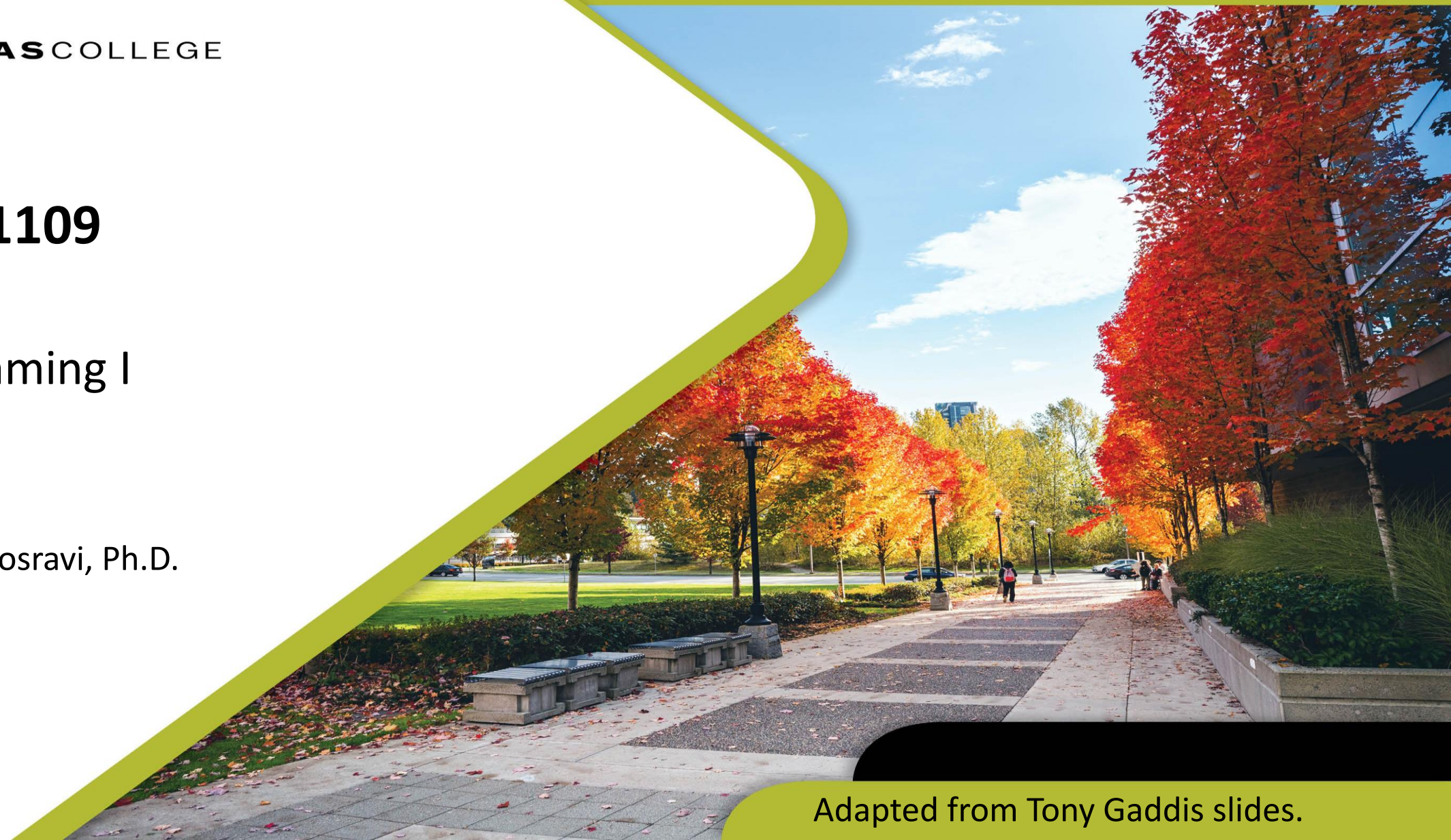
DOUGLAS COLLEGE

CMPT 1109

Programming I

Shahriar Khosravi, Ph.D.

Lecture 1



Adapted from Tony Gaddis slides.

Plan for Today

- Introduction
- The Welcome Survey (so far)
- Our Course Policies
- A Brief Introduction to Computer Systems
- Programs and Programming Languages
- Common Elements of A Program (with A Focus on C++)
- Lines and Statements
- Variables and Variable Definitions
- Typical Program Structure
- The Programming Process
- Parts of a C++ Program
- The **`cout`** Object
- The **`#include`** Directive

Shahriar Khosravi (He/Him)

- **Bachelor of Science** in Aerospace Science and Engineering (**University of Michigan**, 2012)
- **Doctor of Philosophy** in Aerospace Science and Engineering (**University of Toronto**, 2016)
  - Thesis topic: Aircraft wing design using numerical optimization based on computational fluid dynamics and finite-element structural analysis
- **Credit Risk Analytics Manager** at BMO Bank of Montreal (Toronto)
  - Data science, software engineering, and numerical modelling using the SAS and SQL scripting languages
- **Structural Dynamics and Vibrations Engineer** at Pratt and Whitney Canada (Toronto)
  - Aircraft engine design from a vibrations perspective and data science for engine component performance prediction
- **Professor of Electronics and Mechanical Engineering Technology** at Seneca College (Toronto)
  - Taught courses on the C, C++, C# programming languages, and the .NET Framework

What You Have Said in the Welcome Survey (Adjectives)

What You Have Said in the Welcome Survey (Concerns)

Getting lost in the course.

Difficulty of the content and work.

No prior programming experience.

Assignments length.

No hands-on practice.

I am concerned about coming off as annoying when reaching out for help.

No fun assignments!

What You Need to Succeed in This Course

1. Attention to detail.
2. Willingness to learn the logical mindset.
3. A working computer with the **Windows 10 Operating System** that can connect to the Internet.
4. The Integrated Development Environment **Microsoft Visual Studio 2019** (tutorial for installing it is on the Blackboard for your reference).

DOUGLAS COLLEGE

Our Course Policies

DOUGLAS COLLEGE

Introduction to Computer Systems

What Is A Program and Why We Need It?

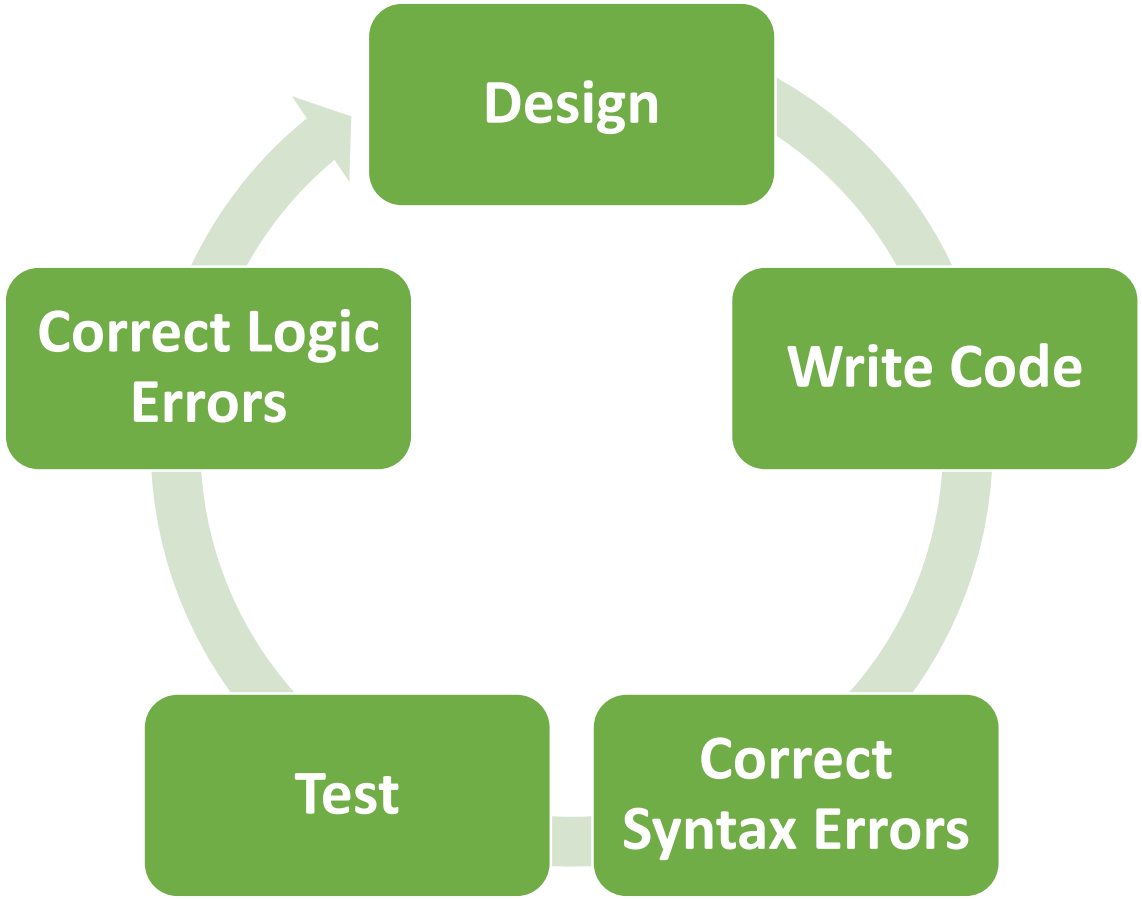
- Humans use computers in a wide variety of activities.
- Computers can do such a wide variety of things **because they can be programmed**.
- This means that computers are not designed to do just one job, but any job that their programs instruct them to do.
- A **program** is **a set of instructions** that a computer follows to perform a task.
- Programs are commonly referred to as **software** in Computer Science.
- All of the software that we use to make our computers useful is created by **programmers** (or software developers).

What Is Programming?

- Computer programming requires a **detail-oriented mindset** because every single aspect of a program must be carefully designed:
  - The logical flow of the instructions
  - The mathematical procedures
  - The appearance of the screens
  - The way information is presented to the user
  - The program's "user-friendliness"
  - Manuals and other forms of written documentation

The Programming Life-Cycle

- Programs **rarely work right the first time they are written**, a lot of testing, correction, and redesigning is required.
- Computer programming is **an iterative process**, so one must not expect or assume that a program works unless it is thoroughly tested.
- This demands patience and persistence from the programmer.

