Lecture Notes on Jan/17/2023

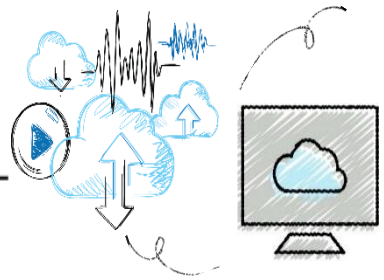
Chapter 1

An Overview of Computers and Programming Languages

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)



Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Current Position: Assistant Professor at Gannon University
- Previous Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

The Evolution of Programming Languages (1 of 3)

- Early computers were programmed in machine language
- To calculate wages = rate * hours in machine language:

```
100100 010001     //Load
100110 010010     //Multiply
100010 010011     //Store
```

The Evolution of Programming Languages (2 of 3)

- Assembly language instructions are mnemonic
  - Instructions are written in an easy-to-remember form
- An assembler translates a program written in assembly language into machine language
- Using assembly language instructions, **`wages = rate * hours`** can be written as:

```
LOAD rate
MULT hours
STOR wages
```

The Evolution of Programming Languages (3 of 3)

- High-level languages include Basic, FORTRAN, COBOL, C, C++, C#, Java, and Python
- Compiler: translates a program written in a high-level language into machine language
- In C++, the weekly wages equation can be written as:

```
wages = rate * hours;
```

CENGAGE Learning

Processing a C++ Program (1 of 4)

```
#include <iostream>
using namespace std;
int main()
{
    cout << "My first C++ program." << endl;
    return 0;
}
```

Sample Run:

```
My first C++ program.
```

<iostream>: Standard Input / Output Streams Library Header that defines the standard input/output stream objects.

using namespace std: All the files in the C++ standard library declare all of its entities within the std namespace. That is why we have generally included the using namespace std; statement in all programs that used any entity defined in iostream.

Processing a C++ Program (2 of 4)

- Steps needed to process a C++ program
  1. Use a text editor to create the source code (source program) in C++
  2. Include preprocessor directives
     - Begin with the symbol # and are processed by the preprocessor
  3. Use the compiler to:
     - Check that the program obeys the language rules
     - Translate the program into machine language (**object program**)
  4. Use an integrated development environment (IDE) to develop programs in a high-level language
     - Programs such as mathematical functions are available
     - The library contains prewritten code you can use
     - A linker combines object program with other programs in the library to create executable code
  5. The loader loads executable program into main memory
  6. The last step is to execute the program

CENGAGE Learning

Processing a C++ Program (3 of 4)

- IDEs are quite user friendly
  - Compiler identifies the syntax errors and also suggests how to correct them
  - Build or Rebuild is a simple command that links the object code with the resources used from the IDE

Processing a C++ Program (4 of 4)

