Lecture 02

M M Imran

Evolution of Programming Languages

- A computer program is a set of instructions that the computer can perform in order to perform some task.
- The process of creating a program is called programming.
- Programmers typically create programs by producing source code.
- The collection of physical computer parts that make up a computer and execute programs is called the hardware.
- When a computer program is loaded into memory and the hardware sequentially executes each instruction, this is called running or executing the program.

Evolution of Programming Languages

- Machine Language
- Assembly Language
- High-level Languages

Machine Language

- A computer's CPU is incapable of speaking C++.
- The limited set of instructions that a CPU can understand directly is called machine code (or machine language or an instruction set).
- Here is a sample machine language instruction: **10110000 01100001**
- Back when computers were first invented, programmers had to write programs directly in machine language, which was a very difficult and time consuming thing to do.

Machine Language

- Each instruction is composed of a sequence of 1s and 0s. Each individual 0 or 1 is called a binary digit, or bit for short.
- The number of bits that make up a single command vary. Some CPUs process instructions that are always 32 bits long, whereas some other CPUs (x86 family) have instructions that can be a variable length.
- Each set of binary digits is interpreted by the CPU into a command to do a very specific job, such as compare these two numbers, or put this number in that memory location.
- However, because different CPUs have different instruction sets, instructions that were written for one CPU type could not be used on a CPU that didn’t share the same instruction set.
- This meant programs generally weren’t portable means the code had to be written all over again.

Assembly Language

- Because machine language is so hard for humans to read and understand, assembly language was invented.
- In an assembly language, each instruction is identified by a short abbreviation (rather than a set of bits), and names and other numbers can be used.
- Here is the same instruction as above in assembly language: **mov al, 061h**
- This makes assembly much easier to read and write than machine language.
- CPU can not understand assembly language directly. Instead, the assembly program must be translated into machine language before it can be executed by the computer. This is done by using a program called an assembler.
- Programs written in assembly languages tend to be very fast, and assembly is still used today when speed is critical.

Assembly Language

- Requires a lot of instructions to do even simple tasks.
- While the individual instructions themselves are somewhat human readable, understanding what an entire program is doing can be challenging (it’s a bit like trying to understand a sentence by looking at each letter individually).
- Assembly language still isn’t very portable. a program written in assembly for one CPU will likely not work on hardware that uses a different instruction set, and would have to be rewritten or extensively modified.