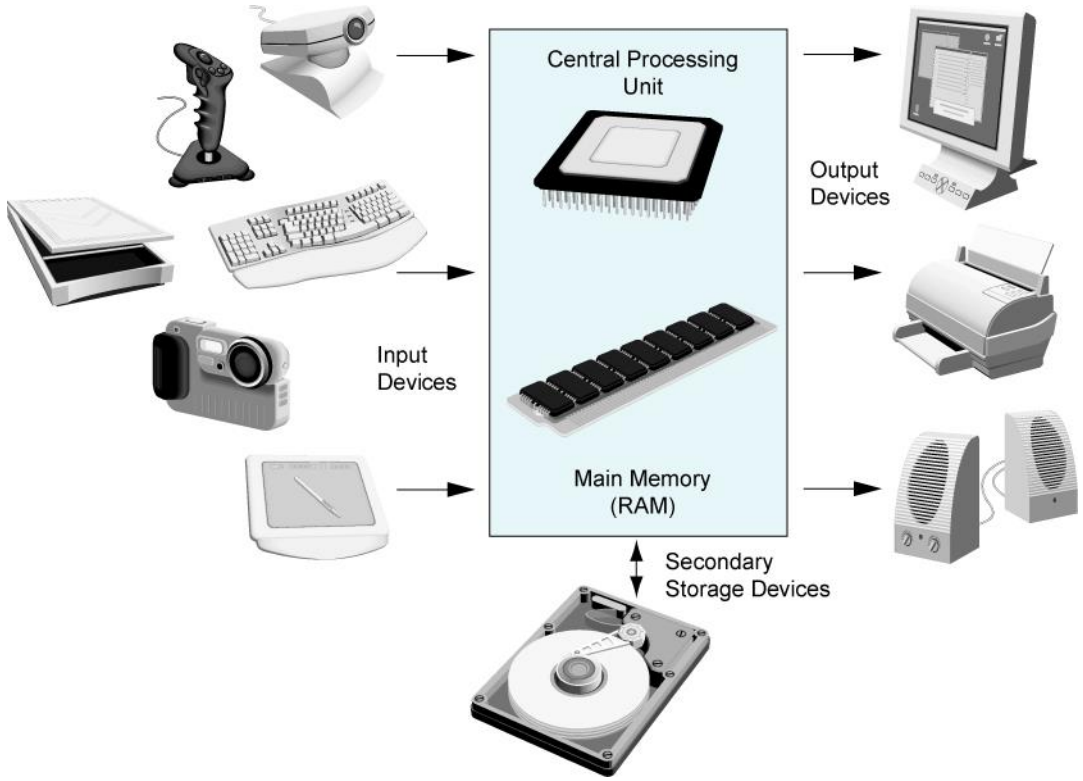


Writing Programs

- Writing software demands discipline as well because programmers must learn special languages like C++ since computers do not understand English or other human languages.
- Languages such as C++ have strict rules that must be carefully.
- A “minor” mistake, such as misplacing a semicolon or missing a curly bracket, often has catastrophic consequences for the programs.
- Therefore, attention to detail is an extremely important aspect of writing computer programs.

Hardware

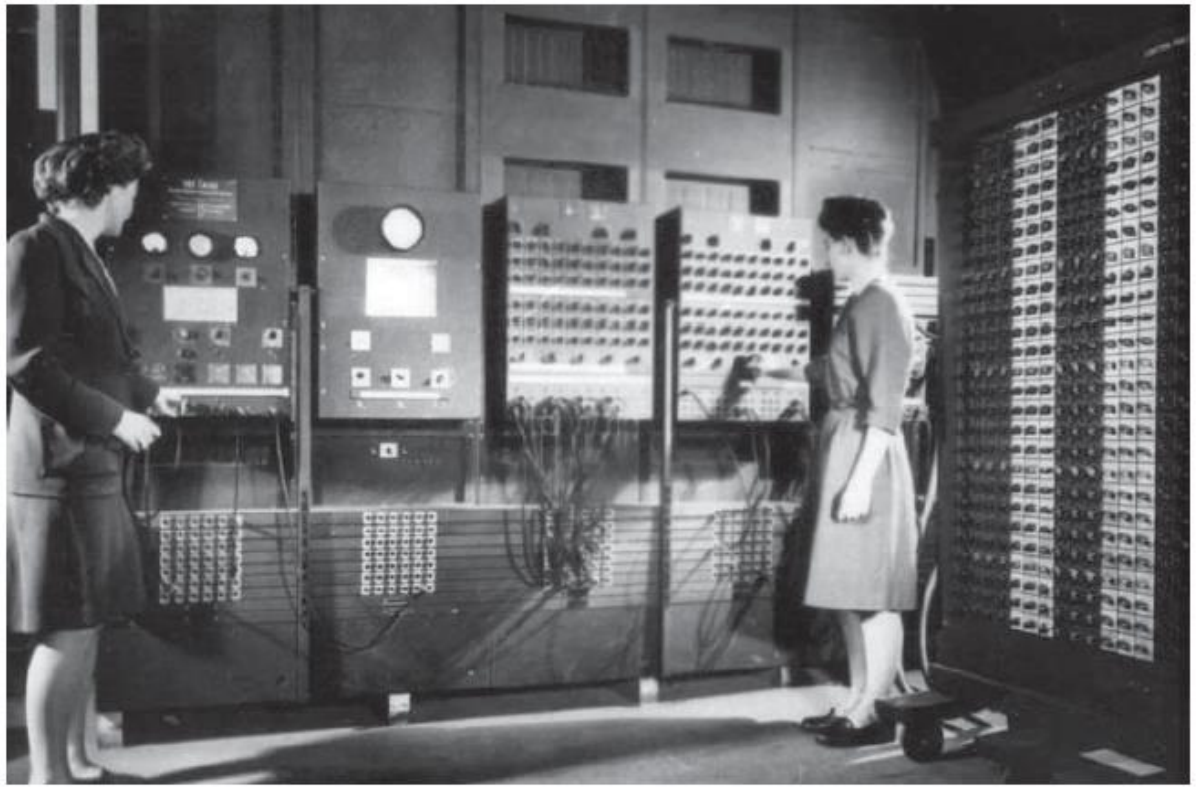
- **Hardware** refers to the physical components that a computer is made of.
- A computer is not an individual device, but a system of devices.
- A typical computer system consists of the following major components:
  - The central processing unit (CPU)
  - Main memory
  - Secondary storage devices
  - Input devices
  - Output devices



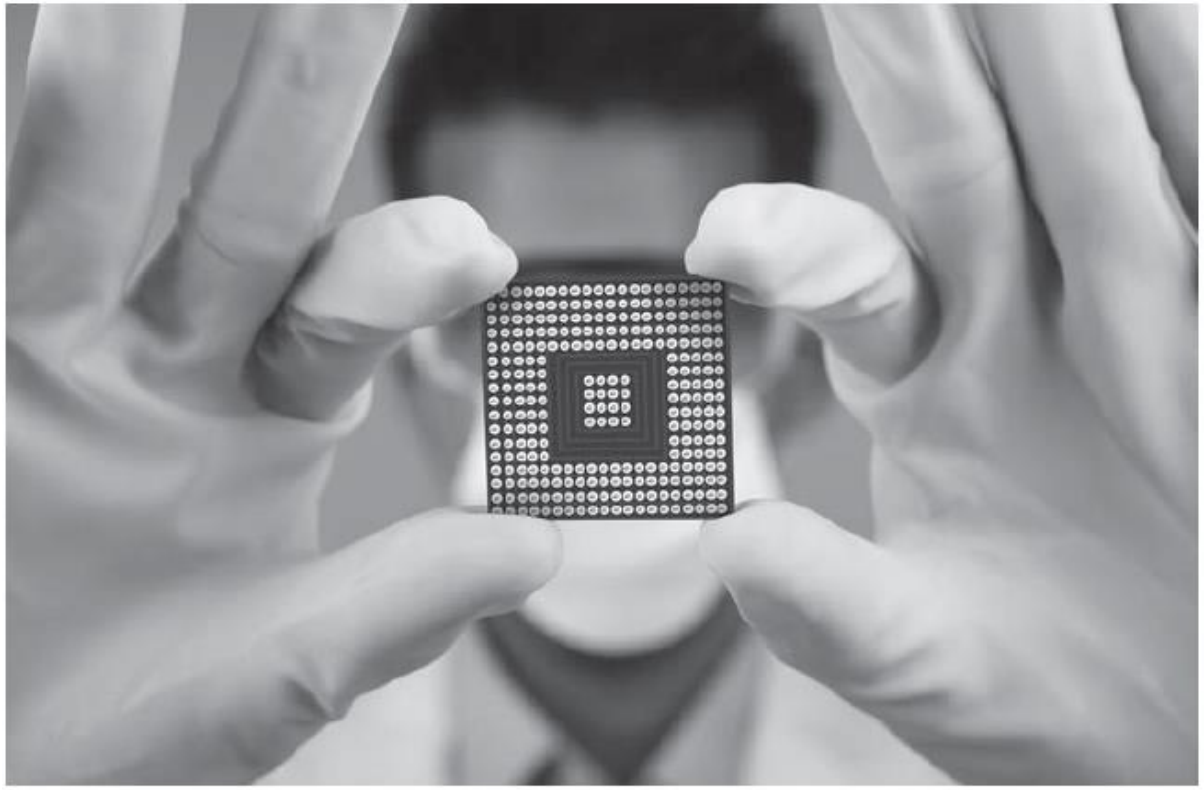
The CPU

- When a computer is performing the tasks that a program instructs it to do, the computer is said to be **running** or **executing** the program.
- The **central processing unit**, or **CPU**, is the part of a computer that actually executes our programs.
- The CPU is the most important component in a computer because without it, the computer could not run software.
- In the earliest computers, CPUs were huge devices made of electrical and mechanical components such as vacuum tubes and switches.

The CPU



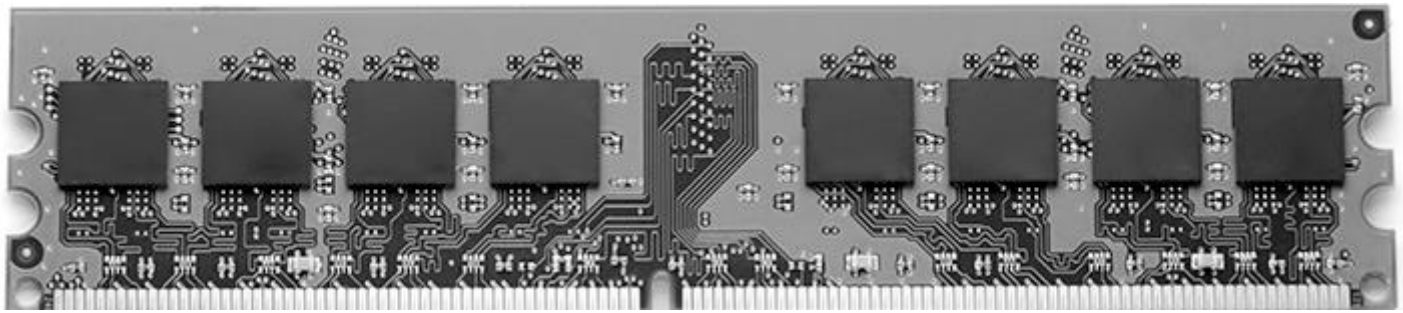
The historic ENIAC computer in 1945 was 8 feet tall and weighed 30 tons.



Today's CPUs are small chips known as microprocessors.

