

COSC 120: Computer Science I

Module 1

Instructor:

Dr. Xiaohong (Sophie) Wang
(xswang@salisbury.edu)

Department of Computer Science

Salisbury University

Spring 2024



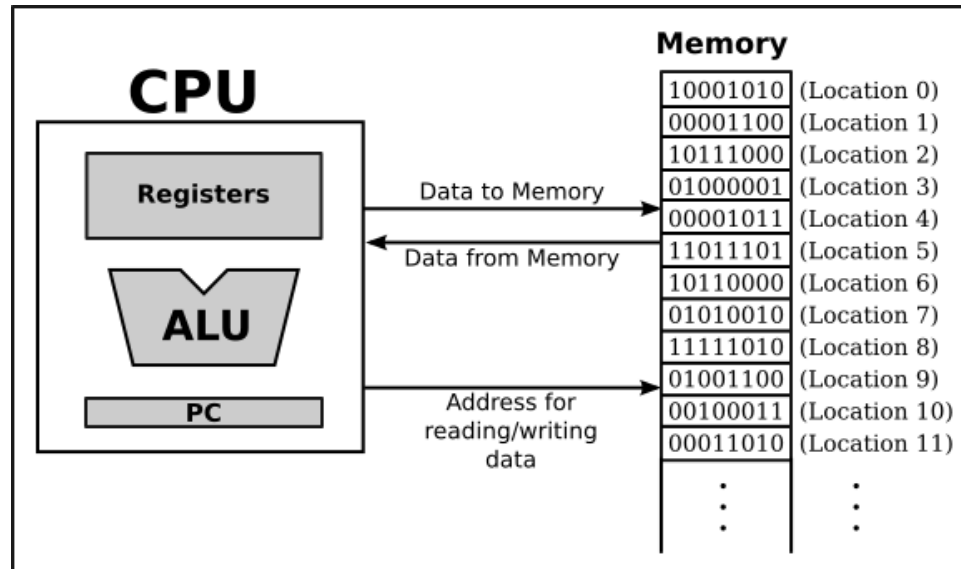
Content

- Introduction to fundamental computer and programming concepts
- Introduction to the C++ Programming Language
- Expressions, Input, Output and Data Type Conversions

Fundamental Computer & Programming Concepts

1. Von Neumann computer architecture and the Fetch-and-Execute Cycle
2. Language compilation process
3. Problem solving process
4. Variables