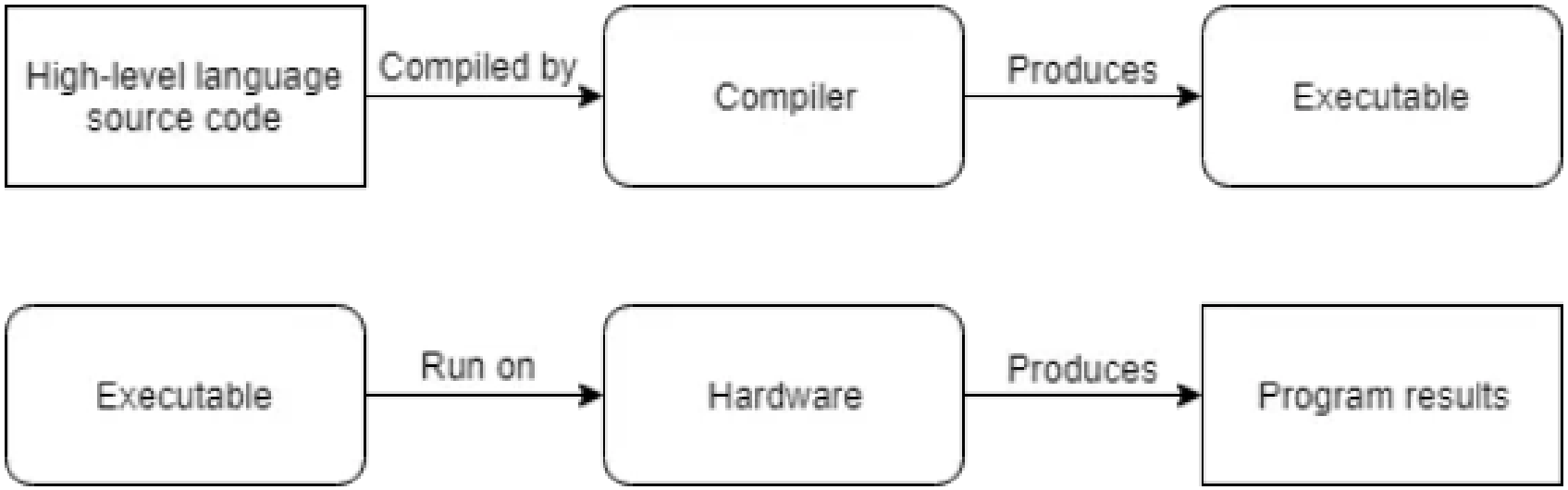
High-level Languages

- To address the readability and portability concerns, new programming languages such as C, C++, Pascal (and later, languages such as Java, Javascript, and Perl) were developed. These languages are called high level languages.
- They are designed to allow the programmer to write programs without having to be as concerned about what kind of computer the program will be run on.
- Here is the same instruction as above in C/C++: **a = 97;**
- Much like assembly programs, programs written in high level languages must be translated into a format the computer can understand before they can be run. There are two primary ways this is done: **compiling** and **interpreting**.

Compiler

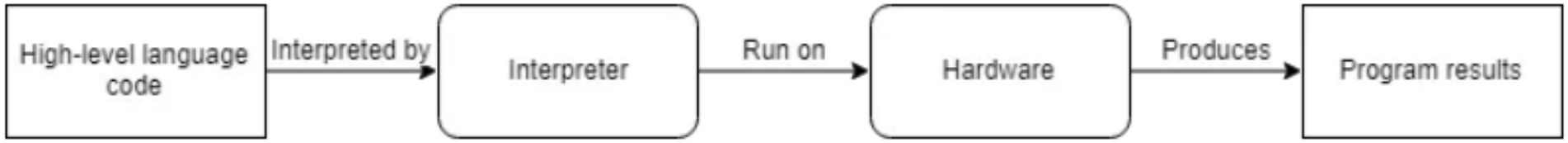
- A compiler is a program that reads source code and produces a stand-alone **executable program** that can then be run.
- Once your code has been turned into an executable, you do not need the compiler to run the program.
- In the beginning, compilers were primitive and produced slow, unoptimized code. However, over the years, compilers have become very good at producing fast, optimized code, and in some cases can do a better job than humans can in assembly language!

Compiler


High-level language source code
Compiled by
Compiler
Produces
Executable
Executable
Run on
Hardware
Produces
Program results

Interpreter

- An interpreter is a program that directly executes the instructions in the source code without requiring them to be compiled into an executable first.
- Interpreters tend to be more flexible than compilers, but are less efficient when running programs because the interpreting process needs to be done every time the program is run. This means the interpreter is needed every time the program is run.



