Lecture Notes on Jan/26/2023

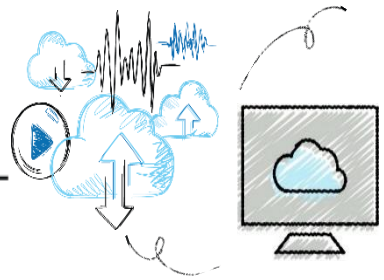
Chapters 2 and 3

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Please see the slide number 37.



Allocating Memory with Constants and Variables (1 of2)

- Named constant: memory location whose content cannot change during execution
- Syntax to declare a named constant

```
const dataType identifier = value;
```

- In C++, `const` is a reserved word

EXAMPLE 2-11

Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

Allocating Memory with Constants and Variables (2 of 2)

- Variable: memory location whose content may change during execution
- Syntax to declare one or multiple variables

```
dataType identifier, identifier, . . .;
```

EXAMPLE 2-12

Consider the following statements:

```
double amountDue;
int counter;
char ch;
int x, y;
string name;
```

Putting Data into Variables

- Ways to place data into a variable
  - Use C++'s assignment statement
  - Use input (read) statements

Assignment Statement (1 of 4)

- The assignment statement takes the form:

```
variable = expression;
```

- Expression is evaluated and its value is assigned to the variable on the left side
- A variable is said to be initialized the first time a value is placed into it
- In C++, = is called the assignment operator

Assignment Statement (2 of 4)

EXAMPLE 2-13

Suppose you have the following variable declarations:

```
int num1, num2;
double sale;
char first;
string str;
```

Now consider the following assignment statements:

```
num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```

Assignment Statement (3 of 4)

- Example 2-14 illustrates a walk-through (tracing values through a sequence)

	Values of the Variables/Statement	Explanation
Before Statement 1	num1: ? num2: ? num3: ?	
After Statement 1	num1: 18 num2: ? num3: ? `num1 = 18;`	
After Statement 2	num1: 45 num2: ? num3: ? `num1 = num1 + 27;`	**num1 + 27 = 18 + 27 = 45.** This value is assigned to **num1**, which replaces the old value of **num1**.
After Statement 3	num1: 45 num2: 45 num3: ? `num2 = num1;`	Copy the value of **num1** into **num2**.
After Statement 4	num1: 45 num2: 45 num3: 9 `num3 = num2 / 5;`	**num2 / 5 = 45 / 5 = 9.** This value is assigned to **num3**. So **num3 = 9.**
After Statement 5	num1: 45 num2: 45 num3: 2 `num3 = num3 / 4;`	**num3 / 4 = 9 / 4 = 2.** This value is assigned to num3, which replaces the old value of **num3**.

Assignment Statement (4 of 4)

- Given `int` variables `x`, `y`, and `z`. How is this legal C++ statement evaluated?

```
x = y = z
```

- The assignment operator is evaluated from right to left
  - The associativity of the assignment operator is from right to left

Saving and Using the Value of an Expression

- Declare a variable of the appropriate data type
- Assign the value of the expression to the variable that was declared
  - Use the assignment statement
- Wherever the value of the expression is needed, use the variable holding the value

Declaring and Initializing Variables

- Not all types of variables are initialized automatically
- Variables can be initialized when declared:

```
int first = 13, second = 10;
char ch = ' ';
double x = 12.6;
```

- All variables must be initialized before they are used
  - But not necessarily during declaration

Input (Read) Statement (1 of 3)

- **`cin`** is used with **`>>`** to gather one or more inputs

```
cin >> variable >> variable ...;
```

- This is called an input (read) statement
- The stream extraction operator is **`>>`**
- For example, if miles is a **`double`** variable:
  `cin >> miles;`
  - Causes the computer to get a value of type double and places it in the variable **`miles`**

CENGAGE Learning

Input (Read) Statement (2 of 3)

- Using more than one variable in **`cin`** allows more than one value to be read at a time
- Example: if **`feet`** and **`inches`** are variables of type **`int`**, this statement:

```
cin >> feet >> inches;
```

  - Inputs two integers from the keyboard
  - Places them in variables **`feet`** and **`inches`** respectively

Input (Read) Statement (3 of 3)

EXAMPLE 2-17

Suppose we have the following statements:

```
int feet;
int inches;
```

Suppose the input is:

```
23 7
```

Next, consider the following statement:

```
cin >> feet >> inches;
```

Increment and Decrement Operators

- Increment operator (++): increase variable by 1
  - Pre-increment: **++variable**
  - Post-increment: **variable++**
- Decrement operator: (--) decrease variable by 1
  - Pre-decrement: **--variable**
  - Post-decrement: **variable--**
- What is the difference between the following?