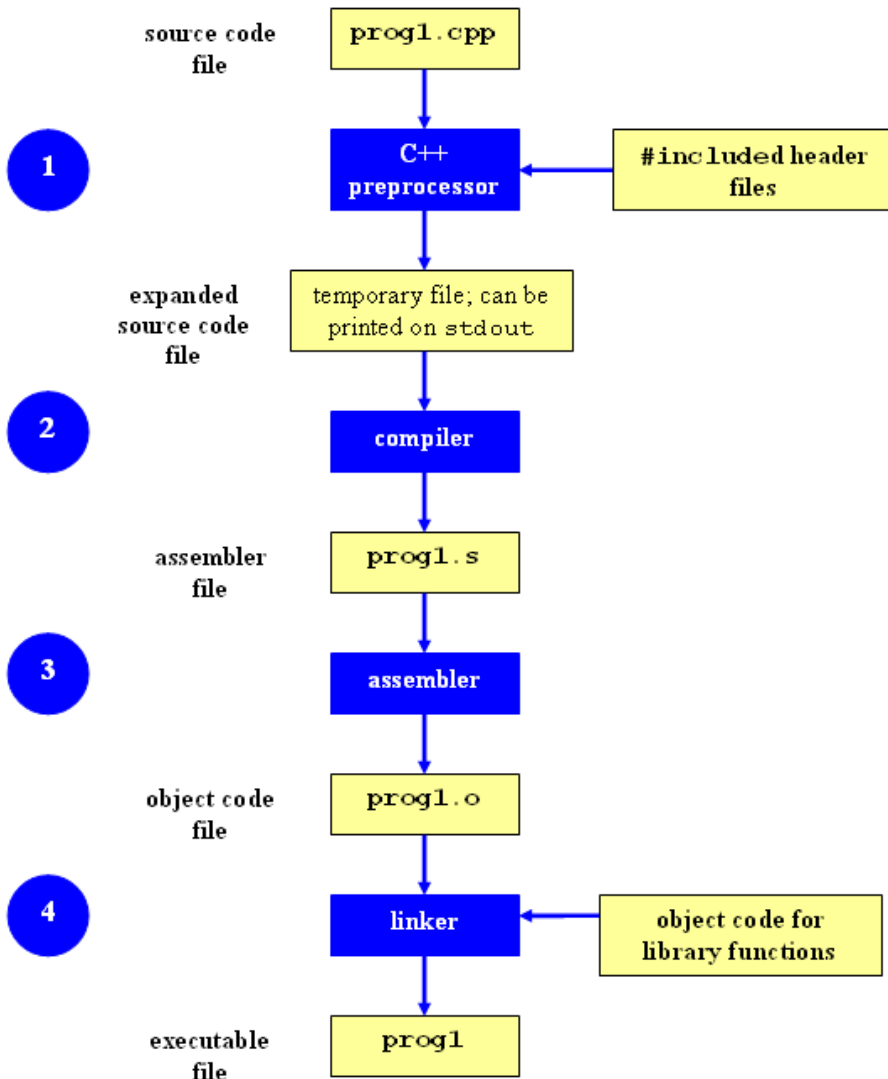
1. Von Neumann Computer Architecture and Fetch-and-Execute Cycle



The fetch-and-execute cycle:

- CPU executes a program that is stored as a sequence of machine language instructions in main memory.
- It does this by reading/fetching an instruction from memory, load it into registers, and then carrying out/executing, that instruction.
- This "fetch and execute" process repeats until all instructions of a program finish.

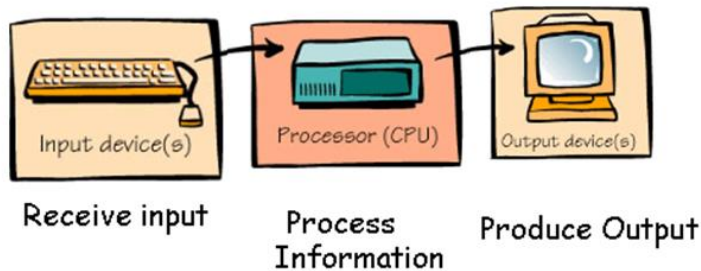
2. Language Compilation Process



- Most programs written in higher level programming languages: Java, C++, Python
- They need to be translated into machine language programs so computer can understand
- Compilation is a sequence of steps that processes statements written in a higher programming language and turns them into machine language code.

3. Problem Solving Process

What Computers Do



- A **computer can** be programmed to accept data (**input**), **process** it into useful information (**output**).
- **input, process and output** actions are controlled by programs (software) and executed by hardware (cpu, etc.).
- When we write programs, we need to gather input, process it and produce output.
- Information being processed should be stored in the memory.

4. Variables



- Information being read in and being processed are stored in memory.
- Variables are memory used to keep information such as numbers or words.
- Variables have types, names and sizes.

Introduction to C++ Programming Language

1. What is C++ language?
2. Variables
3. Data types
4. Input/output (Formatted output)
5. Arithmetic operator
6. ++ and -- operators
7. C++ library

- Partial contents of this note refer to <https://www.pearson.com/us/>
- Copyright 2018, 2015, 2012, 2009 Pearson Education, Inc., All rights reserved
- Dissemination or sale of any part of this note is NOT permitted

1. C++ language

- A cross-platform language that can be used to create high-performance applications
- An extension to the C language
- One of the world's most popular programming languages
- An object-oriented programming (OOP) language
- Give programmers a high level of control over system resources and memory