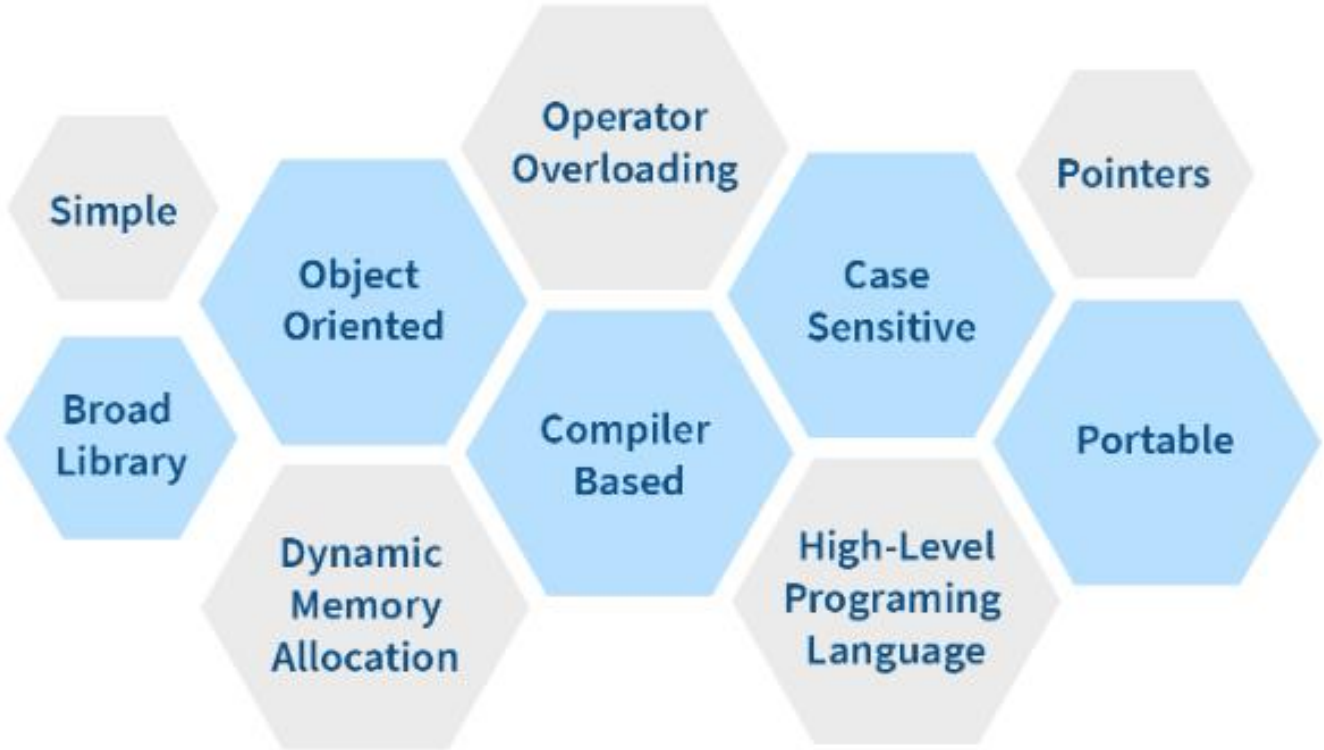
High level language properties

- First, high level languages are much easier to read and write because the commands are closer to natural language that we use every day.
- Second, high level languages require fewer instructions to perform the same task as lower level languages, making programs more concise and easier to understand. In C++ you can do something like a = b * 2 + 5; in one line. In assembly language, this would take 5 or 6 different instructions.
- Third, programs can be compiled (or interpreted) for many different systems, and you don't have to change the program to run on different CPUs (you just recompile for that CPU)
- Most languages can be compiled or interpreted, however, traditionally languages like C, C++, and Pascal are compiled, whereas "scripting" languages like Perl and Javascript tend to be interpreted.

C++ Origin

- C++ is a general-purpose, high-level programming language developed in 1979 by Bjarne Stroustrup at AT & T Bell Laboratories.
- C++ was created as an advanced version of the C programming language.
- It extended the features of C and added new ones including classes and objects, type checking, exception handling, inheritance, polymorphism, etc., to the C language.
- Over the years, the language has received several significant updates to stay in touch with modern programming languages.
- Even though C++ was created decades ago, it is widely used to develop many software programs even today.

Features of C++



Programming Languages Developed Before C++

Language Name	Developed By	Year of Origin
Algol	International Group	1960
Basic Combined Programming Language	Martin Richards	1967
B	Ken Thompson	1970
C	Dennis Ritchie	1972
K&R C	Brian Kernighan & Dennis Ritchie	1978
C++	Bjarne Stroustrup	1980

Development Steps

- Step 1: Define the problem that you would like to solve
- Step 2: Determine how you are going to solve the problem
- Step 3: Write the program

Algorithm

- An algorithm is a step-by-step approach for performing a task, or solving a problem, usually using a computer.
- An algorithm is not written in a programming language but can typically be translated into one fairly easily.
- An algorithm may already be defined and ready to use in a computer program. For example, finding the least common multiple of two numbers.
- An algorithm may need to be defined because it is unique to the computer program at hand.
- A software application typically uses one or more algorithms.

