COMP1011
Programming Fundamentals

Lecture 1 Introduction to Computer Processing and C++ Programming

Lecture 1

- Why Computers?
- Computer Organization
- Computer Languages
- C++
- Your First Program in C++
- Variables and Operators

Computers are Everywhere

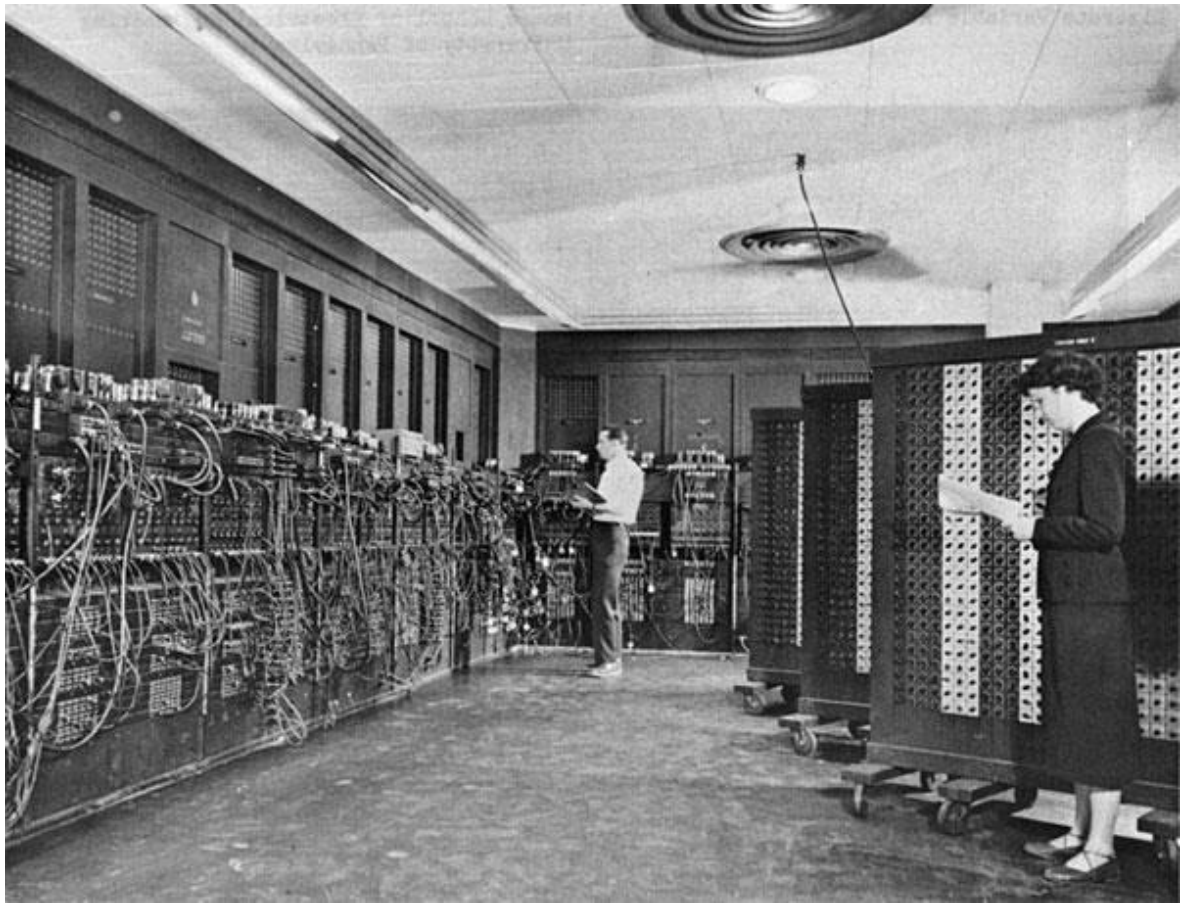
Sources: http://staff.brighton.ac.uk/is/computing/SiteAssets/Pages/Staff%20computers/Staff_computers/computers.png (Retrieved in 2021)
https://images.wondershare.com/data-recovery/mobile-data-recovery.jpg

Why Computers?

- Computer
  - A device capable of performing the following two major functions
    1. Computations
    2. Making logical decisions
- Two major components
  - Hardware
    - Various devices comprising computer
      - monitor, keyboard, mouse, speaker, camera, microphone, memory modules, CPU, etc.
  - Software
    - Programs that run on computer, i.e., sets of instructions to command computer to perform actions and make decisions
    - Computer programs are “subset” of software