Advantage of C++

- C++ is relatively-low level and is a system programming language
- It has a large community
- It has a relatively clear and mature standard
- Modularity
- Reusability and readability

Disadvantage of C++

- Can't support garbage collection
- Not secure because it has a pointer, friend function, and global variable

Example 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 to operating system
} ← end of block for main
```

Special Characters

Character	Name	Meaning
//	Double slash	Beginning of a comment
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
()	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Open/close quotation marks	Encloses string of characters
;	Semicolon	End of a programming statement

2. Variables

- Variable: a storage location in memory
 - Has a **name** and a **type** of data it can hold
 - Must be defined before it can be used

Program 2-7

```
1  // This program has a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int number;
8
9      number = 5;
10     cout << "The value in number is " << number << endl;
11     return 0;
12 }
```

Variable Definition

Program Output

The value in number is 5

Identifier

- An identifier is a programmer-defined **name** for some part of a program: variables, functions, etc.
- How to define an identifier?
 - Should be representative.
 - E.g. $\pi = 3.14$
 - Follow the following rules:
 - A combination of alphabets, numbers and underscores
 - Not start with number
 - Can **NOT** use any of the C++ key words

Note: C++ is a case-sensitive language. So does the identifier name.

Question: What do key words mean? Why do we need them?

C++ Key Words

Table 2-4 The C++ Key Words

<code>alignas</code>	<code>const</code>	<code>for</code>	<code>private</code>	<code>throw</code>
<code>alignof</code>	<code>constexpr</code>	<code>friend</code>	<code>protected</code>	<code>true</code>
<code>and</code>	<code>const_cast</code>	<code>goto</code>	<code>public</code>	<code>try</code>
<code>and_eq</code>	<code>continue</code>	<code>if</code>	<code>register</code>	<code>typedef</code>
<code>asm</code>	<code>decltype</code>	<code>inline</code>	<code>reinterpret_cast</code>	<code>typeid</code>
<code>auto</code>	<code>default</code>	<code>int</code>	<code>return</code>	<code>typename</code>
<code>bitand</code>	<code>delete</code>	<code>long</code>	<code>short</code>	<code>union</code>
<code>bitor</code>	<code>do</code>	<code>mutable</code>	<code>signed</code>	<code>unsigned</code>
<code>bool</code>	<code>double</code>	<code>namespace</code>	<code>sizeof</code>	<code>using</code>
<code>break</code>	<code>dynamic_cast</code>	<code>new</code>	<code>static</code>	<code>virtual</code>
<code>case</code>	<code>else</code>	<code>noexcept</code>	<code>static_assert</code>	<code>void</code>
<code>catch</code>	<code>enum</code>	<code>not</code>	<code>static_cast</code>	<code>volatile</code>
<code>char</code>	<code>explicit</code>	<code>not_eq</code>	<code>struct</code>	<code>wchar_t</code>
<code>char16_t</code>	<code>export</code>	<code>nullptr</code>	<code>switch</code>	<code>while</code>
<code>char32_t</code>	<code>extern</code>	<code>operator</code>	<code>template</code>	<code>xor</code>
<code>class</code>	<code>false</code>	<code>or</code>	<code>this</code>	<code>xor_eq</code>
<code>compl</code>	<code>float</code>	<code>or_eq</code>	<code>thread_local</code>	

Note: Editors or IDEs may help you to mark the key words in a specific color

3. C++ data types

- int: Integer
- float: Floating-point
- char: Individual character
- string: 0, 1 or more characters
- bool: Boolean value: true or false

3.1 Integer

- Integer variables hold whole numbers

Table 2-6 Integer Data Types

Data Type	Typical Size	Typical Range
<code>short int</code>	2 bytes	−32,768 to +32,767
<code>unsigned short int</code>	2 bytes	0 to +65,535
<code>int</code>	4 bytes	−2,147,483,648 to +2,147,483,647
<code>unsigned int</code>	4 bytes	0 to 4,294,967,295
<code>long int</code>	4 bytes	−2,147,483,648 to +2,147,483,647
<code>unsigned long int</code>	4 bytes	0 to 4,294,967,295
<code>long long int</code>	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>unsigned long long int</code>	8 bytes	0 to 18,446,744,073,709,551,615

Note: The exact sizes of these data types are dependent on the computer system you are using

Example:

```
int length, width = 5;  
unsigned int area;
```

Same type can be defined on the same line
Different types must be in different definitions

3.2 Floating-point

- Can hold real numbers:

- Fixed point (decimal) notation:

31.4159

0.0000625

- E notation:

3.14159E1

6.25e-5

- Three floating-point data types

float

double

long double

- Are `double` by default

Assign floating-point value to integer variable

- The fractional part of the value is discarded

```
int number;
```

```
number = 6.9; //assign 6 to number
```

- What is the output of the following code?

```
int num;
```

```
num = 6.9999999999999999999999999999;
```

```
cout << "The output is: " << num;
```

The output is: 7

Note: Floating-point value has limited precision. WHY?

3.3 char

- Used to hold one character
- Usually 1 byte of memory
 - Question: how many different characters C++ can store at most?
- A character must be enclosed in **single quote marks**
 - Note: double quote marks enclose a string
- **Numeric value** of character from the character set is stored in memory:

CODE:

```
char letter;  
letter = 'C';
```

MEMORY:

letter

67

For numeric value of a character, please refer to Appendix A: The ASCII Character Set

3.4 string

- A series of characters in consecutive memory locations:
 "Hello"
- Stored with the null terminator, **\0**, at the end:
- Comprised of the characters between the **double quote marks " "**

H	e	l	l	o	\0
---	---	---	---	---	-----------

The C++ `string` Class

- Special data type supports working with strings

```
#include <string>
```

- Can define `string` variables in programs:

```
string firstName, lastName;
```

- Can receive values with assignment operator:

```
firstName = "George";
```

```
lastName = "Washington";
```

- Can be displayed via `cout`

```
cout << firstName << " " << lastName;
```

String length and concatenation

- Use `length()` to find the length of a string:

```
string state = "Texas";  
int size = state.length();
```

- To concatenate (join) multiple strings:

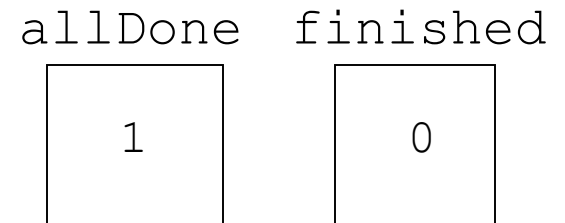
```
greeting2 = greeting1 + name1;  
greeting1 = greeting1 + name2;
```

3.5 bool

- Represents values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:

```
bool allDone = true;
```

```
bool finished = false;
```



Determining the Size of a Data Type

- The `sizeof` operator reports the **number of bytes of memory** used by any data type:

```
double amount;  
cout << "A double is stored in "  
      << sizeof(double) << " bytes\n";  
cout << "Variable amount is stored in "  
      << sizeof(amount)  
      << " bytes\n";
```

Constant variable

- Constant variable (named constant): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;  
const int NUM_STATES = 50;
```
- Often named in uppercase letters

3.6 Initialization and Assignment

- To initialize a variable means to assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```

Initialization and Assignment

- An assignment statement uses the **=** operator to store a value in a variable.

```
item = 12;
```

➤ This statement assigns the value 12 to the `item` variable.