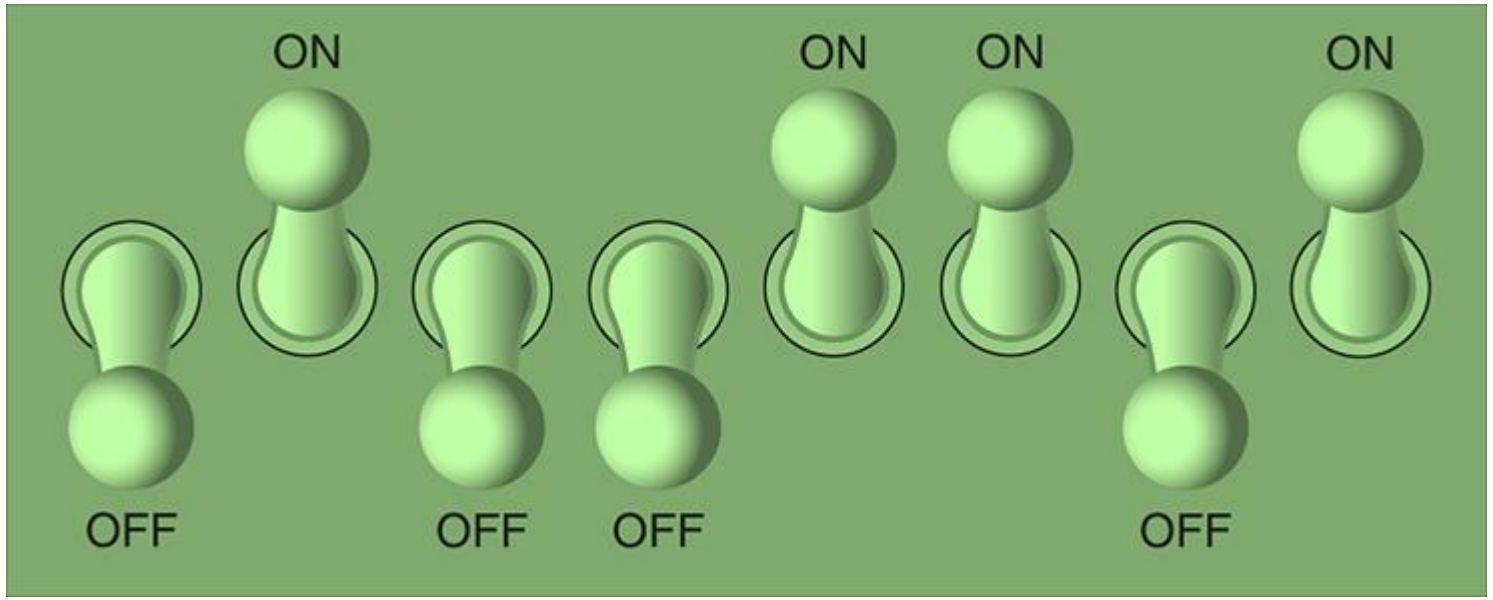
Main Memory

- **Main memory**: where computer stores a program while program is running, and data used by the program.
  - Known as **Random Access Memory** or **RAM**.
  - CPU is able to quickly access data in RAM.
  - **Volatile memory** used for temporary storage while program is running.
  - **Contents are erased when computer is off**.



Main Memory

- A computer's main memory is divided into tiny storage locations known as **bytes**.
- **Byte**: just enough memory to store letter or small number.
  - Divided into eight **bits**.
  - **Bit**: electrical component that can hold positive or negative charge, like on/off switch.
  - The on/off pattern of bits in a byte represents data stored in the byte.



Think of a byte as eight switches each of which can either be in an "ON" or "OFF" state.

Main Memory – Example

- In this example, the number 149 is stored in the byte with the address 16, and the number 72 is stored at address 23.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16 **149**	17	18	19
20	21	22	23 **72**	24	25	26	27	28	29

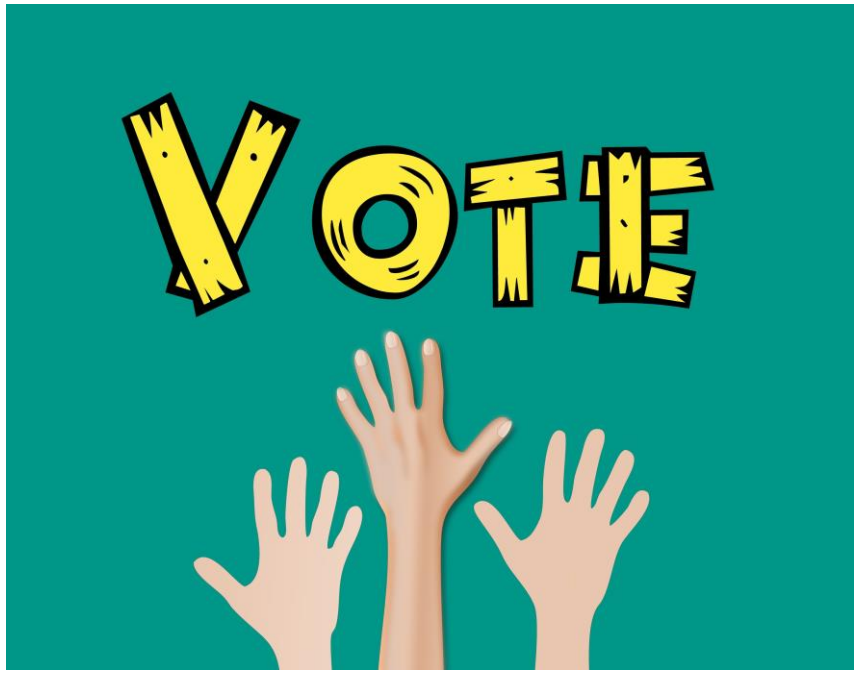
Secondary Storage

- **Secondary storage** is a type of memory that can hold data for long periods of time – even when there is no power to the computer.
- Frequently used programs are stored in secondary memory and loaded into main memory as needed.
- Secondary storage comes in a variety of media:
  - **Magnetic disk drives**: traditional hard drives that use a moveable mechanical arm to read/write
  - **Solid-state hard drives**: data stored in chips, no moving parts
  - **Optical disks**: CD-ROM (compact disk read-only memory), DVD (digital versatile disc)
  - **Flash drives**: connected to the USB port

Poll 1 (Extra Credit)

Accessing data from secondary storage is faster than accessing data from RAM.

a) True
b) False
c) Not sure!



Please use the "Poll" window to participate for extra credit! One answer only please!

Input and Output (I/O) Devices

- **Input devices** are devices that send information to the computer from “outside”.
- Many devices can provide input:
  - Keyboard, mouse, touchscreen, scanner, digital camera, microphone
  - Disk drives, CD drives, and DVD drives
- Any output information that the computer must send to the outside world is sent to an **output device**, which formats and presents it.
- Common output devices are monitors, printers, and speakers.
- Disk drives, USB drives, and CD/DVD recorders can also be considered output devices because the CPU sends them information to be saved

System Software

- The programs that control and manage the basic operations of a computer are generally referred to as **system software**.
- System software typically includes the following types of programs:
  - **Operating System:**
    - controls the internal operations of the computer's hardware, manages all the devices connected to the computer, allows data to be saved to and retrieved from storage devices, and allows other programs to run on the computer.
  - **Utility Programs:**
    - performs a specialized task that enhances the computer's operation or safeguards data. Examples of utility programs are virus scanners, file-compression programs, and data-backup programs.
  - **Software Development Tools:**
    - tools that programmers use to create, modify, and test software. Examples include compilers and integrated development environments (IDEs).

Application Software

- Programs that make a computer useful for everyday tasks are known as **application software**.
- Application software are essentially programs that provide services to the user.
- Examples:
  - Word processing, games (e.g., Need for Speed Most Wanted and Hitman Codename 42), and programs to solve specific problems.

DOUGLAS COLLEGE

Programs and Programming Languages

Programs and Programming Languages

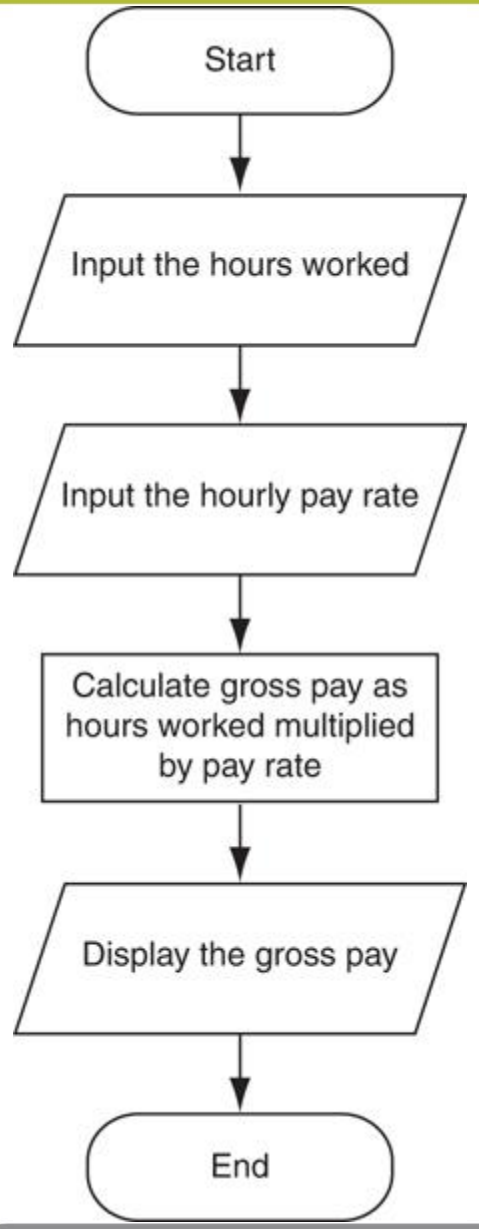
- A **program** is a set of instructions that the computer follows to perform a task.
- To write a program, we must start with an **algorithm**, which is a set of well-defined steps.
- For example, suppose we want the computer to calculate someone's gross pay. Here is a list of things the computer should do:
  1. Get the number of hours worked.
  2. Get the hourly pay rate.
  3. Multiply the number of hours worked by the hourly pay rate.
  4. Display the result of the calculation that was performed in step 3.
- These steps have to be translated into C++ code.
- Programmers commonly use two tools to help them accomplish this:
  - Pseudocode.
  - Flowcharts.

Pseudocode

- **Pseudocode**: fake code!
  - Informal language that has no syntax rule.
    - **Not meant to be compiled or executed!**
    - Used to create model program (i.e., prototype).
    - No need to worry about syntax errors, can focus on program's design.
    - Can be translated directly into actual code in any programming language.
  - Example:
    - Input the hours worked.
    - Input the hourly pay rate.
    - Calculate gross pay as hours worked multiplied by pay rate.
    - Display the gross pay.
- Each statement in the pseudocode represents an operation that can be performed in Python.