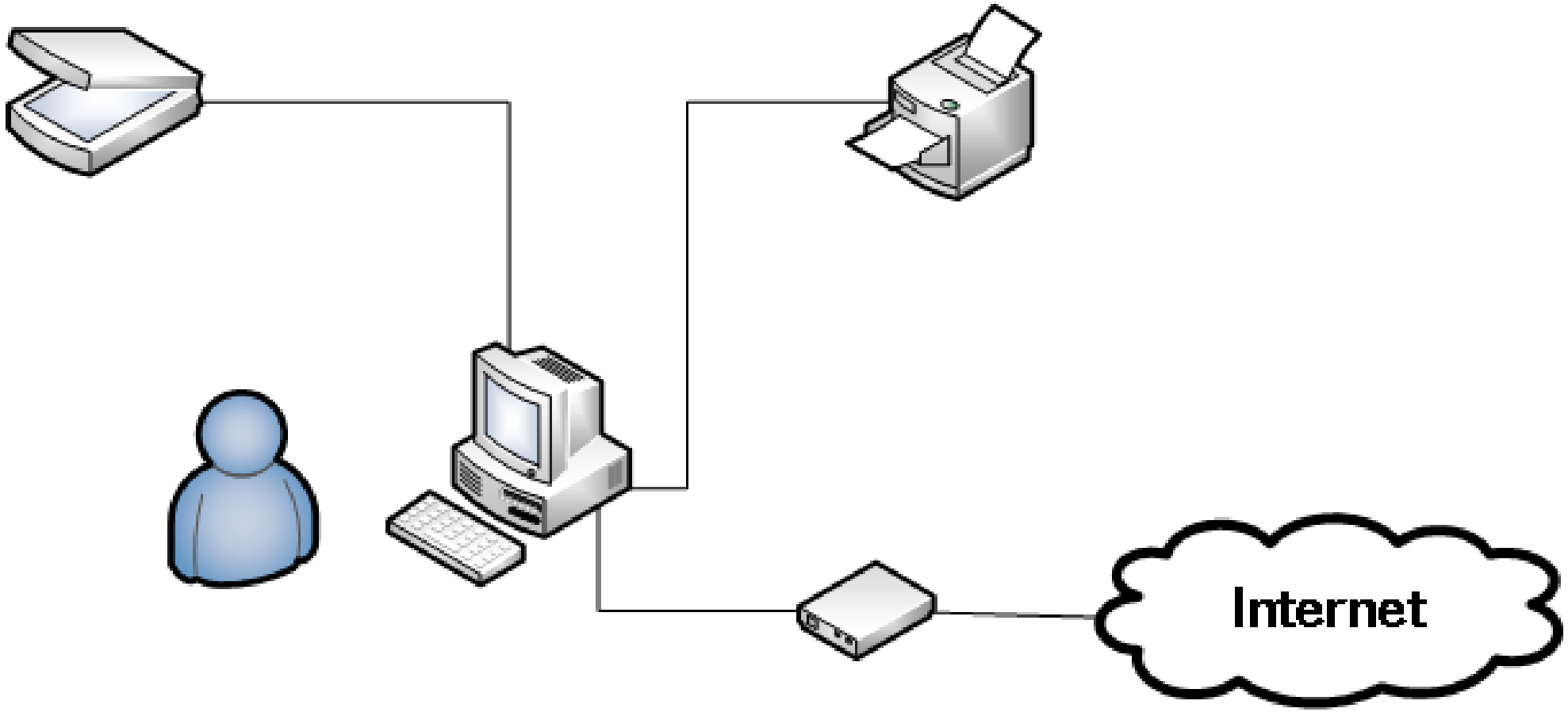
Why were Computers invented?

- **ENIAC (Electronic Numerical Integrator and Computer)** was the world's first general-purpose electronic digital computer
  - Designed in 1943 and was completed in 1946
- Planned to be used to compute the range and trajectory tables for new weapons during World War II



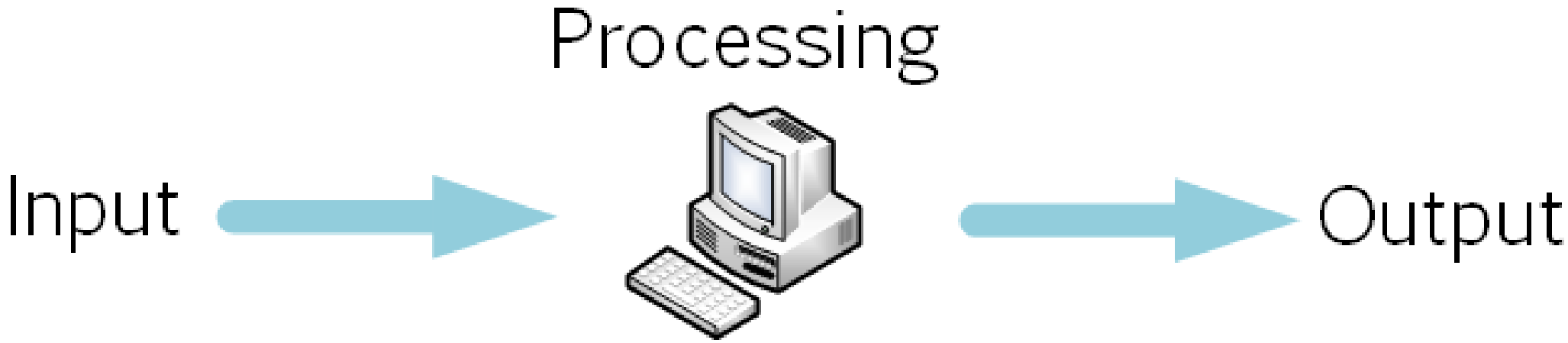
Source: https://www.computerhope.com/jargon/e/eniac.htm