- The variable receiving the value must appear on the left side of the **=** operator.
- This will NOT work:

```
12 = item;      // ERROR!
```

4. Input/output

- `cout` object
 - Display information on the computer's screen
- `cin` object
 - Read data typed at the keyboard

4.1 The `cout` Object

- Need `iostream` file, i.e. `#include <iostream>`
- Use the stream insertion operator `<<` to send one or more items to `cout` then display:

```
cout << "Programming is fun!";
```

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

- Use the `endl` manipulator to start a new line

```
cout << "Programming is" << endl;
```

```
cout << "fun!";
```

NOT in
quotation
marks

- Use the `\n` escape sequence to start a new line

```
cout << "Programming is\n";
```

```
cout << "fun!";
```

in quotation
marks

Formatting Output

- Requires **iomanip** file, i.e. `#include <iomanip>`
- Can control how output displays for numeric, string data:
 - size
 - position
 - number of digits

Example:

	720
	720.0
	720.000000000
	7.2e+2

Stream Manipulators (1)

- Some affect **just the next value** displayed:
 - `setw(x)`: print in a field at least `x` spaces wide.
Use more spaces if field is not wide enough

```
int num1 = 2897, num2 = 5;  
cout << num1 << " " << num2;
```

Output:
2897 5

```
cout << setw(6) << num1 << setw(6) << num2;
```

Output:
2897 5

Stream Manipulators (2)

- Some affect values **until changed again**:
 - `fixed`: use decimal notation for floating-point values
 - `setprecision(x)`:
 - when used with `fixed`, print floating-point value using `x` digits after the decimal.
 - Without `fixed`, print floating-point value using `x` significant digits (i.e. total digits before and after the decimal point)
 - `showpoint`: display decimal point and trailing zeros even if there is no fractional part

Note: if a value is in fewer digits of precision than specified by `setprecision`, the manipulator will have no effect.

Example

Program 3-17

```
1  // This program asks for sales amounts for 3 days. The total
2  // sales are calculated and displayed in a table.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9      double day1, day2, day3, total;
10
11     // Get the sales for each day.
12     cout << "Enter the sales for day 1: ";
13     cin >> day1;
14     cout << "Enter the sales for day 2: ";
15     cin >> day2;
16     cout << "Enter the sales for day 3: ";
17     cin >> day3;
18
19     // Calculate the total sales.
20     total = day1 + day2 + day3;
21
```

Continued...

Example (cont'd)

```
22 // Display the sales amounts.
23 cout << "\nSales Amounts\n";
24 cout << "-----\n";
25 cout << setprecision(2) << fixed;
26 cout << "Day 1: " << setw(8) << day1 << endl;
27 cout << "Day 2: " << setw(8) << day2 << endl;
28 cout << "Day 3: " << setw(8) << day3 << endl;
29 cout << "Total: " << setw(8) << total << endl;
30 return 0;
31 }
```

Use setprecision with fixed,
2 digits after the decimal point

Display at least 8 spaces wide

Program Output with Example Input Shown in Bold

Enter the sales for day 1: **1321.87**

Enter the sales for day 2: **1869.26**

Enter the sales for day 3: **1403.77**

Sales Amounts

Day 1: 1321.87

Day 2: 1869.26

Day 3: 1403.77

Total: 4594.90

4.2 The `cin` Object

- Standard input object
- Need `iostream` header file
 - `#include <iostream>`
- Used to read input from **keyboard**
- Information retrieved from `cin` with `>>`
- Input is stored in one or more variables

The `cin` Object

- You should always use `cout` to display a prompt before each `cin` statement
- `cin` converts data to the type that matches the variable:

```
int height;  
cout << "How tall is the room? ";  
cin >> height;
```

How tall is the room? 6.45

```
cout << height;
```

6

The `cin` Object

- Can be used to input more than one value:

```
cin >> height >> width;
```

- Multiple values from keyboard must be separated by spaces
- Order is important: first value entered goes to first variable, etc.