```
x = 5;
y = ++x;
```

```
x = 5;
y = x++;
```

CENGAGE Learning

Output (1 of 4)

- The syntax of **`cout`** and **<<** is:

```
cout << expression or manipulator << expression or manipulator...;
```

  - Called an output statement
- The stream insertion operator is **<<**
- Expression evaluated and its value is printed at the current cursor position on the screen

Output (2 of 4)

- A manipulator is used to format the output
  - Example: **`endl`** causes the insertion point to move to beginning of next line

EXAMPLE 2-21

Consider the following statements. The output is shown to the right of each statement.

	Statement	Output
1	`cout << 29 / 4 << endl;`	`7`
2	`cout << "Hello there." << endl;`	`Hello there.`
3	`cout << 12 << endl;`	`12`
4	`cout << "4 + 7" << endl;`	`4 + 7`
5	`cout << 4 + 7 << endl;`	`11`
6	`cout << 'A' << endl;`	`A`
7	`cout << "4 + 7 = " << 4 + 7 << endl;`	`4 + 7 = 11`
8	`cout << 2 + 3 * 5 << endl;`	`17`
9	`cout << "Hello \nthere." << endl;`	`Hello` `there.`

Output (3 of 4)

- The new line character (new line escape sequence) is **'\n'**
  - May appear anywhere in the string
- Examples

```
cout << "Hello there.";
cout << "My name is James.";
```

Output:

```
Hello there.My name is James.
```

```
cout << "Hello there.\n";
cout << "My name is James.";
```

Output :

```
Hello there.
My name is James.
```

Output (4 of 4)

TABLE 2-4 Commonly Used Escape Sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

Preprocessor Directives (1 of 2)

- C++ has a small number of operations
- Many functions and symbols needed to run a C++ program are provided as collection of libraries
- Every library has a name and is referred to by a header file
- Preprocessor directives are processed by the preprocessor program
- All preprocessor commands begin with **#**
- No semicolon is placed at the end of these commands

Preprocessor Directives (2 of 2)

- Syntax to include a header file

```
#include <headerFileName>
```

- For example:

```
#include <iostream>
```

  - Causes the preprocessor to include the header file **iostream** in the program

- Preprocessor commands are processed before the program goes through the compiler

namespace and Using cin and cout in a Program

- **cin** and **cout** are declared in the header file **iostream**, but within **std namespace**
- To use **cin** and **cout** in a program, use the following two statements:

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

Using the `string` Data Type in a Program

- To use the **`string`** type, you need to access its definition from the header file `string`
- Include the following preprocessor directive:

```
#include <string>
```

Creating a C++ Program (1 of 3)

- A C++ program is a collection of functions, one of which is the function **`main`**
- The syntax of the function **`main`** used in this book has this form:

```
int main()
{
    statement_1
        .
        .
        .
    statement_n

    return 0;
}
```

Creating a C++ Program (2 of 3)

- Source code is comprised of preprocessor directives and program statements
- The source code file (source file) contains the source code
- The compiler generates the object code (file extension **`.obj`**)
- Executable code (file extension **`.exe`**) results when object code is linked with the system resources
- The first line of the function **`main`** is called the heading of the function:

```
int main()
```

Creating a C++ Program (3 of 3)

- The statements enclosed between the curly braces (**{** and **}**) form the body of the function
- A C++ program contains two types of statements:
  - Declaration statements declare things, such as variables
  - Executable statements perform calculations, manipulate data, create output, accept input, etc.

CENGAGE Learning

Debugging: Understanding and Fixing Syntax Errors (1 of 2)

- Sample program with line numbers added on the left

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      int num
8.
9.      num = 18;
10.
11.     tempNum = 2 * num;
12.
13.     cout << "Num = " << num << ", tempNum = " < tempNum << endl;
14.
15.     return ;
16. }
```

Debugging: Understanding and Fixing Syntax Errors (2 of 2)