Computer Organization



Computer Organization

› Input/Output (I/O) Model



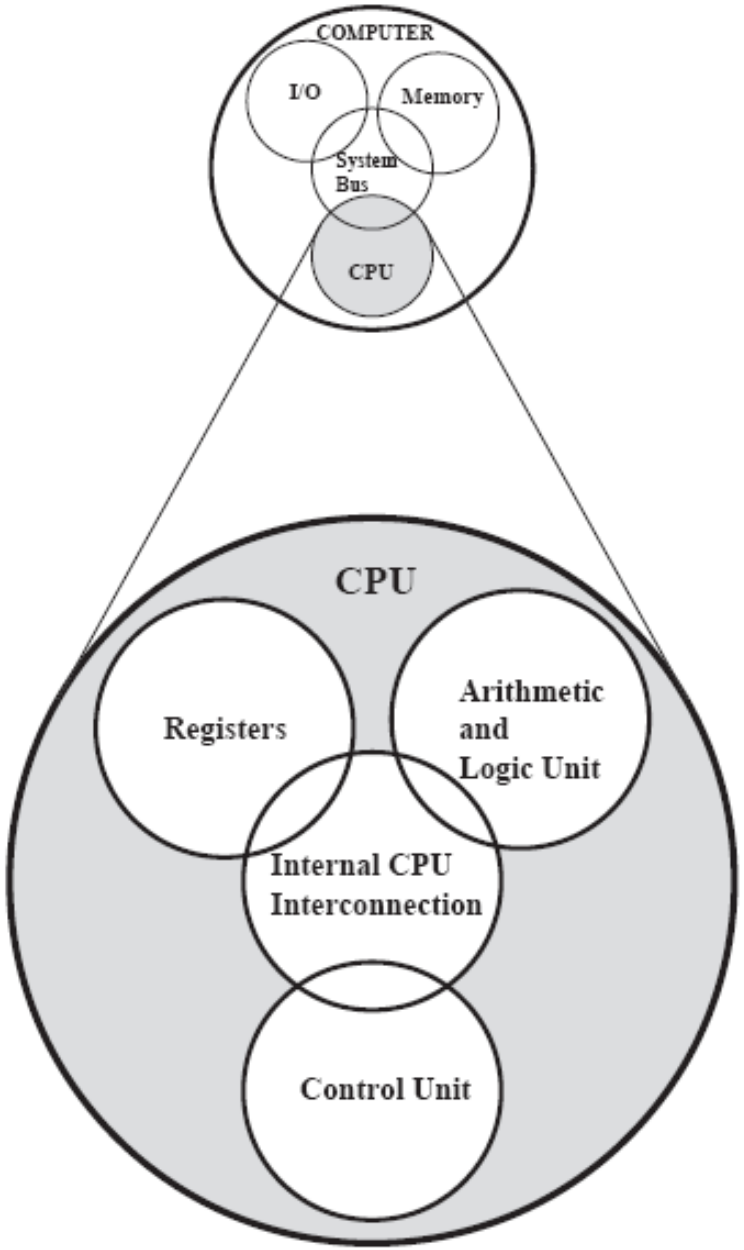
Computer Organization

- Six logical units
  1. **Input unit**
     - "Receiving" section
     - Obtains information from input devices
       - keyboard, mouse, microphone, scanner, networks, etc.
  2. **Output unit**
     - "Shipping" section
     - Takes information processed by computer
     - Places information on output devices
       - screen, printer, networks, etc.
- Note: Information may be used to control other devices (becomes input of other devices)

Image source: http://www.octopus.com.hk/en/business/become-a-merchant/reader-writer/retail/index.html

Computer Organization

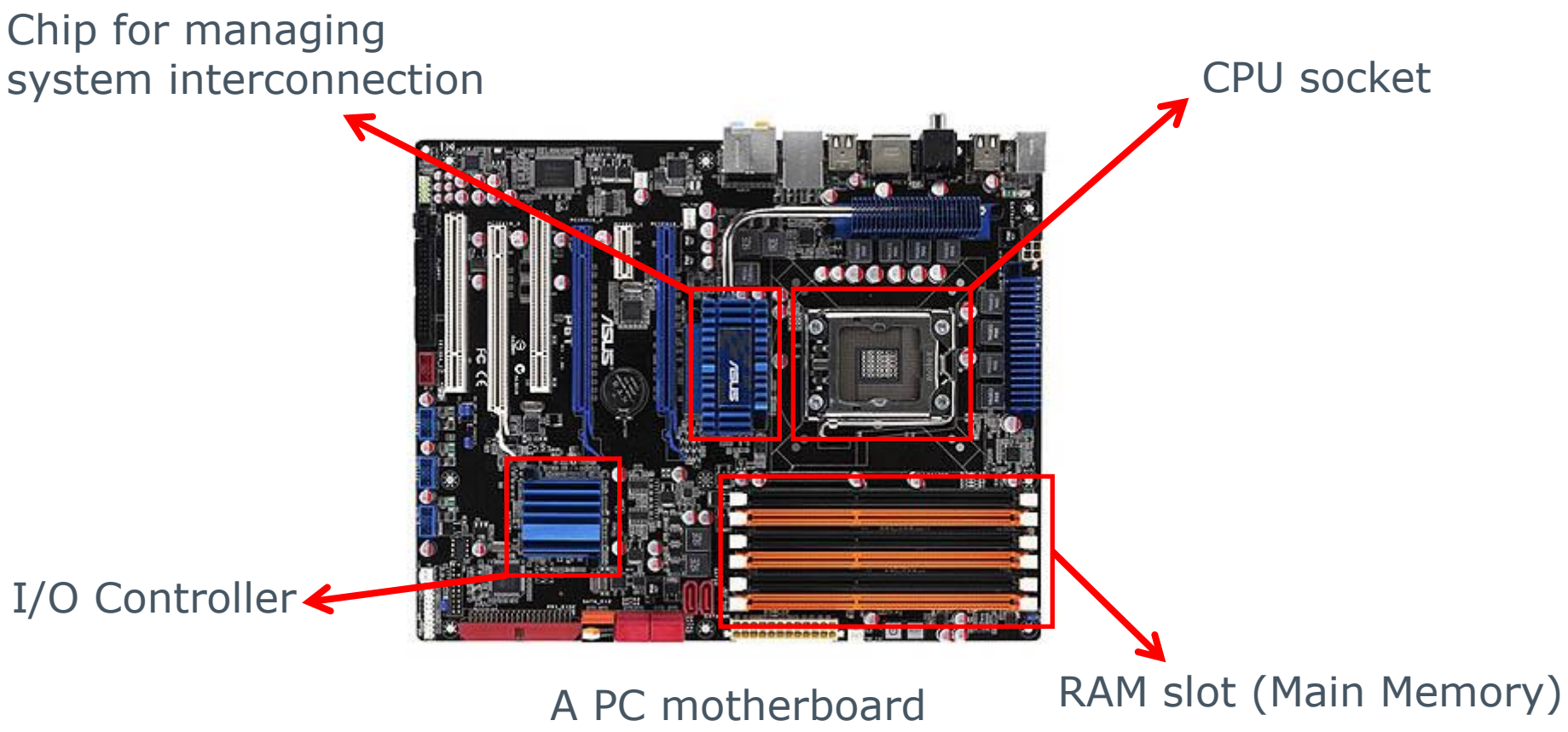
3. **Memory unit**
   - "Warehouse" section
   - Rapid access, relatively low capacity
   - Retains information from input unit
     - Immediately available for processing
   - Retains processed information
     - Until placed on output devices
   - E.g., Random Access Memory (RAM), Registers
4. **Arithmetic and logic unit (ALU)**
   - "Manufacturing" section
   - Performs arithmetic calculations and logical decisions