Input a string

- Using **cin** with the **>>** operator to input strings can cause problems:
 - It passes over and ignores any leading *whitespace characters* (*spaces, tabs, or line breaks*)
 - It stops reading when it gets to the next whitespace
- You can use a C++ function named **getline**

```
string name;
```

```
cout << "Enter you name: "  
cin >> name;
```

Input:

Kate Smith

name = Kate

```
string name;
```

```
cout << "Enter you name: "  
getline(cin, name);
```

Input:

Kate Smith

name = Kate Smith

5. Arithmetic operator

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators:
 - unary (1 operand) -5
 - binary (2 operands) $13 - 7$
 - ternary (3 operands) $\text{exp1} ? \text{exp2} : \text{exp3}$

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	<code>ans = 7 + 3;</code>	10
-	subtraction	<code>ans = 7 - 3;</code>	4
*	multiplication	<code>ans = 7 * 3;</code>	21
/	division	<code>ans = 7 / 3;</code>	2
%	modulus	<code>ans = 7 % 3;</code>	1

Note:

- Division operator returns the quotient.
- Modulus operator returns the remainder of an integer division.

A Closer Look at the / Operator

- / (division) operator performs **integer division** if both operands are integers

- The result is always an integer

- The remainder will be discarded

```
cout << 13 / 5;    // displays 2
```

```
cout << 91 / 7;    // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;  // displays 2.6
```

```
cout << 91.0 / 7;  // displays 13.0
```

A Closer Look at the % Operator

- % (modulus) operator computes the remainder resulting from integer division

```
cout << 13 % 5;    // displays 3
```

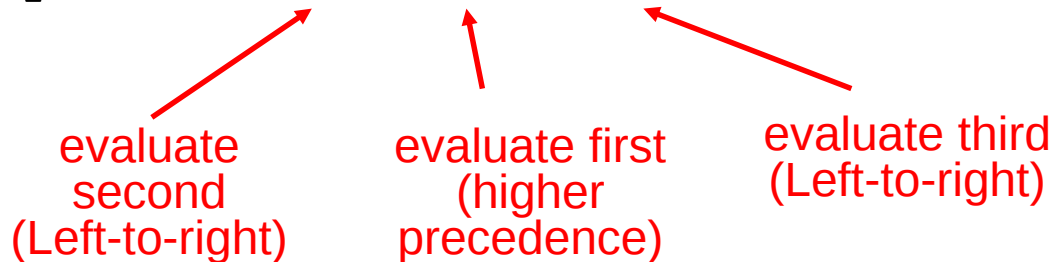
- % requires integers for both operands

```
cout << 13 % 5.0;  // error
```

Operator precedence

- In an expression with more than one operator, the operator with the highest **precedence** works first.
 - https://en.cppreference.com/w/cpp/language/operator_precedence
- If two operators have the same precedence, they work according to their **associativity**.

Example: $2 + 2 * 2 - 2$



- Use parentheses () to override the order of operations

Example: $(2 + 2) * 2 - 2$

Practice

$$5 + 2 * 4 = \mathbf{13}$$

$$10 / 2 - 3 = \mathbf{2}$$

$$8 + 12 * 2 - 4 = \mathbf{28}$$

$$4 + 17 \% 2 - 1 = \mathbf{4}$$

Type Conversion (automatically)

- When using the = operator, the type of expression on right will be converted to type of variable on left.
- When operating on values of different data types, the lower one is promoted to the type of the higher one.

Highest ↑ long double
double
float
unsigned long
long
unsigned int
Lowest ↓ int

Type Casting (manually)

- Manual data type conversion, the format is:

```
static_cast<DataType>(Value)
```

- Useful for floating point division using `ints`:

```
double m;
```

```
m = static_cast<double>(y2-y1)  
                               / (x2-x1) ;
```

- Useful to see `int` value of a `char` variable:

```
char ch = 'C';
```

```
cout << ch << " is "
```

```
    << static_cast<int>(ch) ;
```

Combined Assignment

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

`sum = sum + 1;`

is equivalent to

`sum += 1;`

Table 3-9

Operator	Example Usage	Equivalent to
<code>+=</code>	<code>x += 5;</code>	<code>x = x + 5;</code>
<code>-=</code>	<code>y -= 2;</code>	<code>y = y - 2;</code>
<code>*=</code>	<code>z *= 10;</code>	<code>z = z * 10;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>c %= 3;</code>	<code>c = c % 3;</code>

6. ++ and -- operators

- ++ is the increment operator. It adds one to a variable.