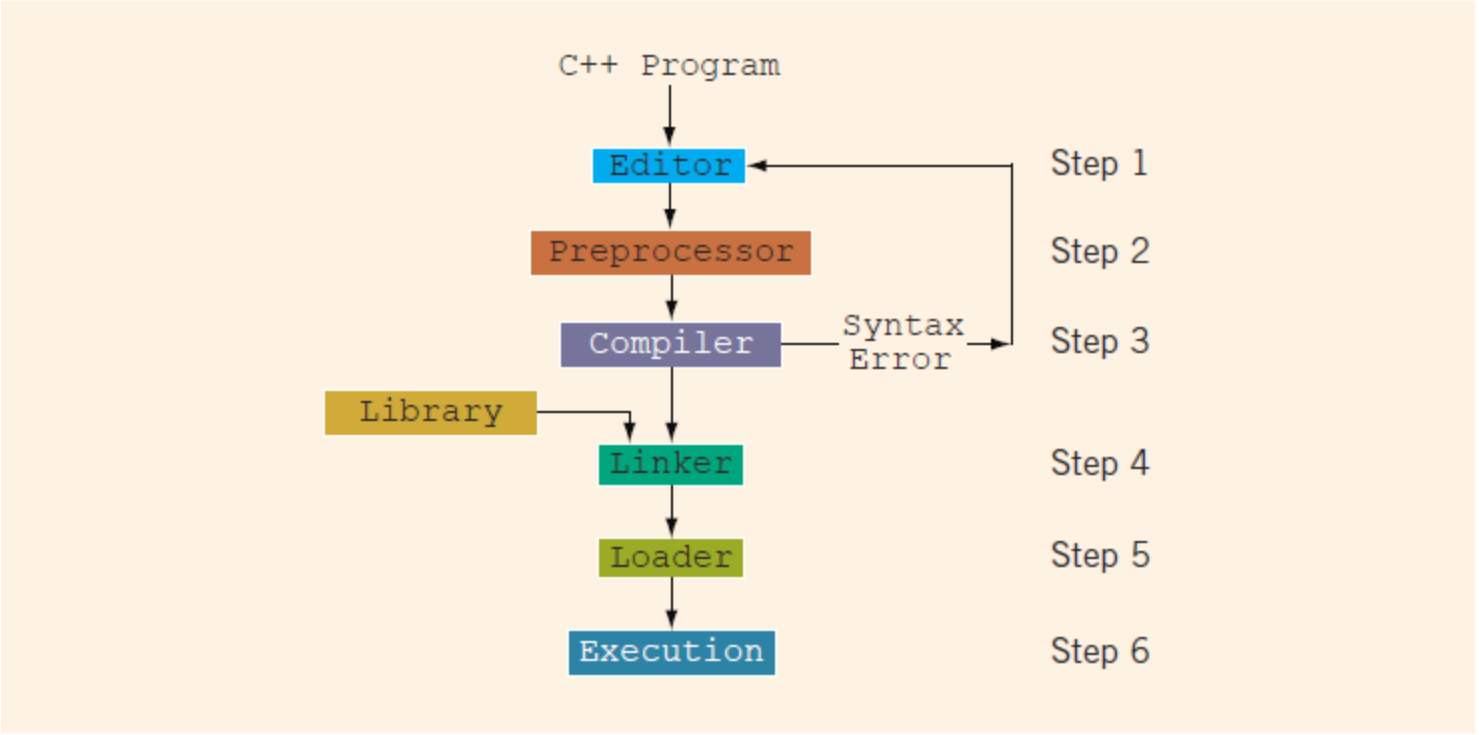


FIGURE 1-2 Processing a C++ program

Programming with the Problem Analysis–Coding–Execution Cycle

- Programming is a process of problem solving
- An algorithm is a step-by-step problem-solving process
  - A solution is achieved in a finite amount of time

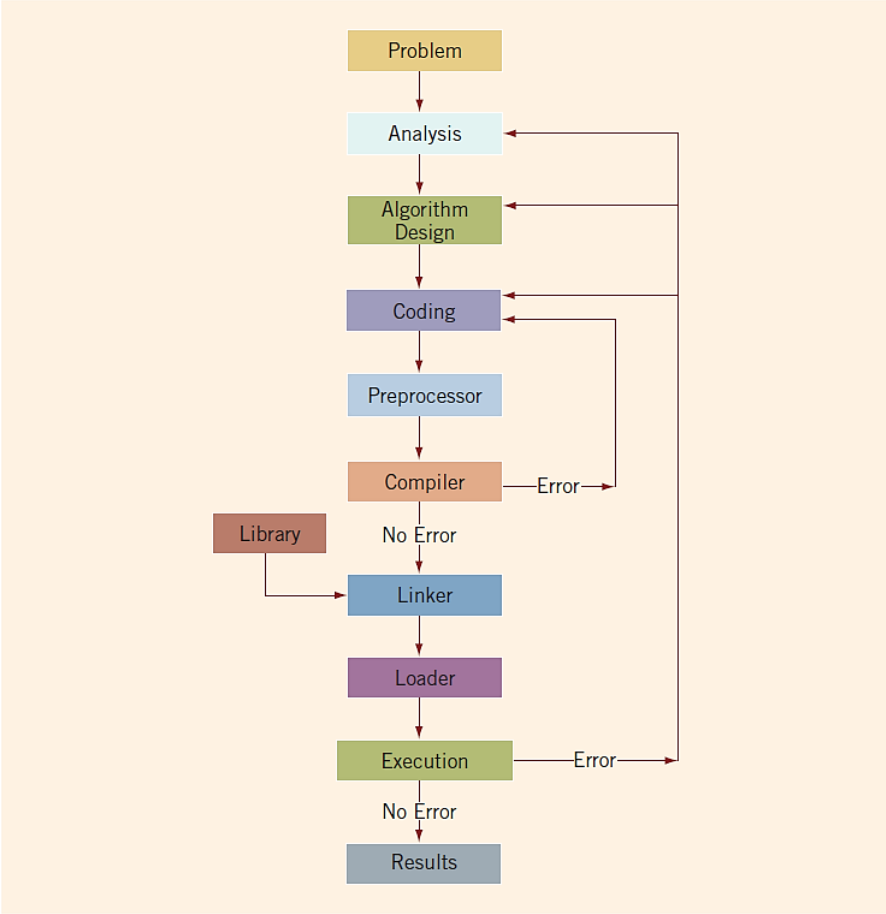


FIGURE 1-3 Problem analysis–coding–execution cycle

The Problem Analysis–Coding–Execution Cycle (1 of 5)