Computer Organization

5. **Control Unit**
   - "Administrative" section
   - Supervises and coordinates other sections of computer
6. **Secondary storage unit**
   - Long-term, high-capacity "warehouse" section
   - Storage
     - Inactive programs or data (e.g., document files, game saves)
   - Secondary storage devices
     - Magnetic harddisk, USB flash memory, CD, DVD, Blu-ray
   - Persistent
   - Longer to access than primary memory
   - Less expensive per unit than primary memory
   - Sometimes write-once and read-many

Physical Structure



Source: http://www.asus.com/websites/Global/products/QtpKQuERkuYw6trc/EDOWPJm2Oj7Nuyla_500.jpg

What is Programming?

- The process of arranging ("writing") instructions to form programs that are executable by computers to solve information/computational problems
  - Also known as **coding**
- Who can do it?
  - You!
- Programs can be written in various ***programming languages***
  - Some are directly understandable by computers
  - Others require intermediate translation steps

Computer Languages

1. **Machine language**
   - The only language that computer can understand
   - "Natural language" of computer
   - Defined by hardware design
     - Machine-dependent
   - Generally consist of strings of binary numbers
     - **0**s and **1**s
   - Instruct computers to perform elementary operation
   - One at a time (E.g., ADD and MOVE)
     - Cumbersome for humans to read and understand
   - Example,

Computer Languages

2. **Assembly language**
   - English-like abbreviations representing elementary computer operations
   - More intuitive to humans
   - Incomprehensible to computers
     - Require a translator (essentially a program), called *assembler*, to convert it to machine language
   - E.g., to calculate the gross pay of an employee

```
LOAD        BASEPAY
ADD         OVERTIMEPAY
STORE       GROSSPAY
```

Computer Languages

3. **High-level languages**
   - Similar to English, use common mathematical notations
   - A single statement accomplishes substantial tasks
     - Assembly language requires many instructions to accomplish simple tasks of human
   - Translator programs (compilers)
     - Convert to machine language
   - Interpreter programs
     - Directly execute high-level language programs
   - E.g.,

```
grossPay = basePay + overTimePay
```

C++

- A high-level language
- Use C++ to study ***Structured Programming*** in this course
  - A disciplined approach to write programs
- Top-down design model
  - A large problem is divided into smaller problems
  - Each small problem is solved separately in terms of *code segment/function*
  - Solutions of smaller problems are integrated to form the final program
  - Better understanding and easier maintenance by programmers

History of C++

- Extension of C
- Early 1980s: Bjarne Stroustrup (Bell Laboratories)
- “Spruces up” C
- Provides capabilities for object-oriented programming (OOP)
- Hybrid language
  - C-like style
  - Object-oriented style
  - Both

Other Programming Language Examples

- Java
- Visual Basic
- C#
- Python
- FORTRAN
- COBOL
- RPG

Basics of a Typical C++ Environment

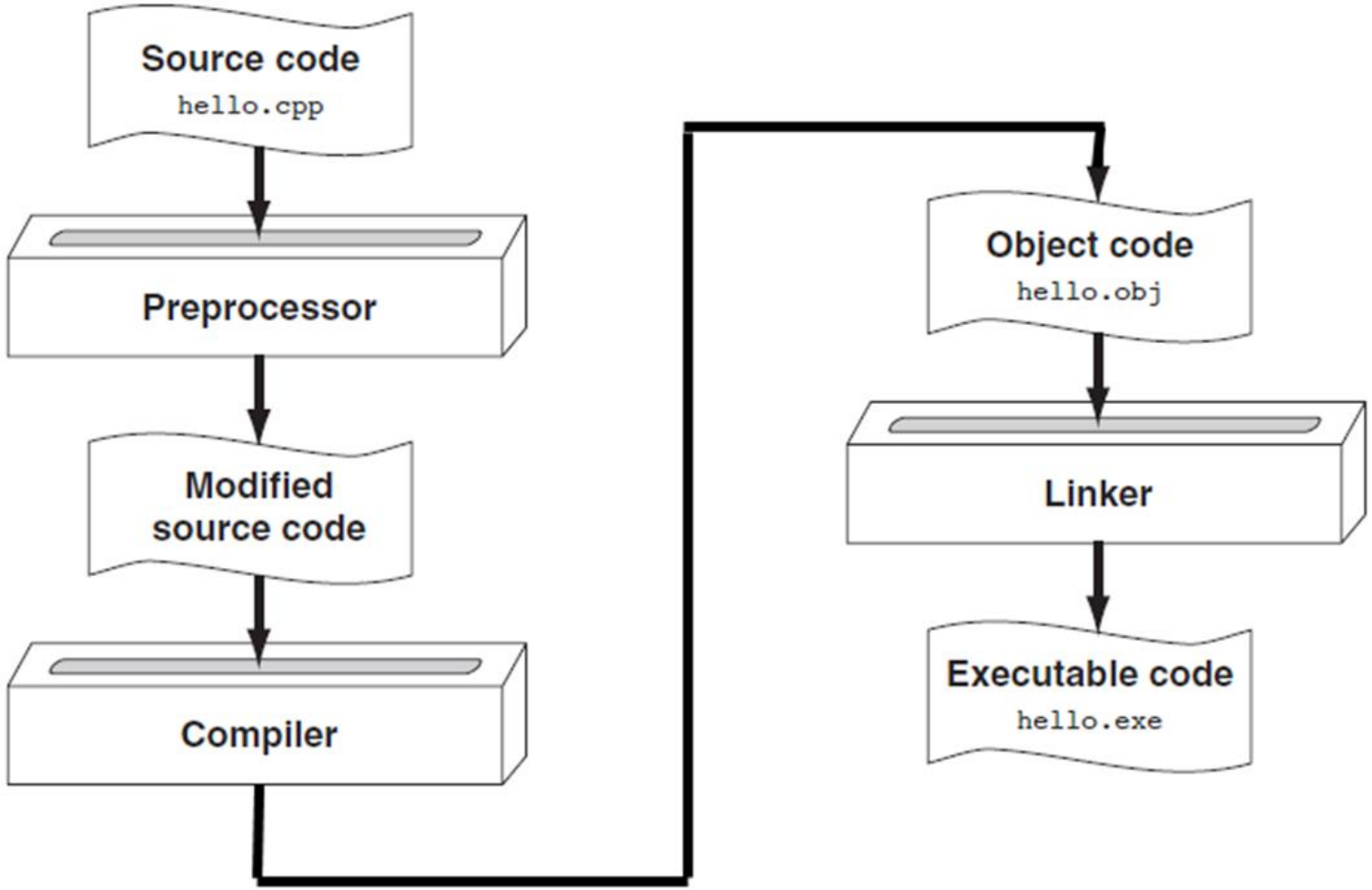
- Three main components
  1. Program Development Environment
  2. Language (Syntax)
  3. C++ Standard Library