Flowcharts

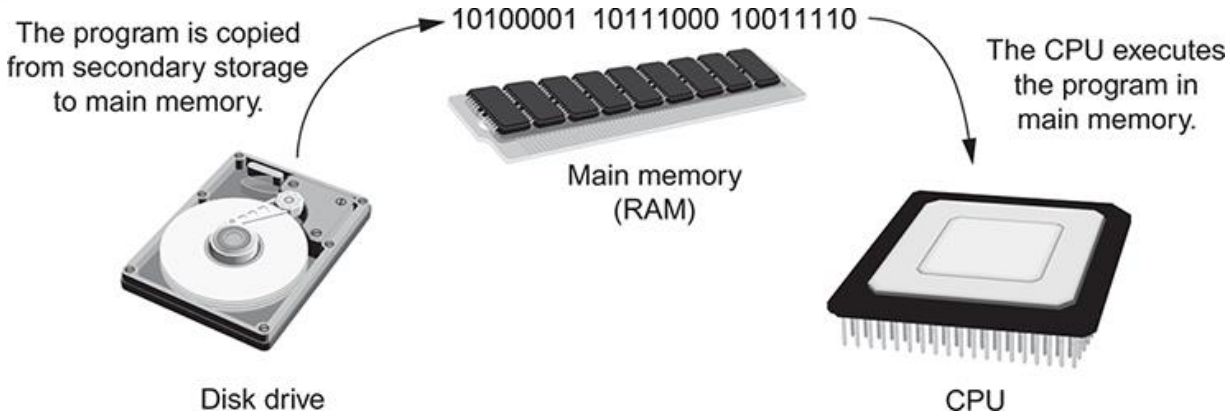
- **Flowchart**: diagram that graphically depicts the steps in a program
  - Ovals are terminal symbols.
  - Parallelograms are input and output symbols.
  - Rectangles are processing symbols.
  - Symbols are connected by arrows that represent the flow of the program.

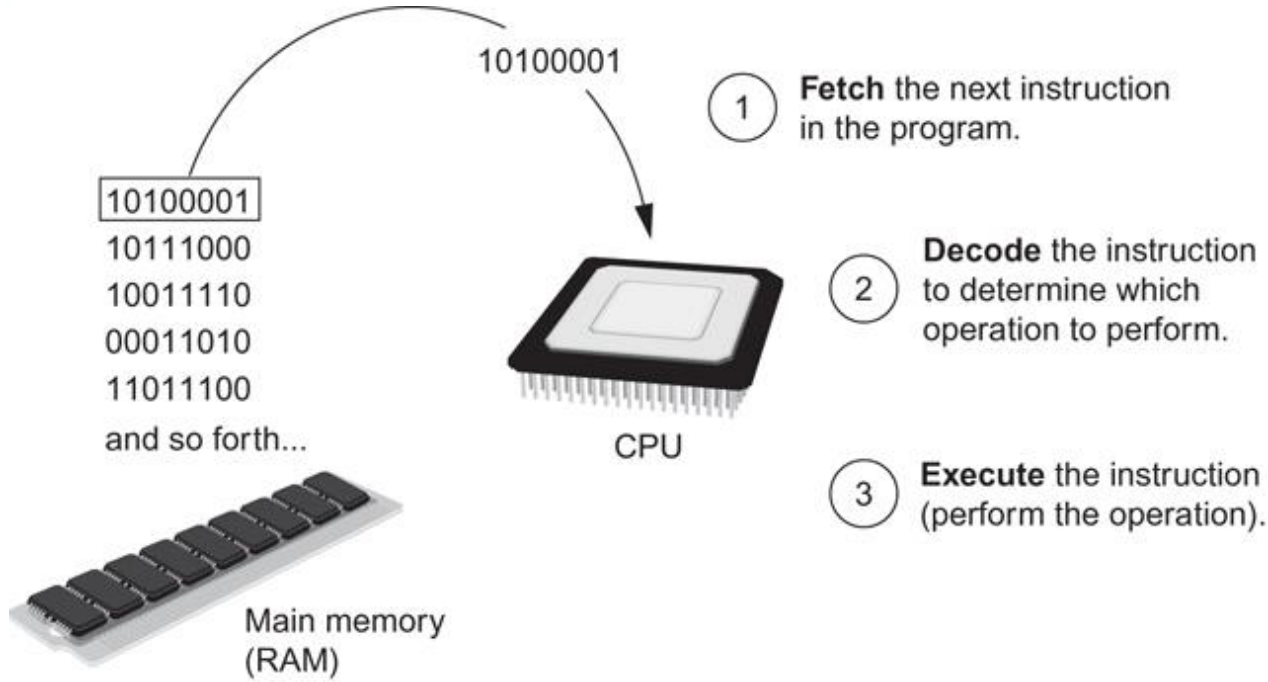


Programs and Programming Languages

- A computer's CPU can only process instructions that are written in **machine language**.
- If we were to look at a machine language program, we would see a stream of **binary numbers** (numbers consisting of only 1's and 0's).
- The binary numbers form machine language instructions, which the CPU interprets as commands (example: 1011010000000101).
- The process of encoding an algorithm in machine language is very tedious and difficult. In addition, each different type of CPU has its own machine language.
- **Programming languages**, which use words instead of numbers, were invented to ease the task of programming.
- A program can be written in a programming language, such as C++, which is much easier to understand than machine language.

Programs and Programming Languages



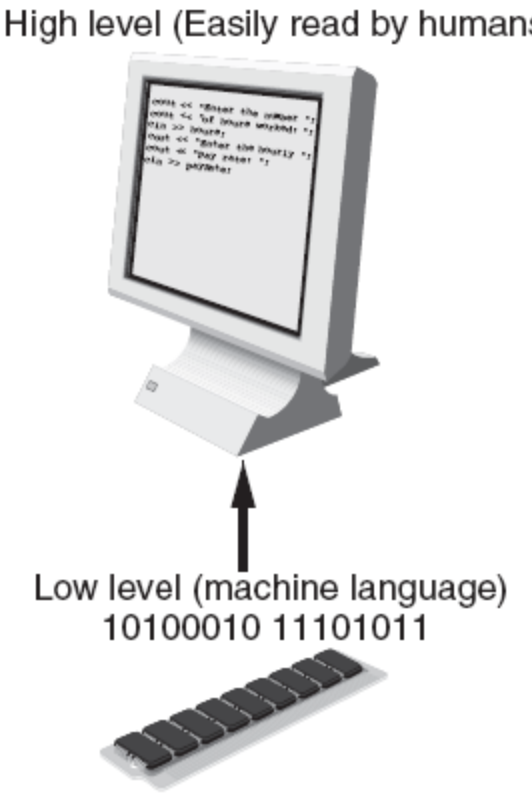


Example C++ Program

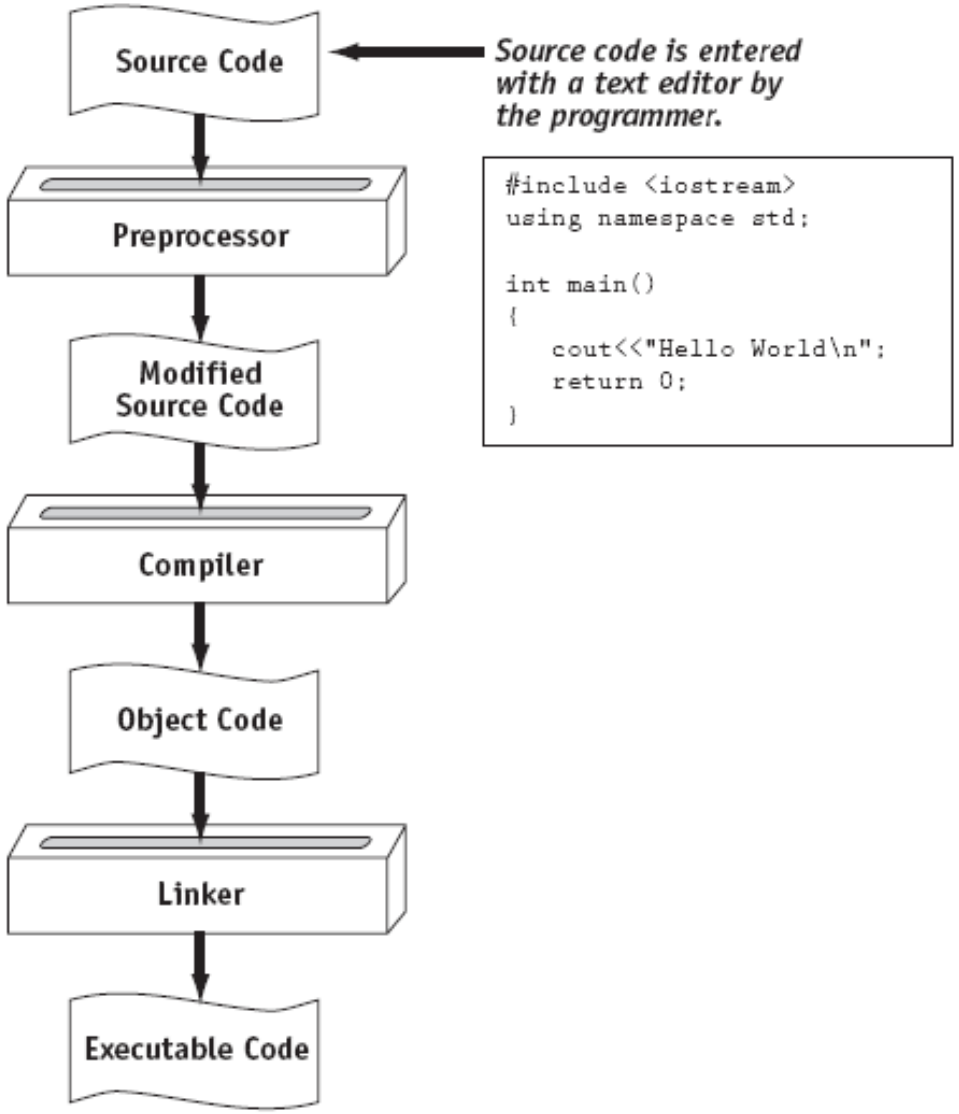
```
// This progam calculates the user's pay.
#include <iostream>
using namespace std;

int main()
{
    double hours, rate, pay;

    // Get the number of hours worked.
    cout << "How many hours did you work? ";
    cin >> hours;

    // Get the hourly pay rate.
    cout << "How much do you get paid per hour? ";
    cin >> rate;

    // Calculate the pay.
    pay = hours * rate;

    // Display the pay.
    cout << "You have earned $" << pay << endl;
    return 0;
}
```

Programming Languages and C++

- There are two types of programming languages:
  - **Low-level**:
    - Used for communication with computer hardware directly. Often written in binary machine code (0's/1's) directly.
  - **High-level**:
    - Closer to human language.
- In addition to the high-level features necessary for writing user applications, C++ also has many low-level features.
- **C++ is based on the C language**, which was invented for purposes such as writing operating systems and compilers. Since C++ evolved from C, it carries all of C's low-level capabilities with it.
- C++ is popular not only because of its mixture of low- and high-level features, but also because of its **portability** (i.e., a C++ program can be written on one type of computer and then run on many other types of systems).