- Compile the program
  - Compiler will identify the syntax errors
  - The line numbers where the errors occur are specified:

```
ExampleCh2_Syntax_Errors.cpp
c:\examplech2_syntax_errors.cpp(9): error C2146: syntax error:
missing ';' before identifier 'num'
c:\examplech2_syntax_errors.cpp(11): error C2065: 'tempNum':
undeclared identifier
```

Program Style and Form: Syntax

- Syntax rules indicate what is legal and what is not legal
- Errors in syntax are found in compilation

```
int x;        //Line 1
int y         //Line 2
double z;     //Line 3

y = w + x;    //Line 4
```

- Compilation errors would occur at:
  - Line 2 (missing semicolon)
  - Line 4 (identifier **w** used but not declared)

Use of Blanks

- In C++, you use one or more blanks to separate numbers when data is input
- Blanks are also used to separate reserved words and identifiers from each other and from other symbols
- Blanks must never appear within a reserved word or identifier

Use of Semicolons, Brackets, and Commas

- All C++ statements end with a semicolon
  - Also called a statement terminator
- { and } are not C++ statements
  - Can be regarded as delimiters
- Commas separate items in a list
  - Declaring more than one variable following a data type

Semantics

- Semantics: set of rules that gives meaning to a language
  - Possible to remove all syntax errors in a program and still not have it run
  - Even if it runs, it may still not do what you meant it to do
- Example: `2 + 3 * 5` and `(2 + 3) * 5`
  - Both are syntactically correct expressions but have different meanings

Naming Identifiers

- Identifiers can be self-documenting
  - **`CENTIMETERS_PER_INCH`**
- Avoid run-together words
  - **`annualsale`**
- Solutions may include:
  - Capitalizing the beginning of each new word: **`annualSale`**
  - Inserting an underscore just before a new word: **`annual_sale`**

CENGAGE Learning

Prompt Lines

- Prompt lines: executable statements that inform the user what to do

```
cout << "Please enter a number between 1 and 10 and "
     << "press the return key" << endl;
cin >> num;
```

- Always include prompt lines when input is needed from users

CENGAGE Learning

Documentation

- A well-documented program is easier to understand and modify
- You use comments to document programs
- Comments should appear in a program to:
  - Explain the purpose of the program
  - Identify who wrote it
  - Explain the purpose of particular statements

Form and Style

- Consider two ways of declaring variables:
  - Method 1

```
int feet, inches;
double x, y;
```

  - Method 2

```
int feet,inches; double x,y;
```

- Both are correct; however, the second is harder to read

More on Assignment Statements

- Two forms of assignment
  - Simple and compound
  - Compound operators provide more concise notation
- Compound operators: **+=**, **-=**, ***=**, **/=**, **%=**
- Simple assignment statement example

```
x = x * y;
```

- Compound assignment statement example

```
x *= y;
```

Reading Assignment – Very Important for "GU – ECE 111"

- Malik, D.S., 2014. **C++ programming: Program design including data structures**. Cengage Learning.
  - "**Chapter 1**: **An Overview of Computers and Programming Languages**".
  - "**Chapter 2**: **Basic Elements of C++**".

Chapter 3 - Objectives (1 of 2)

- In this chapter, you will:
  - Learn what a stream is and examine input and output streams
  - Explore how to read data from the standard input device
  - Learn how to use predefined functions in a program
  - Explore how to use the input stream functions **`get`**, **`ignore`**, **`putback`**, and **`peek`**

Chapter 3 - Objectives (2 of 2)

- Become familiar with input failure
- Learn how to write data to the standard output device
- Discover how to use manipulators in a program to format output
- Learn how to perform input and output operations with the **`string`** data type
- Learn how to debug logic errors
- Become familiar with file input and output

CENGAGE Learning

I/O Streams and Standard I/O Devices (1 of 3)

- I/O: sequence of bytes (stream of bytes) from source to destination
  - Bytes are usually characters, unless program requires other types of information
  - Stream: sequence of characters from the source to the destination
  - Input stream: sequence of characters from an input device to the computer
  - Output stream: sequence of characters from the computer to an output device

I/O Streams and Standard I/O Devices (2 of 3)

- Use **`iostream`** header file to receive data from keyboard and send output to the screen
  - Contains definitions of two data types:
    - **`istream`**: input stream
    - **`ostream`**: output stream
  - Has two variables:
    - **`cin`**: stands for common input
    - **`cout`**: stands for common output