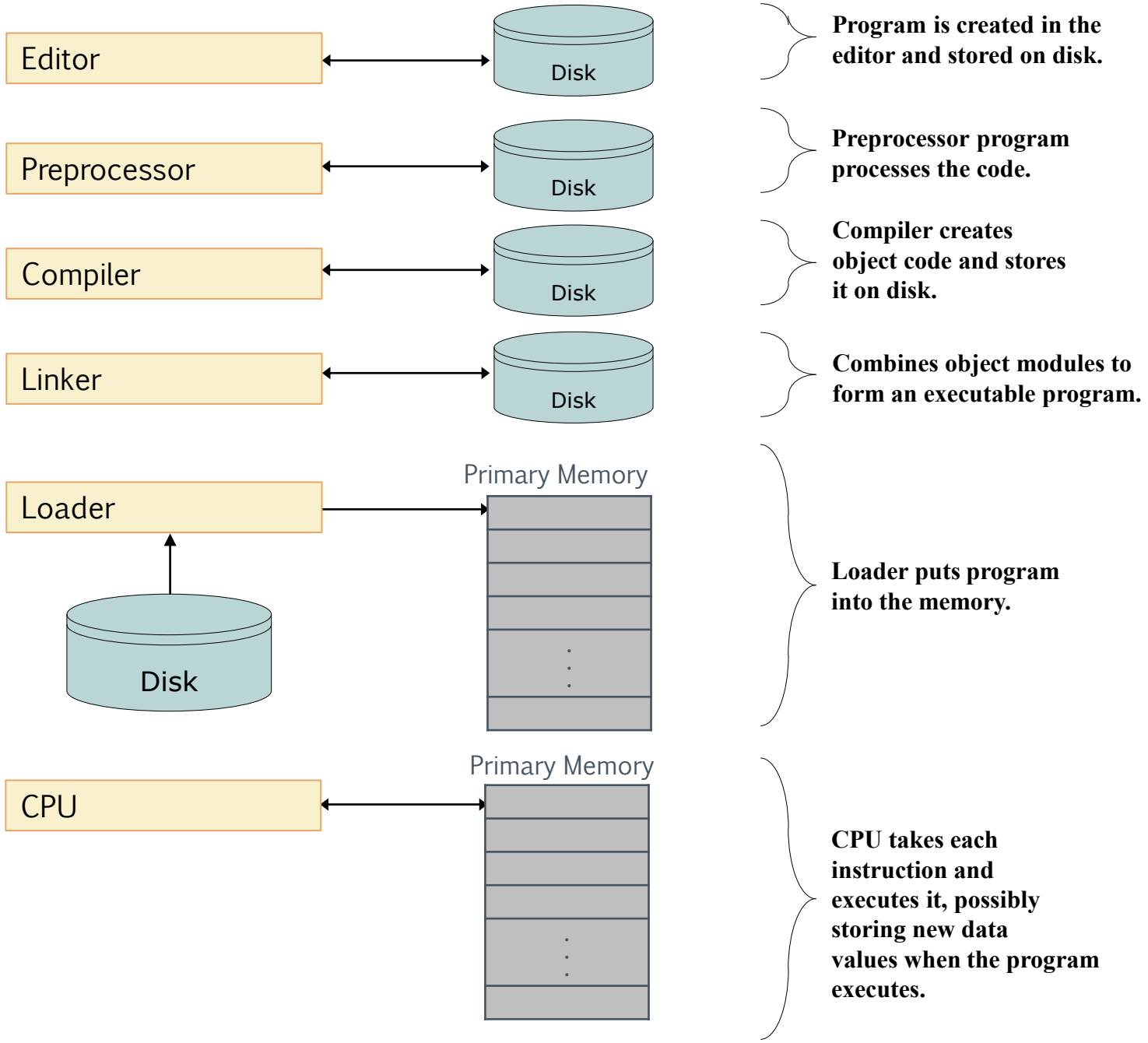
A system of symbols and rules for writing programs

Basics of a Typical C++ Environment

- Six Phases of forming a C++ Program
  - Edit
    - Program is created in a text-based editor and stored on disk
  - Preprocess
    - A program named, Preprocessor, processes the code
    - E.g., to replace certain tokens with string (text) or numerical values
  - Compile
    - Compiler creates object code and stores it on disk
  - Link
    - Combines library/other functions to form an executable program
  - Load
    - Loader puts program in memory
  - Execute
    - CPU takes each instruction and executes it, possibly storing new data values to memory as the program executes

Basics of a Typical C++ Environment




Editor
Disk
Program is created in the editor and stored on disk.
Preprocessor
Disk
Preprocessor program processes the code.
Compiler
Disk
Compiler creates object code and stores it on disk.
Linker
Disk
Combines object modules to form an executable program.
Primary Memory
Loader
Disk
Loader puts program into the memory.
Primary Memory
CPU
CPU takes each instruction and executes it, possibly storing new data values when the program executes.

