integer literal
"12"       //  string literal (yes!)
3.14       // floating-point literal
```

2.6 Identifiers

- Programmer-chosen names to represent parts of the program, such as variables
- Name should indicate the use of the identifier
- Cannot use C++ key words as identifiers
- Must begin with alphabetic character or _, followed by any number of alphabetic, numeric, or _ characters.
- Alphabetic characters may be upper- or lowercase

Multi-word Variable Names

- A variable name should reflect its purpose
- Descriptive variable names may include multiple words
- Two conventions to use in naming variables:
  - Capitalize all words but the first letter of first word. Run words together:

    ```
    quantityOnOrder          totalSales
    ```

  - Use the underscore _ character as a space:

    ```
    quantity_on_order        total_sales
    ```

- Use one convention consistently throughout a program

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_sales`	Yes	
`total.Sales`	No	Cannot contain period
`4thQtrSales`	No	Cannot begin with digit
`total$Sales`	No	Cannot contain $

2.7 Integer Data Types

- Designed to hold whole (non-decimal) numbers
- Can be **`signed`** or **`unsigned`**

  `12        -6        +3`
- Available in different sizes (*i.e.*, number of bytes): **`short int`**, **`int`**, **`long int`**, and **`long long int`**
- **`long long int`** was introduced in C++ 11.

Signed vs. Unsigned Integers

- C++ allocates one bit for the sign of the number. The rest of the bits are for data.
- If your program will never need negative numbers, you can declare variables to be **`unsigned`**. All bits in unsigned numbers are used for data.
- A variable is signed unless the **`unsigned`** keyword is used at variable definition.

Defining Variables

- Variables of the same type can be defined
  - In separate statements

```
int length;
int width;
```

  - In the same statement

```
int length,
    width;
```

- Variables of different types must be defined in separate statements

Abbreviated Variable Definitions

- **`int`** can be omitted from a variable definition for any datatype except an **`int`** itself.
- Examples:

```
short temperatures;
unsigned short booksOnOrder;
unsigned long long magnitude;
int grades;
```

Integral Literals

- To store an integer literal in a long memory location, put '**`L`**' at the end of the number:
  `long rooms = 234L;`
- Use '**`LL`**' at the end to put an integer literal in a long long memory location.
- Literals that begin with '**`0`**' (zero) are octal, or base 8: **`075`**
- Literals that begin with '**`0x`**' are hexadecimal, or base 16: **`0x75A`**

2.8 Floating-Point Data Types

- Designed to hold real numbers

  `12.45` **`-3.8`**

- Stored in a form similar to scientific notation
- Numbers are all signed
- Available in different sizes (number of bytes): **`float`**, **`double`**, and **`long double`**
- Size of **`float`** ≤ size of **`double`**
  ≤ size of **`long double`**

Floating-point Literals

- Can be represented in
  - Fixed point (decimal) notation:

    `31.4159          0.0000625`

  - E notation:

    `3.14159E1        6.25e-5`

- Are **`double`** by default
- Can be forced to be float **`3.14159F`** or long double **`0.0000625L`**

Assigning Floating-point Values to Integer Variables

If a floating-point value (a literal or a variable) is assigned to an integer variable

- The fractional part will be truncated (*i.e.*, "chopped off" and discarded)
- The value is not rounded

```
int rainfall = 3.88;
cout << rainfall;  // Displays 3
```