Source Code, Object Code, and Executable Code




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

int main()
{
   cout<<"Hello World\n";
   return 0;
}
```

Source Code, Object Code, and Executable Code

- When a C++ program is written, it must be typed into the computer and saved to a file. A **text editor** (e.g., Notepad++) is used for this task.
- The statements written by the programmer are called **source code**, and the file they are saved in is called the source file.
- After the source code is saved to a file, the process of translating it to machine language can begin.
- During the first phase of this process, a program called the **preprocessor** reads the source code.
- The preprocessor searches for special lines that begin with the **#** symbol. These lines contain commands that cause the preprocessor to modify the source code in some way.
- During the next phase the compiler steps through the preprocessed source code, translating each source code instruction into the appropriate **machine language instruction**.
- This process will uncover any **syntax errors** that may be in the program. **Syntax errors** are illegal uses of key words, operators, punctuation, and other language elements.
- If the program is free of syntax errors, the compiler stores the translated machine language instructions, which are called **object code**, in an **object file**.

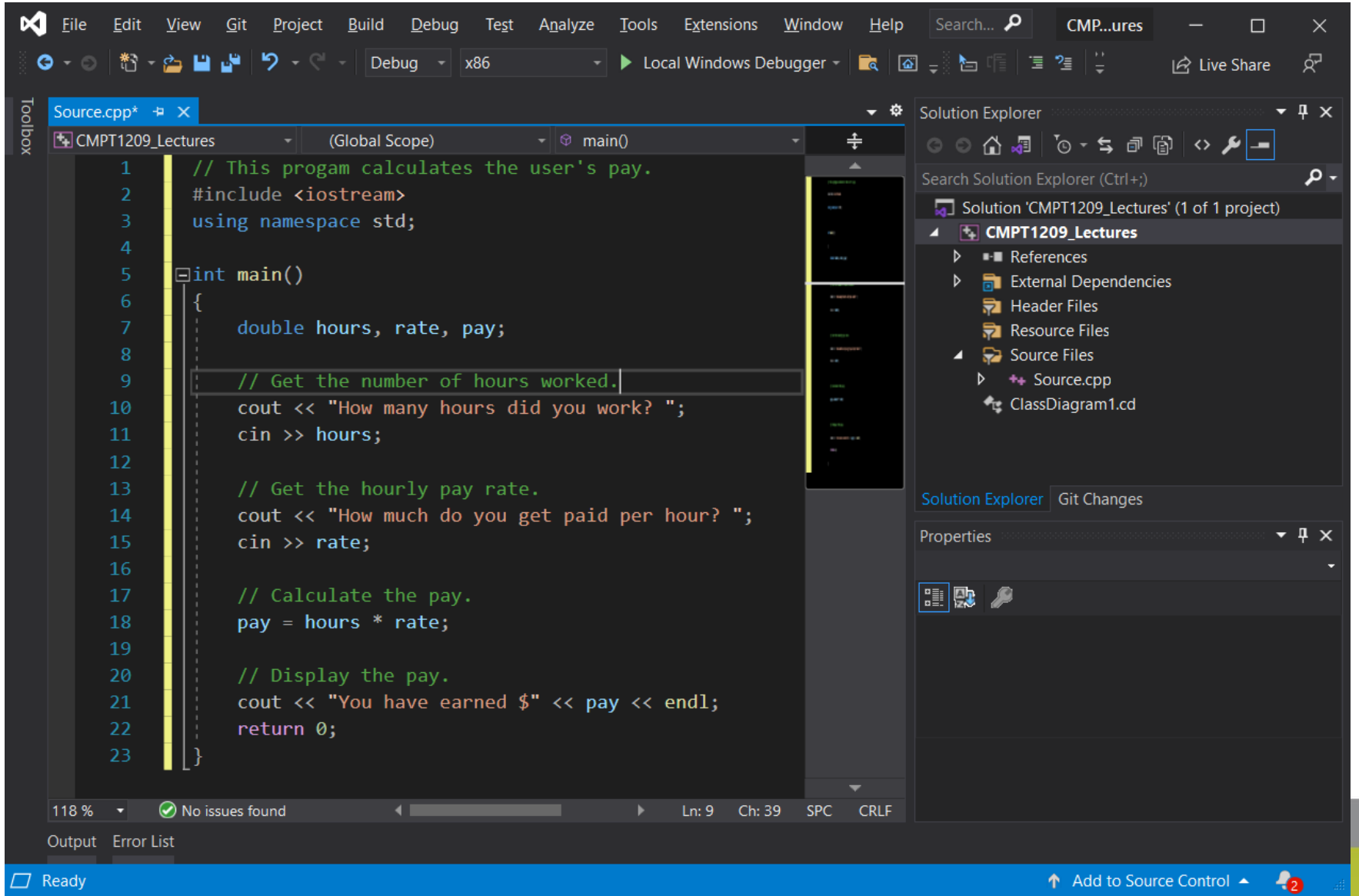
Source Code, Object Code, and Executable Code

- Although an **object file** contains machine language instructions, it is **not** a complete program.
- C++ is conveniently equipped with a library of prewritten code for performing common operations or sometimes-difficult tasks.
- For example, the library contains hardware-specific code for displaying messages on the screen and reading input from the keyboard. This library, called the **run-time library**, is extensive.
- When the compiler generates an **object file**, it does not include machine code for any run-time library routines the programmer might have used.
- During the last phase of the translation process, another program called the **linker** combines the object file with the necessary library routines.
- Once the linker has finished with this step, an **executable file** is created.
- The **executable file** contains machine language instructions, or executable code, and is finally ready to run on the computer

Integrated Development Environments (IDEs)

- Many development systems, particularly those on personal computers, have **integrated development environments (IDEs)**.
- These environments consist of a text editor, compiler, debugger, and other utilities integrated into a package with a single set of menus!
- Preprocessing, compiling, linking, and even executing a program is done with a single click of a button, or by selecting a single item from a menu.
- For the purpose of this course, we will use **Microsoft Visual Studio (2019 or later)** as the integrated development environment.

Integrated Development Environments (IDEs)



DOUGLAS COLLEGE

Common Elements of A Program

What is A Program Made of?

- Common elements in programming languages:
  - **Keywords**
    - Also known as **reserved words**.
    - Have a special meaning in C++.
    - Can **not** be used for any other purpose.
    - Keywords in our example program: **`using`**, **`namespace`**, **`int`**, **`double`**, and **`return`**.

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

What is A Program Made of?

- Common elements in programming languages:
  - **Keywords**
    - Also known as **reserved words**.
    - Have a special meaning in C++.
    - Can **not** be used for any other purpose.
    - Keywords in our example program: **`using`**, **`namespace`**, **`int`**, **`double`**, and **`return`**.

NOTE: The **`#include <iostream>`** statement in line 2 is a preprocessor directive.

NOTE: In C++, keywords are written in all lowercase.

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

What is A Program Made of?

- Common elements in programming languages:
  - **Programmer-Defined Identifiers**
    - Names made up by the programmer.
    - Not part of the C++ language.
    - Used to represent various things: variables (memory locations), functions, etc.
    - In our example program: **hours**, **rate**, and **pay**.

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

What is A Program Made of?

- Common elements in programming languages:
  - **Operators**
    - Used to perform operations on data.
    - Many types of operators, e.g., arithmetic: **+,-,*,/**.

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

What is A Program Made of?

- Common elements in programming languages:
  - **Punctuation**
    - Characters that mark the end of a statement, or that separate items in a list (e.g., **;** or **,**).

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

What is A Program Made of?

- Common elements in programming languages:
  - **Syntax**
    - The rules of grammar that must be followed when writing a program.
    - The rules control the use of key words, operators, programmer-defined symbols, and punctuation.
    - Breaking syntax rules lead to **compilation errors** (i.e., errors that arise during the compilation of the source code by the compiler).
    - Compilation errors are often difficult to understand for novice programmers because, for example, missing just a semicolon or misplacing it could lead to many different compilation errors from the compiler that are counterintuitive.
    - This is why it is extremely important to pay close attention to syntax rules in any programming language.

What is A Program Made of?

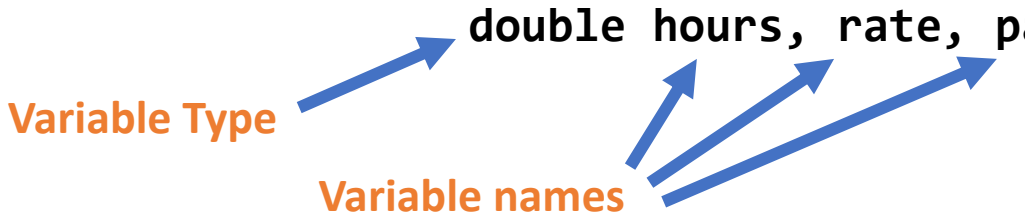
- Common elements in programming languages:
  - **Variables**
    - A variable is a named storage location in the computer's memory for holding a piece of data.
    - In our example, we used three variables:
    - The **`hours`** variable was used to hold the hours worked.
    - The **`rate`** variable was used to hold the pay rate.
    - The **`pay`** variable was used to hold the gross pay.
  - **Variable definitions**
    - To create a variable in a program, we must write a **variable definition** (also called a **variable declaration**).
    - Here is the statement from our example that defines the variables:

```
double hours, rate, pay;
```

What is A Program Made of?

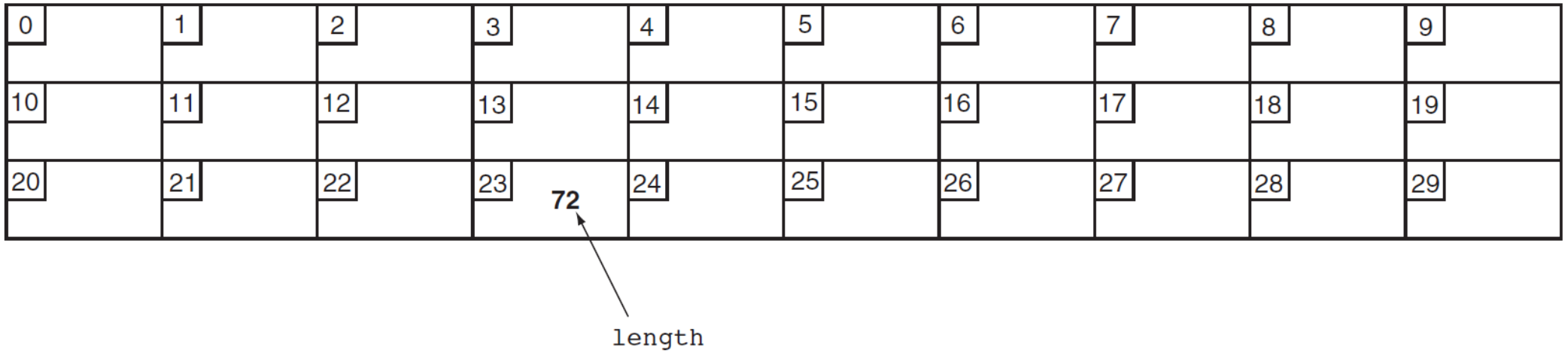
- Common elements in programming languages:
  - **Variables**
    - A variable is a named storage location in the computer's memory for holding a piece of data.
    - In our example, we used three variables:
    - The **`hours`** variable was used to hold the hours worked.
    - The **`rate`** variable was used to hold the pay rate.
    - The **`pay`** variable was used to hold the gross pay.
  - **Variable definitions**
    - To create a variable in a program, we must write a **variable definition** (also called a **variable declaration**).
    - Here is the statement from our example that defines the variables:

```
double hours, rate, pay;
```



Variables

- Variables are symbolic names that represent locations in the computer's random-access memory (RAM).
- When information is stored in a variable, it is stored in RAM.
- Imagine a program has a variable named **`length`**.
- The diagram below illustrates the way the variable name represents a memory location.



Variable Definitions

- There are many different types of data, which we will learn about in this course.
- A variable holds a specific type of data.
- **All variables must be defined before they can be used**.
- The variable definition specifies the type of data a variable can hold, and the variable name.
- The keyword **`double`** specifies that these variables can hold double-precision floating point numbers.