Your First Program in C++

welcome.cpp

```
// Your first program in C++
#include <iostream>

using namespace std;

// function main begins program execution
int main() {

    cout << "Welcome to C++!\n";
    return 0; // indicate that program ended successfully

}
```

```
Welcome to C++!
```

Your First Program in C++

```
// Your first program in C++
```

- Comments start with: **//**
  - Comments are ignored by compiler
- Traditional comments (in C): **/* ... */**

```
/* This is a traditional C-style
   comment. It can be
   split over multiple lines */
```

- Improve program readability

```
#include <iostream>
```

- Preprocessing directive
  - Tells the preprocessor to perform a specific action
  - Format: a line begins with **#**
  - Where to put a directive?
    - Anywhere in a source file, with effect from that point onwards
  - **#include <filename>** means:
    - To add the contents of the library file – filename to the source program during preprocessing
    - **< >** means to find the library file in the default location

Your First Program in C++

```

```

- Blank line
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

```
using namespace std;
```

- Namespace
  - **a set of names in which all names are unique**
  - to prevent ambiguity of names
    - E.g., **`cout`** is defined in the namespace of **`std`** in **`iostream`**
  - If this line is not written explicitly, we can write like this

`std::cout`

Your First Program in C++

```
// function main begins program execution
```

- Another blank line and comment
  - Ignored by compiler

```
int main() {
```

- C++ programs begin executing at **`main`**
  - Parenthesis **`()`** indicates **`main`** is a function
  - C++ programs contain one or more functions
- Functions can perform tasks and return information
  - **`int`** means **`main`** returns an integer when the **`main`** function finishes execution (more on this later)
- Left brace **`{`** begins body of function declaration
  - Ended by right brace **`}`**

Your First Program in C++

```
cout << "Welcome to C++!\n";
```

- A program statement
  - instructs the computer to perform a task
  - must be ended with semicolon **;**
- Standard output stream object
  - **cout**
    - "Connected" to screen
  - **<<**
    - Stream insertion operator
    - Value on the right-hand side (right operand) inserted into output stream

Your First Program in C++

```
cout << "Welcome to C++!\n";
```

- Escape characters
  - \
  - Indicates “special” character output

Escape Sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line (system dependent).
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\"	Double quote. Used to print a double quote character.

Your First Program in C++

- Input/output
  - **`cout`**
    - Standard output stream
    - “The computer screen”
  - **`cin`**
    - Standard input stream
    - Normally from the keyboard

Your First Program in C++

```
return 0; // indicate that program ended successfully
```

- End of the **`main`** function
- Return an integer
- 0 indicates that program ended successfully

```
}
```

- A right brace
- Signifies the end of the **`main`** function

Printing Text on Screen

welcome2.cpp

```
// Printing a line with multiple statements.
#include <iostream>

using namespace std;

// function main begins program execution
int main() {

    cout << "Welcome ";
    cout << "to C++!\n";

    return 0;   // indicate that program ended successfully

}
```

```
Welcome to C++!
```

Printing Text on Screen

welcome3.cpp

```
// Printing multiple lines with a single statement.
#include <iostream>

using namespace std;

// function main begins program execution
int main() {

    cout << "Welcome\nto\n\nC++!\n";

    return 0;   // indicate that program ended successfully

}
```

```
Welcome
to

C++!
```

Introduction to Variables

- Remember one aim of programming is to perform mathematical calculations
- Some calculations may require intermediate steps and therefore there are intermediate values for subsequent calculations
- How do we store these intermediate values in programming?

    -> Variables

Variables

- Areas in memory where values can be stored
  - Informal thought: A box that can be stored a value
- Common data types
  - **`int`** – integer numbers (e.g., -3, 0, 8)
  - **`char`** – characters and symbols (e.g., H, d, ?, %)
  - **`double`** – floating point numbers (e.g., 3.14, 1.618)
- **Declare variables** with *name* and *data type* before use (Imagine as creating a box before putting something into it)

```
int integer1;
int integer2;
int sum;
```

- Can declare several variables of the same type in one declaration
  - Comma-separated list

```
int integer1, integer2, sum;
```

Variables

- Variable names (also known as *identifier*)
  - Rules of naming a variable
    - Series of characters
      - letters, digits, underscores (_)
      - No space in-between
        - E.g., thickness_of_wall, baseArea, flat_widthRatio
    - Cannot begin with digit
      - `7button` is invalid
    - Case sensitive
      - `a1` and `A1` are two different variables
  - As a norm, variable names begin with lowercase letters

```
int sum;    // the norm
int Sum;    // ok, but not recommended
```

C++ Keywords

› Cannot be used as identifiers (variable names)

C++ Keywords				
Keywords common to the C and C++ programming languages				
auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			
C++ only keywords				
asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	using	virtual
wchar_t				