`val++;` is the same as `val = val + 1;`

- -- is the decrement operator. It subtracts one from a variable.

`val--;` is the same as `val = val - 1;`

- ++ and -- can be used before (prefix) or after (postfix) a variable:

`++val;` `val++;`

`--val;` `val--;`

Prefix vs. Postfix

- `++val`, `--val`, `val++`, `val--` are expressions, therefore they have value
- In prefix mode (`++val`, `--val`) the operator increments or decrements, *then* returns the value of the variable, i.e., the value of the expression is the "update" value of the variable
- In postfix mode (`val++`, `val--`) the operator returns the value of the variable, *then* increments or decrements, i.e., the value of the expression is the value of the variable before being updated

Examples

```
int num, val = 12;  
cout << val++;      // displays 12,  
                     // val is now 13;  
cout << ++val;      // sets val to 14,  
                     // then displays it  
num = --val;        // sets val to 13,  
                     // stores 13 in num  
num = val--;        // stores 13 in num,  
                     // sets val to 12
```

7. C++ library

- A library is a package of code that can be reused
- Typically, a C++ library comes in two pieces:
 - 1) A header file that defines the functionality the library is exposing (offering) to the programs
 - 2) A precompiled binary that contains the implementation of that functionality pre-compiled into machine language.
- C++ Standard Library headers (open the link)
 - <https://en.cppreference.com/w/cpp/header>
 - E.g.: `<iostream>`, `<string>`, `<iomanip>`, `<cmath>`

<cmath> header file

- Require `cmath` header file
- Take `double` as input, return a `double`
- Commonly used functions: `pow(2,8)`

<code>sin</code>	Sine
<code>cos</code>	Cosine
<code>tan</code>	Tangent
<code>sqrt</code>	Square root
<code>log</code>	Natural (e) log
<code>abs</code>	Absolute value (also take and return an int)

For more functions in <cmath> library, refer to:

- <https://en.cppreference.com/w/cpp/header/cmath>

<cstdlib> header file

- Include general functions for program control, dynamic memory allocation, random numbers, sort and search
- Functions for random number generation
 - `rand()`:
 - Return a pseudo-random integer between 0 and the largest int the compute holds.
 - Yield same sequence of numbers each time program is run.
 - `srand(x)`:
 - Initialize random number generator with `unsigned int x`

rand() function

- How to limit the range of the generated random number?

```
y = (rand() % (maxValue - minValue + 1)) + minValue;
```

- **minValue is the lowest number in the range**
- **maxValue is the highest number in the range**

- Example:

```
const int MIN_VALUE = 1;  
const int MAX_VALUE = 100;  
y = (rand() % (MAX_VALUE - MIN_VALUE + 1)) + MIN_VALUE ;
```

This code assigns a random number in range of 1 – 100 to variable y.

Thanks

Reading textbook

- Chapter 1, 2, 3

Where to find resources/help?

- Required textbook:
 - Starting Out with C++: From Control Structures through Objects, by Tony Gaddis, Pearson (9th Edition)
- Instructor's web page for the course:
 - Dr. Sophie Wang: <http://faculty.Salisbury.edu/~xswang/cosc120.htm>
- SU myClass Canvas:
 - <https://www.salisbury.edu/administration/academic-affairs/instructional-design-delivery/cms/>
 - Course policy, lecture ppts/videos, labs, assignments, quizzes, projects, etc.
- Department tutoring program (free!):
 - [Mathematical Sciences & Computer Science Tutoring Center | Salisbury University](#)
- Lab assistant – Ethan Grey (egray3@gulls.salisbury.edu):
 - Office hours:

How to be successful?

- Show up
- Read textbook/lecture notes before and after each class
 - Many small pieces of information that may not be covered in class
- Start early to work on your projects
- Experiment & **practice**
 - Try and find out what will happen. Why?
- Ask for help