```
double hours, rate, pay;
```

What is A Program Made of?

- Common elements in programming languages:
  - **Lines**
    - A “line” is just that—a single line as it appears in the body of a program.
    - Most of the lines of a program contain something meaningful.
    - However, some of the lines are empty. The blank lines are only there to make the program
    - more readable.
  - **Statements**
    - A statement is a complete instruction that causes the computer to perform some action.
    - Here is the statement that appears in our example program:

```
cout << "How many hours did you work? ";
```

    - Statements can be a combination of keywords, operators, and programmer-defined symbols.
    - Statements often occupy only one line in a program, but sometimes they are spread out over more than one line.

DOUGLAS COLLEGE

Program Structure

Program Structure

- The three primary activities of a program are **input**, **processing**, and **output**.
- Input is information a program collects from the outside world. It can be sent to the program from the user, who is entering data at the keyboard or using the mouse. It can also be read from disk files or hardware devices connected to the computer.
- Our example program allows the user to enter two pieces of information: the number of hours worked and the hourly pay rate:

```
cin >> hours;
cin >> rate;
```

- Once information is gathered from the outside world, a program usually processes it in some manner:

```
pay = hours * rate;
```

- The following lines all perform output:

```
cout << "How many hours did you work? ";
cout << "How much do you get paid per hour? ";
cout << "You have earned $" << pay << endl;
```

Program Structure

- The three primary activities of a program are **input**, **processing**, and **output**.
- Input is information a program collects from the outside world. It can be sent to the program from the user, who is entering data at the keyboard or using the mouse. It can also be read from disk files or hardware devices connected to the computer.
- Our example program allows the user to enter two pieces of information: the number of hours worked and the hourly pay rate:

```
cin >> hours;
cin >> rate;
```

Extraction Operator

- Once information is gathered from the outside world, a program usually processes it in some manner:

```
pay = hours * rate;
```

- The following lines all perform output:

```
cout << "How many hours did you work? ";
cout << "How much do you get paid per hour? ";
cout << "You have earned $" << pay << endl;
```

Insertion Operator

Program Structure

- Three steps that a program typically performs:
  1. Gather input data:
     - from keyboard
     - from files on disk drives
  2. Process the input data
  3. Display the results as output:
     - send it to the screen
     - write to a file

DOUGLAS COLLEGE

The Programming Process

The Programming Process

1. Clearly define what the program is to do.
2. Visualize the program running on the computer.
3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.
5. Type the code, save it, and compile it.
6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.
9. Validate the results of the program.

DOUGLAS COLLEGE

Parts of A C++ Program

The Parts of A C++ Program

```
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, world!";

    return 0;
}
```

The Parts of A C++ Program

```
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, world!";
    return 0;
}
```

comment

preprocessor directive

which namespace to use

beginning of function named **main**

beginning of block for **main**

output statement

string literal

Send **0** to operating system

end of block for **main**

Important Notes

NOTE: C++ is a **case-sensitive language**. That means it views uppercase letters as entirely different characters than their lowercase counterparts. In C++, the name of the function **`main`** must be written in all lowercase letters. C++ does not recognize "Main" the same as "main," or "INT" the same as "int." **This is true for all the C++ keywords**.

NOTE: Make sure you have a closing brace for every opening brace in your program!

NOTE: the **`cout`** statement is the only line in the program that causes anything to be printed on the screen. The other lines, like **`#include <iostream>`** and **`int main()`**, are necessary for the framework of your program, but they do not cause any screen output.

Special Characters

- Our sample program has many special characters.
- Here is a brief summary of how they are used.

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

DOUGLAS COLLEGE

The `cout` Object

DOUGLAS COLLEGE

The `cout` Object

Pronounced "see out"
or
"Console Out"

The cout Object

- The **cout** object (from the **std** namespace within the **iostream** library) displays output on the computer screen.
- We can use the stream **insertion operator** **<<** to send output to **cout**:

```
cout << "I am a Belieber!";
```

- More than one item can be sent to cout for printing onto the console screen:

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

- Or:

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

The cout Object

- The **cout** object (from the **std** namespace within the **iostream** library) displays output on the computer screen.
- We can use the stream **insertion operator** **<<** to send output to **cout**:

```
cout << "I am a Belieber!";
```

- More than one item can be sent to cout for printing onto the console screen:

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

Operator cascading

- Or:

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

The cout Object

- An important concept to understand about the following example is that, although the output is broken up into two programming statements, **this program will still display the message on a single line**.

```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Programming is ";
    cout << "great fun!";
    return 0;
}
```

The endl Manipulator and the \n Escape Sequence

- We can use the **endl** manipulator to start a new line of output. This will produce two lines of output.
- We do **not** put quotation marks around **endl**.
- The last character in **endl** is a lowercase L, not the number **1**.
- We can also use the **\n** escape sequence to start a new line of output.

```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Programming is " << endl;
    cout << "great fun!";
    return 0;
}
```

```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Programming is\n";
    cout << "great fun!";
    return 0;
}
```

The endl Manipulator and the \n Escape Sequence

- We can use the **endl** manipulator to start a new line of output. This will produce two lines of output.
- We do **not** put quotation marks around **endl**.
- The last character in **endl** is a lowercase L, not the number **1**.
- We can also use the **\n** escape sequence to start a new line of output.

Notice that the **\n** is **INSIDE** the string literal.

```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Programming is " << endl;
    cout << "great fun!";
    return 0;
}
```

```
// A simple C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Programming is\n";
    cout << "great fun!";
    return 0;
}
```

Common Escape Sequences

- There are many escape sequences in C++.
- When you type an escape sequence in a string, you type two characters (a backslash followed by another character). However, **an escape sequence is stored in memory as a single character**.
- Here are a few useful ones:

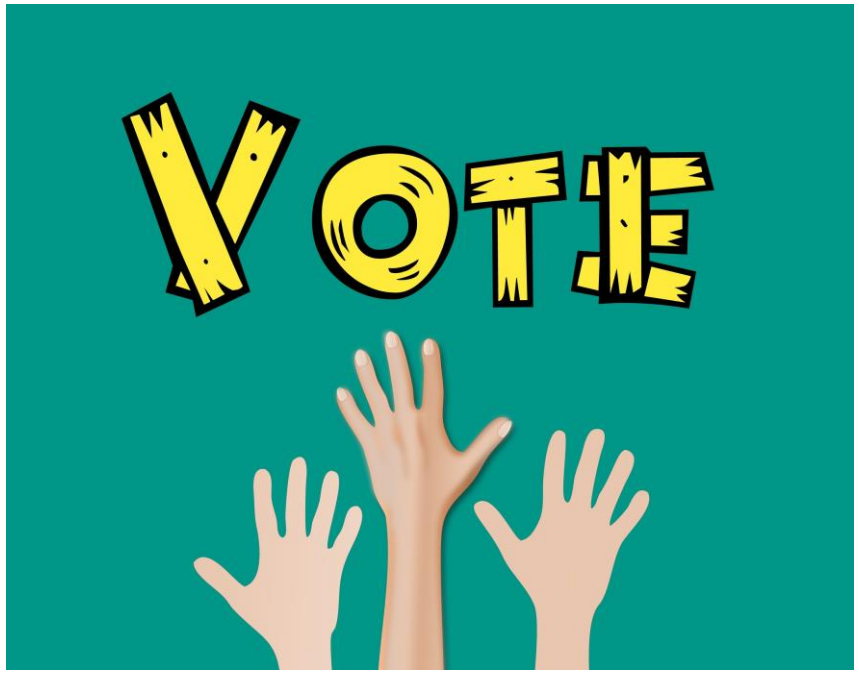
Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing.
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop.
`\a`	Alarm	Causes the computer to beep.
`\b`	Backspace	Causes the cursor to back up, or move left one position.
`\r`	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.

Poll 2 (Extra Credit)

In C++, it is okay to add a space between the backslash and the subsequent character within any escape sequence.

For example, `"Doja \n Cat"` **can safely be replaced with** `"Doja \ n Cat"`**.**

a) True
b) False
c) Not sure!



Please use the “Poll” window to participate for extra credit! One answer only please!

DOUGLAS COLLEGE

The `#include` Directive

The `#include` Directive

- The **`#include`** directive inserts the contents of another file into the program.
- This is a preprocessor directive, not part of C++ language.
- An **`#include`** directive must always contain the name of a file.
- The preprocessor inserts the entire contents of the file into the program at the point it encounters the #include directive.
- As a result, the **`#include`** lines are **not** seen by compiler.
- Do **not** place a semicolon at end of **`#include`** line!

Plan for Today

- Variables and Literals
- Identifiers
- Integer Data Types
- The **`char`** Data Type
- The C++ **`string`** Class
- Floating Point Data Types
- The Boolean Data Type
- Determining the Size of a Variable or Data Type
- Variable Assignment and Initialization
- Variable Scope
- Arithmetic Operators
- Named Constants

In-Class Lab Exercise #1

Write a C++ program that will display your name on the first line, your street address on the second line, your city, state, and ZIP code on the third line, and your telephone number on the fourth line. Place a comment with today's date at the top of the program. Test your program to ensure it works as expected!

Once you are done, name your program "Lab1_1.cpp" and upload it to the Blackboard. Do not hit "Submit" yet, as there are more parts to complete later on.

DOUGLAS COLLEGE

Variables and Literals

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:

```
int item;
```

- The word **`int`** in the above example stands for integer, so **`item`** can only be used to hold integer numbers.
- We must have a definition for every variable you intend to use in a program.
- In C++, variable definitions can appear at any point in the program (but "good coding practices" will guide us in determining where to put them in our programs).

Variables and Literals

```
// This program has a variable.
#include <iostream>
using namespace std;

int main()
{
    int number;
    number = 5;

    cout << "The value in number is " << number << endl;

    return 0;
}
```

Variable Definition

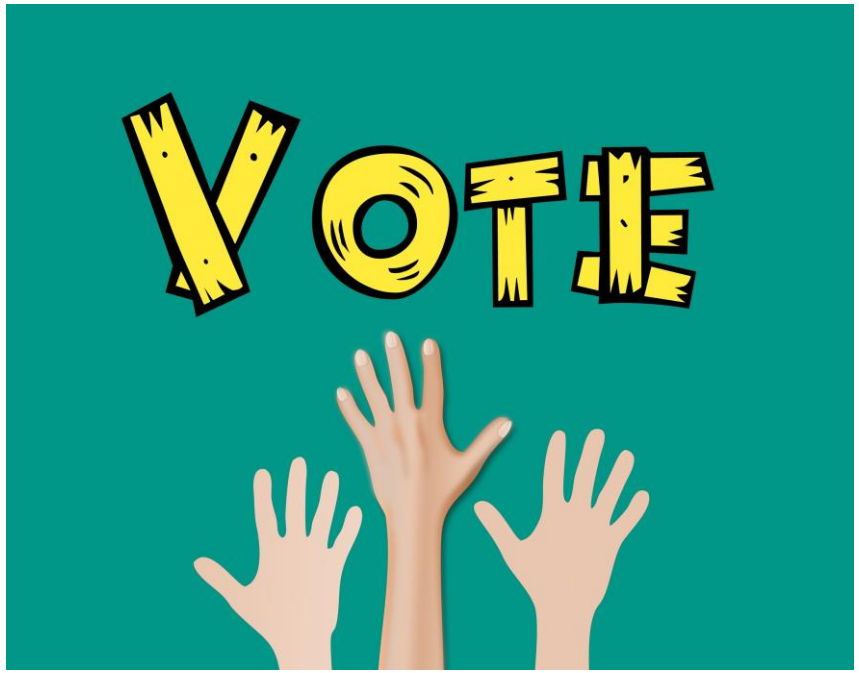
Variable Assignment

Poll 3 (Extra Credit)

The following statement alone will print the value of 5 onto the console screen.