More Operators

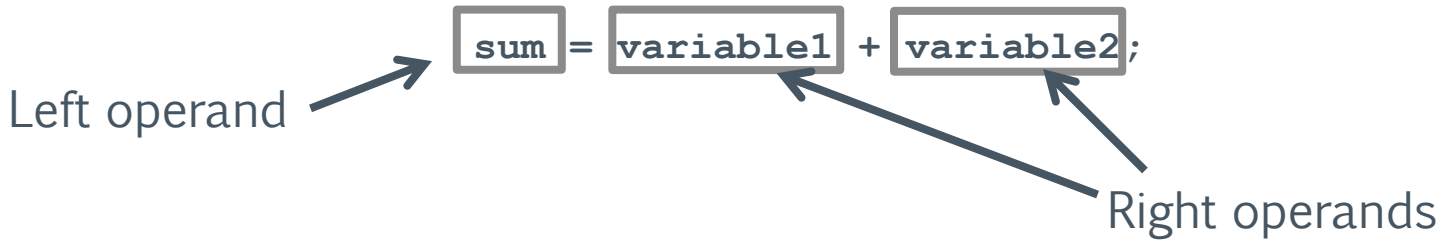
>> **(stream extraction operator)**

- Used with **`cin`**
- Also in the namespace **`std`**
- Waits for user to input value, then press Enter key
- Stores value in variable to right of operator
- E.g.,

```
cin >> radius
```

= **(assignment operator)**

- Assigns value to variable (from right to left)
- Binary operator (two operands)
- E.g.,



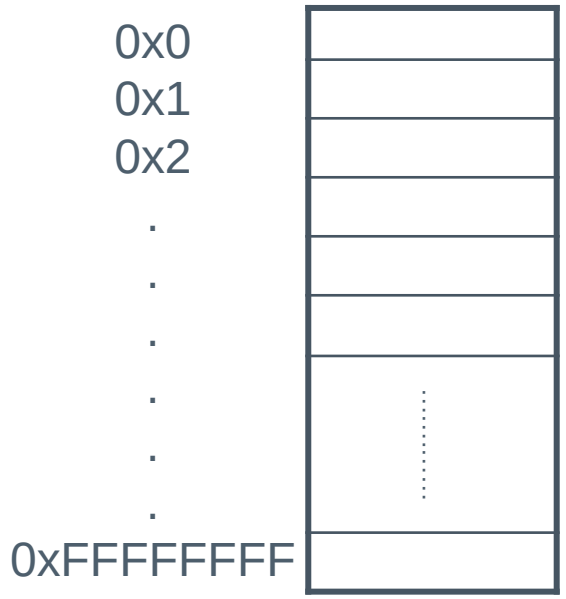
sum.cpp

```cpp
// A program for summing two input number.
#include <iostream>

using namespace std;

// function main begins program execution
int main() {

    int integer1;  // first number to be input by user
    int integer2;  // second number to be input by user
    int sum;       // variable in which sum will be stored

    cout << "Enter first integer\n";  // prompt
    cin >> integer1;                  // read an integer

    cout << "Enter second integer\n"; // prompt
    cin >> integer2;                  // read an integer

    sum = integer1 + integer2;  // assign result to sum

    cout << "Sum is " << sum << endl; // print sum

    return 0;   // indicate that program ended successfully
}
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Variable and Memory

- Variable names
  - Correspond to actual locations in computer's memory
  - Every variable has **name**, **type**, **size** and **value**
  - When new value placed into variable, overwrites previous value
  - Reading variables from memory non-destructive



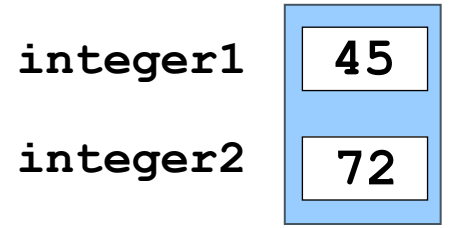
Variable and Memory (A high-level illustration)

```
cin >> integer1;
```

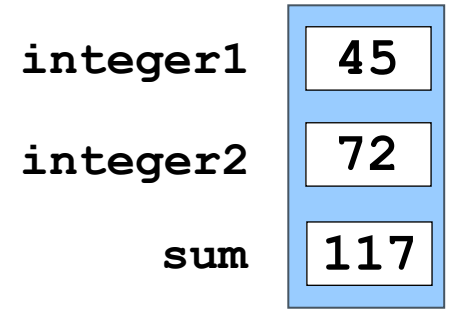
– Assume user entered 45


```
cin >> integer2;
```

– Assume user entered 72




```
sum = integer1 + integer2;
```



Variable and Memory (A high-level illustration)

- Note
  - **integer1**, **integer2** and **sum** are created in the program and memory spaces are allocated to them (even they are not used yet)
  - In most C++ environment, variables are not initialized to default values (e.g., 0, NULL)
    - Try

```
int integer1;
cout << integer1 << endl;
```

    - What do you realize? Why?

Other Arithmetic Operators

- `*`
  - Multiplication
- `/`
  - Division
  - Integer division truncates the decimal value
    - `7 / 5` evaluates to `1`
- `%`
  - Modulus operator returns remainder
    - `7 % 5` evaluates to `2`

arithmetic.cpp

```cpp
// A program showing arithmetic
#include <iostream>

using namespace std;

int main() {

    int num1 = 65;
    int num2, answer;

    num2 = 5;

    answer = num1 + num2;  // addition
    cout << "Answer is " << answer << endl;

    answer = num1 - num2;  // subtraction
    cout << "Answer is " << answer << endl;

    answer = num1 * num2;  // multiplication
    cout << "Answer is " << answer << endl;

    answer = num1 / num2;  // division
    cout << "Answer is " << answer << endl;

    return 0;