Find the perimeter and area of a rectangle

The algorithm to find the perimeter and area of the rectangle is as follows:

1. Get the length of the rectangle.

2. Get the width of the rectangle.

3. Find the perimeter using the following equation:

perimeter = 2 · (length + width)

4. Find the area using the following equation:

area = length · width

Calculating price of an item

Now, we design an algorithm that calculates the sales tax and the price of an item sold in a particular state.

The state's portion of the sales tax is 4%, and the city's portion of the sales tax is 1.5%.

If the item is a luxury item, such as a car more than $50,000, then there is a 10% luxury tax.

Algorithm of calculating price of an item

1. Get the selling price of the item.
2. Determine whether the item is a luxury item.
3. Find the state's portion of the sales tax using the formula:

stateSalesTax = salePrice · 0.04

4. Find the city's portion of the sales tax using the formula:

citySalesTax = salePrice · 0.015

5. Find the luxury tax using the following formula:

if (item is a luxury item)
luxuryTax = salePrice · 0.1
otherwise
luxuryTax = 0

6. Find salesTax using the formula:

salesTax = stateSalesTax + citySalesTax + luxuryTax

7. Find amountDue using the formula:

amountDue = salePrice + salesTax

Pseudo code

- Pseudo code is the most common way to show the steps of an algorithm.
- Pseudo code is an English-like, informal, high-level description of some algorithm.
- Pseudo code is similar to a programming language but is intended for human reading rather than machine reading.
- Pseudo code typically omits details that are not essential for human understanding of the algorithm.
- Pseudo code is often used for sketching out the structure of a program before the actual coding takes place.
- Pseudo code should document any assumptions made about the algorithm.