```
number = 5;
```

a) True
b) False
c) Not sure!



Please use the "Poll" window to participate for extra credit! One answer only please!

Variables and Literals

```
// This program has a variable.
#include <iostream>
using namespace std;

int main()
{
    int number;

    number = 5;

    cout << "The value in number is " << number << endl;

    return 0;
}
```

Variable Definition

Variable Assignment

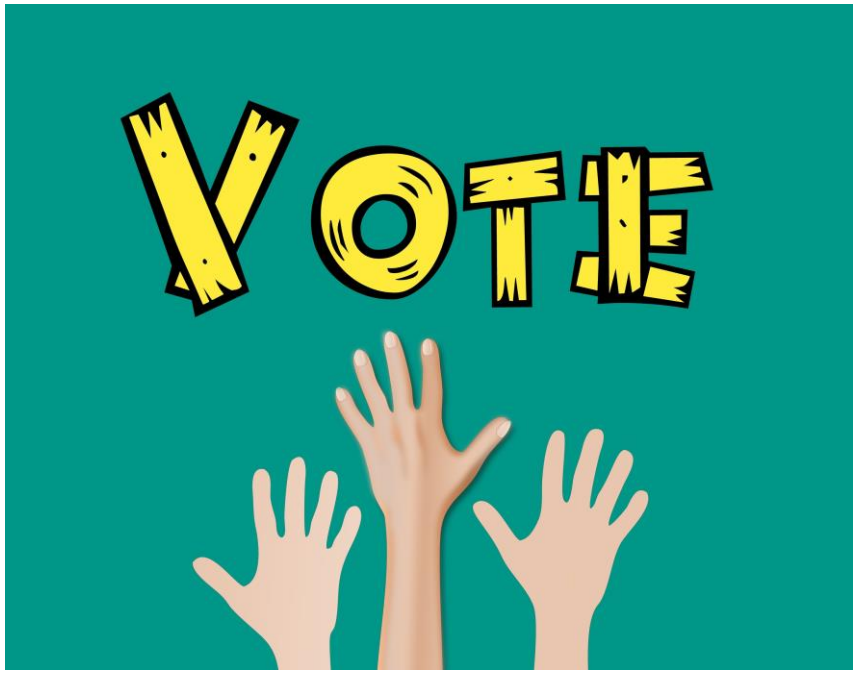
Prints the value of number onto the console screen.

Poll 3 (Extra Credit)

The following statement will also print the value of number onto the console screen.

```
cout << "The value in number is " << "number" << endl;
```

a) True
b) False
c) Not sure!



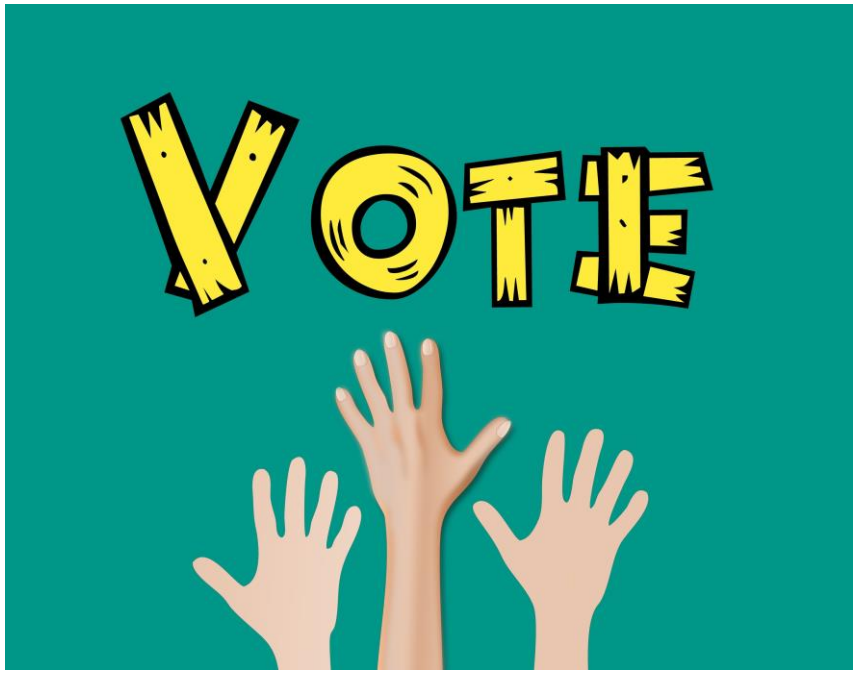
Please use the "Poll" window to participate for extra credit! One answer only please!

Poll 3 (Extra Credit)

The following statement will assign a value of 5 to variable `number`**.**

```
Number = "5";
```

a) True
b) False
c) Not sure!



Please use the "Poll" window to participate for extra credit! One answer only please!

Literals

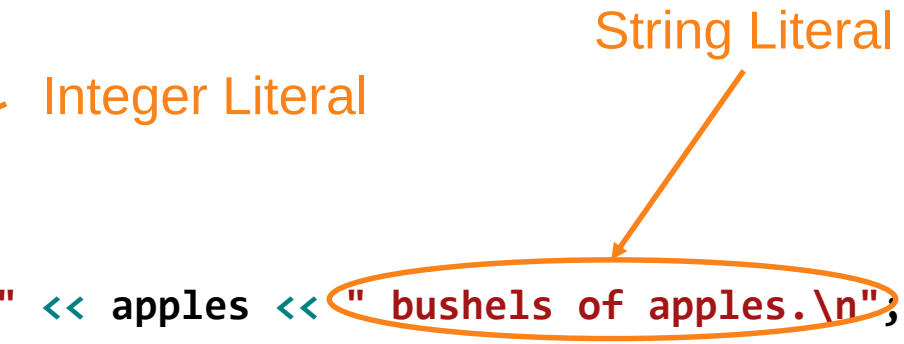
- **Literal**: a value that is written into a program's code.
  - **`"hello, there"`** (string literal)
  - **`12`** (integer literal)
- Literals are also sometimes called **`constants`**.

```
// This program has literals and a variable.
#include <iostream>
using namespace std;

int main()
{
    int apples;

    apples = 20;
    cout << "Today we sold " << apples << " bushels of apples.\n";
    return 0;
}
```

Literals

- **Literal**: a value that is written into a program's code.
  - **`"hello, there"`** (string literal)
  - **`12`** (integer literal)
- Literals are also sometimes called **`constants`**.

```
// This program has literals and a variable.
#include <iostream>
using namespace std;

int main()
{
    int apples;

    apples = 20;
    cout << "Today we sold " << apples << " bushels of apples.\n";
    return 0;
}
```



In-Class Lab Exercise #2

There are a number of syntax errors in the following program. Fix all of them to ensure it works as expected.

Once you are done, name your program "Lab1_2.cpp" and upload it to the Blackboard. Do not hit "Submit" yet, as there are more parts to complete later on.

```
* / What's wrong with this program? /*
#include iostream
using namespace std;
int main();
}
int a, b, c \\ Three integers
    a = 3
    b = 4
    c = a + b
    Cout < "The value of c is %d" < C;
    return 0;
{
```

DOUGLAS COLLEGE

Identifiers

Variables

- An **identifier** is a programmer-defined name for some part of a program:
  - E.g., variables, functions, etc.
- We may choose our own variable names in C++, as long as we do **not** use any of the C++ keywords (because keywords make up the "core" of the language and have specific purposes).
- We should always choose names for our variables that give an indication of what the variables are used for. We may be tempted to define variables with names like this:

```
int x;
```

- The nondescript name, **x**, gives no clue as to the variable's purpose. Here is a better alternative:

```
int itemsOrdered;
```

Variables

- Notice the mixture of uppercase and lowercase letters in the variable name **`itemsOrdered`**.
- Although **all of C++'s key words must be written in lowercase**, we may use uppercase letters in variable names.
- The reason the **`O`** in **`itemsOrdered`** is capitalized is to improve readability because **we cannot insert a space within a variable name**.
- This style of coding is not required from a syntax perspective.
- We are free to use all lowercase letters, all uppercase letters, or any combination of both.
- In fact, some programmers use the underscore character to separate words in a variable name:

```
items_ordered
```

Identifier Rules

- The first character of an identifier must be an alphabetic character or and underscore (**_**).
- After the first character, we may use alphabetic characters, numbers, or underscore characters.
- Upper- and lowercase characters are distinct (due to the case-sensitivity of C++).
- Keywords are reserved and cannot be used for variable names.

IDENTIFIER	VALID?	REASON IF INVALID
totalSales	Yes	
total_Sales	Yes	
total.Sales	No	Cannot contain **.**
4thQtrSales	No	Cannot begin with digit
totalSale$	No	Cannot contain **$**

C++ Keywords

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

We cannot use any of the C++ key words as an identifier. These words have reserved meanings!

DOUGLAS COLLEGE

Integer Data Types

Variables

- There are many different types of data. Variables are classified according to their data type, which determines the kind of information that may be stored in them.
- Integer variables can hold whole numbers such as **12**, **7**, and **-99**.

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

Defining Variables

- Variables of the **same type** can be defined
  - On separate lines:
    - `int length;`
    - `int width;`
    - `unsigned int area;`
  - On the same line:
    - `int length, width;`
    - `unsigned int area;`
- **Variables of different types must be in different definitions**.

Integer Literals

- Integer literals are stored in memory as **`int`** by default
- To store an integer constant in a **`long`** memory location, put **`L`** at the end of the number:

  `1234L`
- To store an integer constant in a **`long long`** memory location, put **`LL`** at the end of the number:

  `324LL`
- Constants that begin with **`0`** (zero) are base-8: **`075`**
- Constants that begin with **`0x`** are base-16: **`0x75A`**

Example Program

```
// This program has variables of several of the integer types.
#include <iostream>
using namespace std;

int main()
{
    int checking;
    unsigned int miles;
    long diameter;

    checking = -20;
    miles = 4276;
    diameter = 100000;
    cout << "We have made a long journey of " << miles;
    cout << " miles.\n";
    cout << "Our checking account balance is " << checking;
    cout << "\nThe galaxy is about " << diameter;
    cout << " light years in diameter.\n";
    return 0;
}
```

DOUGLAS COLLEGE

The `char` Data Type

The char Data Type

- The **char** data type is used to store individual characters.
- A variable of the **char** data type can hold only one character at a time.
- Here is an example:

```
char letter;
```

- In C++, **character literals are enclosed in single quotation marks**.
- Here is an example showing how we would assign a character to the letter variable:

```
letter = 'g';
```

- Since **char** variables can hold only one character, they are not compatible with strings, so we cannot assign a string literal to a char variable:

```
letter = "g"; // ERROR! Cannot assign a string to a char
```

- **Do not confuse character literals, which are enclosed in single quotation marks, with string literals, which are enclosed in double quotation marks!**

Example

```
// This program works with characters.
#include <iostream>
using namespace std;

int main()
{
    char letter;

    letter = 'A';
    cout << letter << endl;
    letter = 'B';
    cout << letter << endl;
    return 0;
}
```

The char Data Type

- Although the **char** data type is used for storing characters, **it is actually an integer data type that typically uses 1 byte of memory**!
- The reason an integer data type is used to store characters is because **characters are internally represented by numbers**.
- Each printable character, as well as many nonprintable characters, is assigned a unique number.
- The most commonly used method for encoding characters is **ASCII**, which stands for the **American Standard Code for Information Interchange**.
- When a character is stored in memory, it is actually the numeric code that is stored.
- When the computer is instructed to print the value on the screen, it displays the character that corresponds with the numeric code.

Example

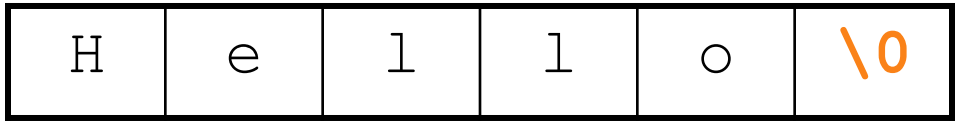
```
// This program demonstrates the close relationship between
// characters and integers.
#include <iostream>
using namespace std;

int main()
{
    char letter;

    letter = 65;
    cout << letter << endl;
    letter = 66;
    cout << letter << endl;
    return 0;
}
```

String Literals

- A string literal is in fact a series of characters in consecutive memory locations:

  "Hello"

- String literals are stored in RAM with the **null-terminating character**, **'\0'**, at the end:



- Do not confuse the **null terminator** with the character **'0'**. They have two unique ASCII codes!

- C++ **automatically** places the null terminator at the end of string literals.

DOUGLAS COLLEGE

The C++ `string` Class

The C++ `string` Class

- Since a **`char`** variable can store only one character in its memory location, another data type is needed for a variable able to hold an entire string.
- Although C++ does **not** have a built-in data type able to do this, standard C++ provides something called the **`string`** class that allows the programmer to create a **`string`** type variable.
- The first step in using the **`string`** class is to **`#include`** the string header file.
  ```
  #include <string>
  ```
- We can then define **`string`** variables in our programs:
  ```
  string firstName, lastName;
  ```
- A **`string`** object in C++ can conveniently receive values with assignment operator **`=`**:
  ```
  firstName = "George";
  lastName = "Washington";
  ```
- A **`string`** object can also be displayed using a **`cout`** with the insertion operator **`<<`**:
  ```
  cout << firstName << " " << lastName;
  ```

Example

```
// This program demonstrates the string class.
#include <iostream>
#include <string> // Required for the string class.
using namespace std;

int main()
{
    string movieTitle;

    movieTitle = "Top Gun";
    cout << "My favorite movie is " << movieTitle << endl;
    return 0;
}
```

DOUGLAS COLLEGE

Floating Point Data Types

Floating Point Data Types

- Floating-point data types are used to define variables that can hold **real** numbers.
- The floating-point data types are:
  - **`float`**
  - **`double`**
  - **`long double`**
- They can hold real numbers such as:

  `12.45` **`-3.8`**
- The **`float`** data type is considered **single-precision**. The **`double`** data type is usually twice as big as **`float`**, so it is considered **double-precision**. The **`long double`** is intended to be larger than the **`double`**.
- The exact sizes of these data types are dependent on the computer system we are using.
- **All floating-point numbers are signed**.

Data Type	Key Word	Description
Single precision	`float`	4 bytes. Numbers between ±3.4E-38 and ±3.4E38
Double precision	`double`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308
Long double precision	`long double*`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308

Floating Point Literals (Constants)

- Floating point literals can be represented in two ways:
  - **Fixed point (decimal) notation**:

```
31.4159                    0.0000625
```

  - **E-notation**:

```
3.14159E1                  6.25e-5
```

- Floating point literals are **`double`** by default in C++.
- To force a floating point literal to be of type **`float`**, add **`f`** or **`F`** to the end:

```
3.14159f
```

- for long double, add **`L`**:

```
0.0000625L
```

Example

```
// This program uses floating point data types.
#include <iostream>
using namespace std;

int main()
{
    float distance;
    double mass;

    distance = 1.495979E11;
    mass = 1.989E30;
    cout << "The Sun is " << distance << " meters away.\n";
    cout << "The Sun\'s mass is " << mass << " kilograms.\n";
    return 0;
}
```

In-Class Lab Exercise #3: Truncation

Write a C++ program that demonstrates what happens when a floating point literal is assigned to an integer variable.

Once you are done, name your program "Lab1_3.cpp" and upload it to the Blackboard. Do not hit "Submit" yet, as there are more parts to complete later on.

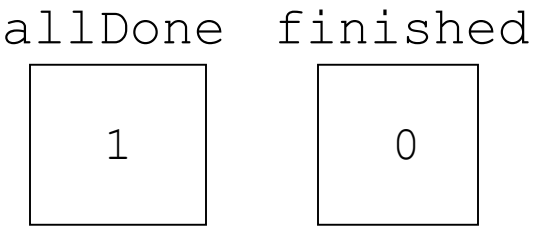
DOUGLAS COLLEGE

The `bool` Data Type

Boolean Variables

- Boolean variables are set to either **`true`** or **`false`**.
- Expressions that have a **`true`** or **`false`** value are called **Boolean expressions**, named in honor of English mathematician George Boole (1815–1864).
- **`bool`** variables are stored as small integers:
  - false is represented by **`0`**, true by **`1`**:

```
bool allDone = true;
bool finished = false;
```



Example

```
// This program demonstrates boolean variables.
#include <iostream>
using namespace std;

int main()
{
    bool boolValue;

    boolValue = true;
    cout << boolValue << endl;
    boolValue = false;
    cout << boolValue << endl;
    return 0;
}
```

DOUGLAS COLLEGE

Determining the Size of a Variable or Data Type

Determining the Size of a Variable or Data Type

- The **`sizeof()`** operator gives the size of any data type or variable:

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

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

int main()
{
    long double apple;

    cout << "The size of an integer is " << sizeof(int);
    cout << " bytes.\n";
    cout << "The size of a long integer is " << sizeof(long);
    cout << " bytes.\n";
    cout << "An apple can be eaten in " << sizeof(apple);
    cout << " bytes!\n";
    return 0;
}
```

DOUGLAS COLLEGE

Variable Assignment and Initialization

Variable Assignment and Initialization

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

```
item = 12;
```

Variable Assignment and Initialization

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

```
item = 12;
```

Lvalue (left-value) **Rvalue (right-value)**

Variable Assignment and Initialization

- An **assignment statement** uses the assignment 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 assignment **=** operator.
- This will **not** work:

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

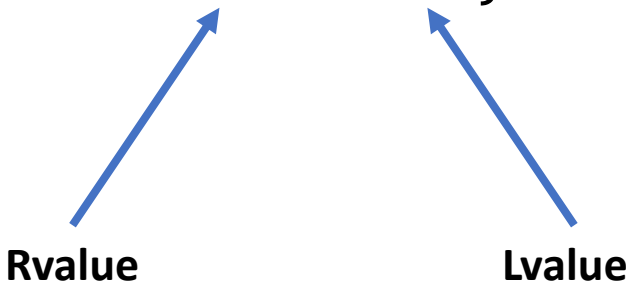
Variable Assignment and Initialization

- An **assignment statement** uses the assignment 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 assignment **=** operator.
- This will **not** work:

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



Variable Assignment and Initialization

- An **assignment statement** uses the assignment 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 assignment **=** operator.
- This will **not** work:

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

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

```
int length = 12;
```

- We can also initialize some or all variables, as shown below:

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

DOUGLAS COLLEGE

Variable Scope

Variable Scope

- **Scope of a variable**:
  - The part of the program in which the variable can be accessed.
- A variable **cannot** be used before it is defined.

```
// This program can't find its variable.
#include <iostream>
using namespace std;

int main()
{
    cout << value; // ERROR! value not defined yet!

    int value = 100;
    return 0;
}
```

DOUGLAS COLLEGE

Arithmetic Operators

Arithmetic Operators

- **Arithmetic operators** are used for performing numeric calculations.
- C++ has **unary**, **binary**, and **ternary** operators:
  - unary (1 operand), e.g. `-5`
  - binary (2 operands), e.g. `13 - 7`

Binary Arithmetic Operators

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

Arithmetic Operators

- **Arithmetic operators** are used for performing numeric calculations.
- C++ has **unary**, **binary**, and **ternary** operators:
  - unary (1 operand), e.g. `-5`
  - binary (2 operands), e.g. `13 - 7`

Binary Arithmetic Operators

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

Requires both operands to be integers!

Arithmetic Operators

- **Arithmetic operators** are used for performing numeric calculations.
- C++ has **unary**, **binary**, and **ternary** operators:
  - unary (1 operand), e.g. `-5`
  - binary (2 operands), e.g. `13 - 7`

Binary Arithmetic Operators

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

If either operand is floating point, returns floating point. Otherwise, integer.

Requires both operands to be integers!

DOUGLAS COLLEGE

Named Constants (Constant Variables)

Variable Assignment and Initialization

- Literals in C++ may be given names that symbolically represent them in a program.
- Imagine the following statement appears in a banking program that calculates data pertaining to loans:

```
amount = balance * 0.069;
```

- First, it is not clear to anyone other than the original programmer what **0.069** is. It appears to be an interest rate, but we cannot be sure.
- The second problem occurs if this number is used in other calculations throughout the program and must be changed periodically.
- Both of these problems can be addressed by using **named constants**.
- A **named constant** is like a variable, but **its content is read-only and cannot be changed while the program is running**.

```
const double INTEREST_RATE = 0.069;
```

References

- Kernighan, Brian W. and Dennis M. Ritchie,*The C Programming Language*, Prentice Ha, 1988.
- Hanly, Jeri R. and Elliot B. Koffman, *C Program Design for Engineers*, Addison Wesley, 2001.
- Lafore, Robert, *Object-Oriented Programming in C++*, SAMS, 2001.
- Gaddis, Tony, *Starting Out with C++ from Control Structures to Objects*, Pearson, 2021.

Thank you.

DOUGLAS COLLEGE

DOUGLAS COLLEGE

The CPU

- The CPU is comprised of:
  - **The Control Unit:**
    - Retrieves and decodes program instructions.
    - Coordinates activities of all other parts of computer.
  - **Arithmetic & Logic Unit:**
    - Hardware optimized for high-speed numeric calculation.
    - Hardware designed for true/false, yes/no decisions.