}
```

```
Answer is 70
Answer is 60
Answer is 325
Answer is 13
```

Operator Precedence

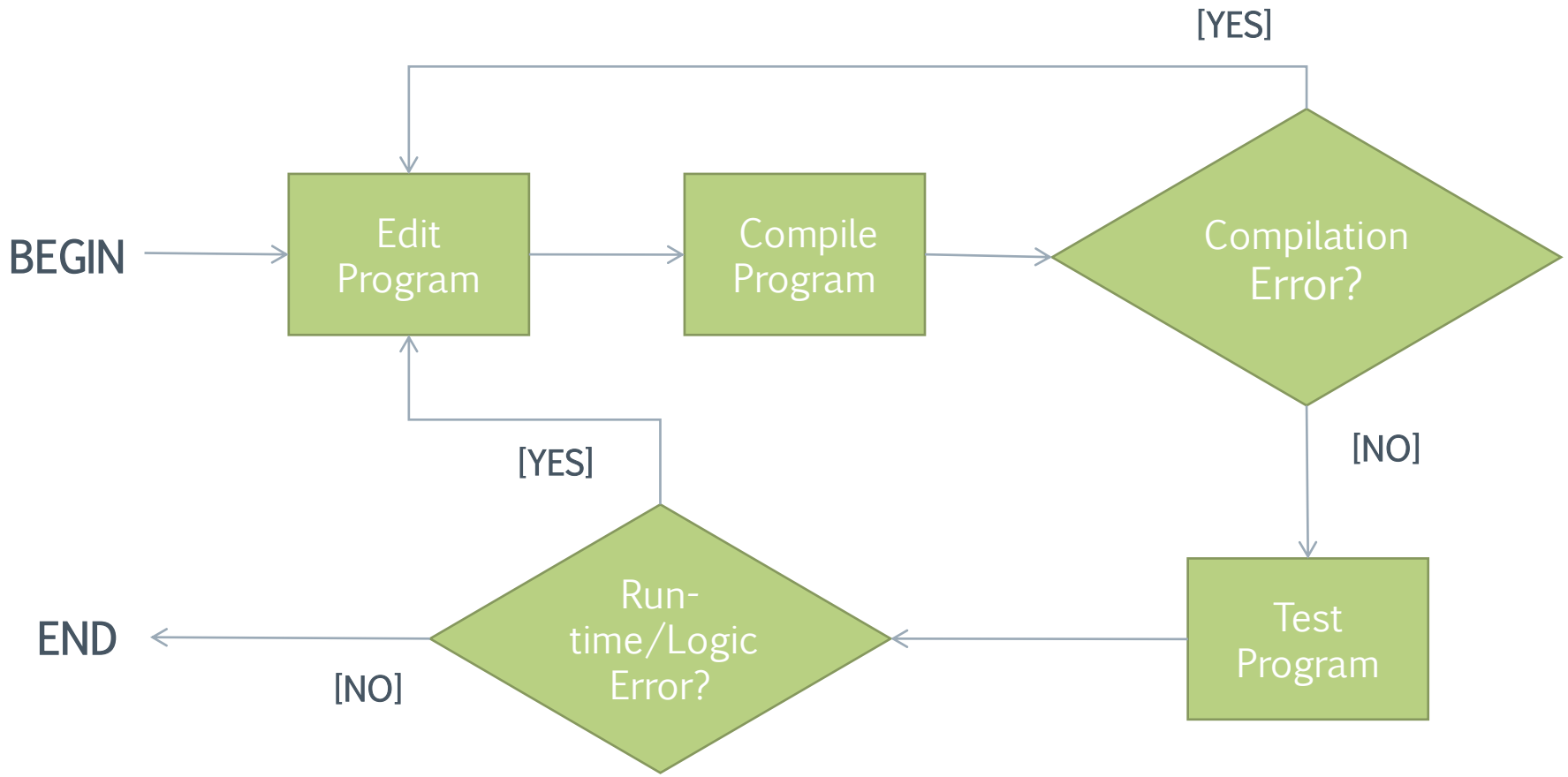
- E.g.,

```
number1 + number2 * (number3 % number4 - number5)
```

  - How to evaluate above?

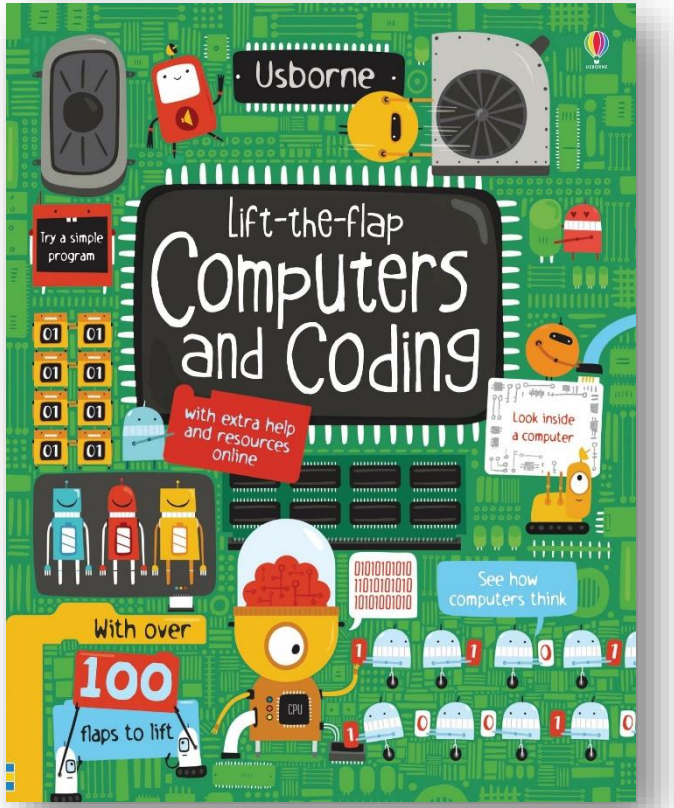
- Rules
  - Operators in parentheses evaluated first
    - Nested/embedded parentheses
      - Operators in innermost pair first
  - Multiplication, division, modulus applied next
    - Operators applied from left to right
  - Addition, subtraction applied last
    - Operators applied from left to right

What do you always do in this course?



Further Reading (for fun)

› Lift-the-Flap Computers and Coding, Usborne.



Summary

- Why Computers?
- Computer Organization
- Computer Languages
- C++
- Your First Program in C++
- Variables and Operators