I/O Streams and Standard I/O Devices (3 of 3)

- Variable declaration is similar to:
  - `istream cin;`
  - `ostream cout;`
- To use **`cin`** and **`cout`**, the preprocessor directive **`#include <iostream>`** must be used
- Input stream variables: type **`istream`**
- Output stream variables: type **`ostream`**

CENGAGE Learning

cin and the Extraction Operator >> (1 of 7)

- The syntax of an input statement using **cin** and the extraction operator **>>** is

```
cin >> variable >> variable...;
```

- The extraction operator **>>** is binary
  - Left-side operand is an input stream variable
    - Example: **cin**
  - Right-side operand is a variable

CENGAGE Learning

cin and the Extraction Operator >> (2 of 7)

- No difference between a single **cin** with multiple variables and multiple **cin** statements with one variable in each statement

```
cin >> payRate >> hoursWorked;

cin >> payRate;
cin >> hoursWorked;
```

- When scanning, **>>** skips all whitespace
  - Blanks and certain nonprintable characters
- **>>** distinguishes between character **2** and number **2** by the right-side operand of **>>**
  - If type **char** or **int** (or **double**), the **2** is treated as a character or as a number **2**, respectively

cin and the Extraction Operator >> (3 of 7)

TABLE 3-1 Valid Input for a Variable of the Simple Data Type

Data Type of a	Valid Input for a
char	One printable character except the blank.
int	An integer, possibly preceded by a + or - sign.
double	A decimal number, possibly preceded by a + or - sign. If the actual data input is an integer, the input is converted to a decimal number with the zero decimal part.

- Entering a **char** value into an **int** or **double** variable causes serious errors, called input failure

cin and the Extraction Operator >> (4 of 7)

- When reading data into a **char** variable
  - **>>** skips leading whitespace, finds and stores only the next character
  - Reading stops after a single character
- To read data into an **int** or **double** variable
  - **>>** skips leading whitespace, reads **+** or **−** sign (if any), reads the digits (including decimal for floating-point variables)
  - Reading stops on whitespace or a non-digit character

cin and the Extraction Operator >> (5 of 7)

EXAMPLE 3-1

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> ch;`	`A`	`ch = 'A'`
2	`cin >> ch;`	`AB`	`ch = 'A', 'B'` is held for later input
3	`cin >> a;`	`48`	`a = 48`
4	`cin >> a;`	`46.35`	`a = 46, .35` is held for later input
5	`cin >> z;`	`74.35`	`z = 74.35`
6	`cin >> z;`	`39`	`z = 39.0`
7	`cin >> z >> a;`	`65.78 38`	`z = 65.78, a = 38`
8	`cin >> a >> b;`	`4 60`	`a = 4, b = 60`
9	`cin >> a >> z;`	`46 32.4 68`	`a = 46, z = 32.4, 68` is held for later input

CENGAGE Learning

cin and the Extraction Operator >> (6 of 7)

EXAMPLE 3-2

Suppose you have the following variable declarations:

```
int a;
double z;
char ch;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> a >> ch >> z;`	`57 A 26.9`	`a = 57, ch = 'A', z = 26.9`
2	`cin >> a >> ch >> z;`	`57 A` `26.9`	`a = 57, ch = 'A', z = 26.9`

3	`cin >> a >> ch >> z;`	`57`
		`A`
		`26.9`
			`a = 57, ch = 'A', z = 26.9`

4	`cin >> a >> ch >> z;`	`57A26.9`	`a = 57, ch = 'A', z = 26.9`

cin and the Extraction Operator >> (7 of 7)

EXAMPLE 3-3

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch, ch1, ch2;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> z >> ch >> a;`	`36.78B34`	`z = 36.78, ch = 'B', a = 34`
2	`cin >> z >> ch >> a;`	`36.78` `B34`	`z = 36.78, ch = 'B', a = 34`
3	`cin >> a >> b >> z;`	`11 34`	`a = 11, b = 34,` computer waits for the next number
4	`cin >> a >> z;`	`78.49`	`a = 78, z = 0.49`
5	`cin >> ch >> a;`	`256`	`ch = '2', a = 56`
6	`cin >> a >> ch;`	`256`	`a = 256,` computer waits for the input value for `ch`
7	`cin >> ch1 >> ch2;`	`A B`	`ch1 = 'A', ch2 = 'B'`