- Step 1: Analyze the problem
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem
- Step 2: Implement the algorithm
  - Implement the algorithm in code
  - Verify that the algorithm works
- Step 3: Maintain the program
  - Use and modify the program if the problem domain changes

The Problem Analysis–Coding–Execution Cycle (2 of 5)

- Analyze the problem using these steps:
  - Step 1: Thoroughly understand the problem and all requirements
  - Step 2: Understand the problem requirements
    - Does program require user interaction?
    - Does program manipulate data?
    - What is the output?
  - Step 3: If complex, divide the problem into subproblems
    - Analyze and design algorithms for each subproblem
- Check the correctness of algorithm
  - Test the algorithm using sample data
  - Some mathematical analysis might be required

The Problem Analysis–Coding–Execution Cycle (3 of 5)

- Once the algorithm is designed and correctness is verified
  - Write the equivalent code in high-level language
- Enter the program using a text editor

The Problem Analysis–Coding–Execution Cycle (4 of 5)

- Run code through the compiler
- If compiler generates errors
  - Look at code and remove errors
  - Run code again through compiler
- If there are no syntax errors
  - Compiler generates equivalent machine code
- Link machine code with the system's resources
  - Performed by the linker

The Problem Analysis–Coding–Execution Cycle (5 of 5)

- Once compiled and linked, the loader can place program into main memory for execution
- The final step is to execute the program
- Compiler guarantees that the program follows the rules of the language
  - Does not guarantee that the program will run correctly

CENGAGE Learning

Programming Methodologies

- Two popular approaches to programming design
  - Structured
  - Object-oriented

Structured Programming

- Structured design
  - Involves dividing a problem into smaller subproblems
- Structured programming
  - Involves implementing a structured design
- The structured design approach is also called:
  - Top-down (or bottom-up) design
  - Stepwise refinement
  - Modular programming

CENGAGE Learning

Object-Oriented Programming (1 of 3)

- Object-oriented design (OOD)
  - Identify components called objects
  - Determine how objects interact with each other
- Specify relevant data and possible operations to be performed on that data
- Each object consists of data and operations on that data

Object-Oriented Programming (2 of 3)

- An object combines data and operations on the data into a single unit
- A programming language that implements OOD is called an object-oriented programming (OOP) language
- To design and use objects, you must learn how to:
  - Represent data in computer memory
  - Manipulate data
  - Implement operations

Object-Oriented Programming (3 of 3)

- To create operations:
  - Write algorithms and implement them in a programming language
  - Use functions to implement algorithms
- Learn how to combine data and operations on the data into a single unit called a class
- C++ was designed to implement OOD
- OOD is used with structured design

A Quick Look at a C++ Program (1 of 5)

EXAMPLE 2-1

```
//*************************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//*************************************************************

#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;
    width = 4.0;
    perimeter = 2 * (length + width);
    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

CENGAGE Learning

A Quick Look at a C++ Program (2 of 5)

- Sample Run:

```
Program to compute and output the perimeter and area of a rectangle.
Length = 6
Width = 4
Perimeter = 20
Area = 24
```

A Quick Look at a C++ Program (3 of 5)

```
//************************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//************************************************************

#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;
```

Comments

Variable declarations. A statement such as `double length;` instructs the system to allocate memory space and name it **length**.

Assignment statement. This statement instructs the system to store **6.0** in the memory space **length**.

FIGURE 2-1 Various parts of a C++ program

A Quick Look at a C++ Program (4 of 5)

```
    width = 4.0;
    perimeter = 2 * (length + width);

    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

Assignment statement. This statement instructs the system to evaluate the expression **length * width** and store the result in the memory space **area**.

Output statements. An output statement instructs the system to display results.

FIGURE 2-1 Various parts of a C++ program (cont'd.)

A Quick Look at a C++ Program (5 of 5)

- Variable: a memory location whose contents can be changed

FIGURE 2-3 Memory allocation

FIGURE 2-4 Memory spaces after the statement `length = 6.0;` executes