Pseudo code of calculating price of an item

stateSalesTax = salePrice · 0.04
citySalesTax = salePrice · 0.015

if (item is a luxury item)
 luxuryTax = salePrice · 0.1
otherwise
 luxuryTax = 0

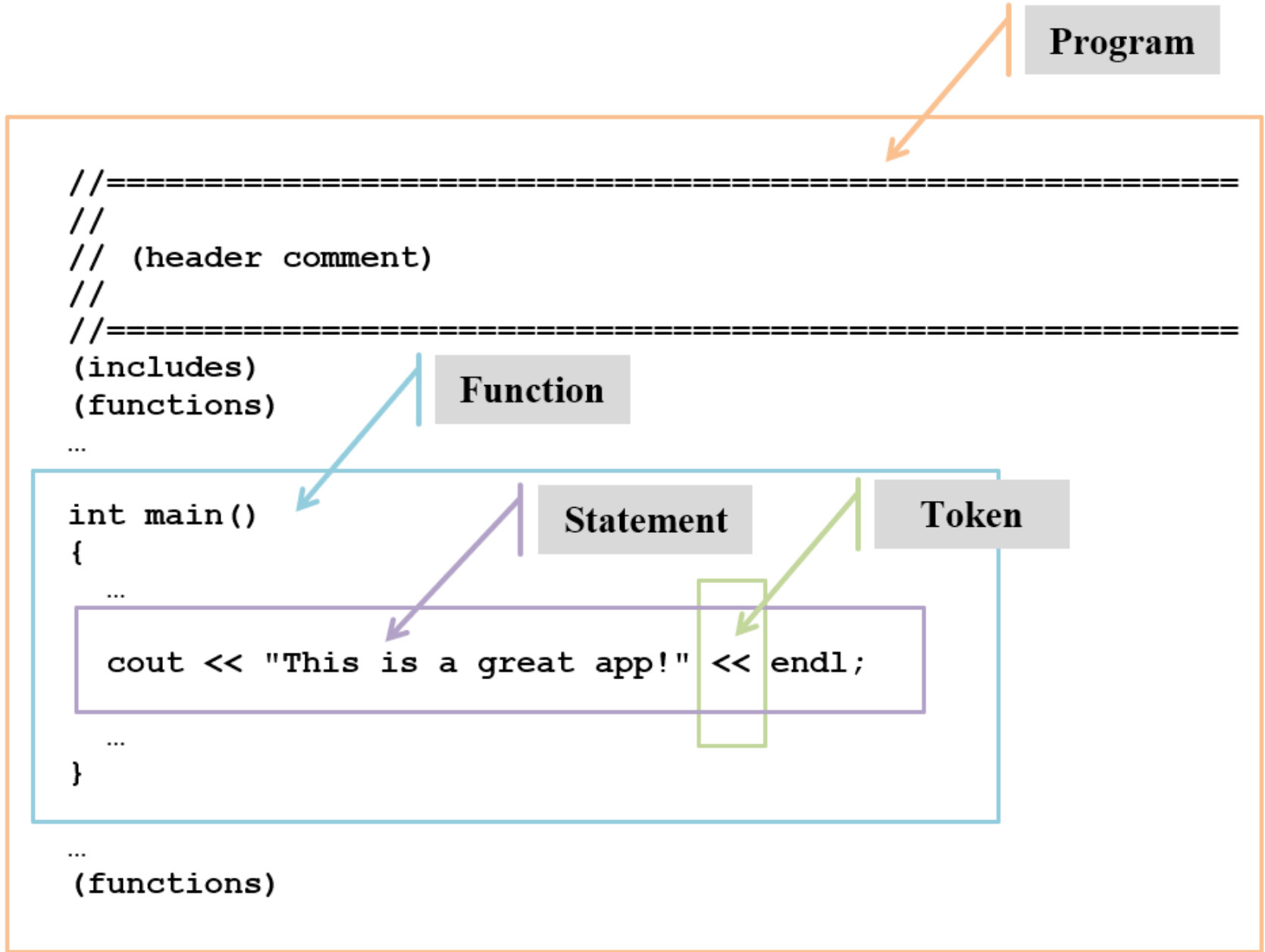
salesTax = stateSalesTax + citySalesTax + luxuryTax
amountDue = salePrice + salesTax

Programming Concepts

- A program is a sequence of computer instructions that:
  - Executes very fast.
  - Does exactly what the instructions say.
- A program is written in one or more programming languages.
- A statement is a single instruction that:
  - Consists of one or more lines.
  - Is usually terminated with a special character (; in C++).
- Syntax is the set of rules determining what a legal programming statement is.
- A token is:
  - The smallest sequence of characters defined by the syntax.
  - The smallest element of any programming language.

Programming Concepts

- A statement consists of one or more tokens.
- One or more spaces or the end-of-line typically separates one token from the next.
- A function is a named group of statements that perform a task.
- A program consists of one or more functions.
- A function consists of one or more statements.


Program
//==========
//
// (header comment)
//
//==========
(includes)
(functions)
...
Function
int main()
{
...
Statement
Token
cout << "This is a great app!" << endl;
...
}
...
(functions)

Comments

- A comment is a note that describes the corresponding code.
- Comments are for the reader, not for the compiler. So when a compiler compiles a program to check for the syntax errors, it completely ignores comments.
- Typically, comments can be used to identify the authors of the program, give the date when the program is written or modified, give a brief explanation of the program, and explain the meaning of key statements in a program.
- There are two common types of comments in a C++ program
  - Single-line comments
  - Multiple-line comments.

Comments

- Single-line comments begin with **//** and can be placed anywhere in the line. Everything encountered in that line after **//** is ignored by the compiler.

```
cout << "7 + 8 = " << 7 + 8 << endl; //prints: 7 + 8 = 15
```

- Multiple-line comments are enclosed between **/*** and ***/**. The compiler ignores anything that appears between **/*** and ***/**

```
/* You can include comments
that can occupy several lines. */
```

Whitespace

- Is a blank area in an application within or between lines of code.
- Involves indentation and spacing.
- Characters include spaces, tabs, and blank lines.
- Is critical to making a program more readable.
- May be used anywhere in a program.
- Although C++ does not require any whitespace between tokens, code is more readable when there is.