CS118 – Programming Fundamentals

Lecture # 06
Monday, September 02, 2019
FALL 2019
FAST – NUCES, Faisalabad Campus

Zain Iqbal

Basic Components of C++ Program

The Evolution of Programming Languages

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

```
100100 010001    //Load rates
100110 010010    //Multiply
100010 010011    //Store in wages
```

The Evolution of Programming Languages (cont'd.)

- Assembly language instructions are mnemonic
- **Assembler:** Translates a program written in assembly language into machine language

TABLE 1-2 Examples of Instructions in Assembly Language and Machine Language

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

The Evolution of Programming Languages (cont'd.)

- Using assembly language instructions, wages = rates * hours can be written as:

```
LOAD  rate
MULT  hour
STOR  wages
```

The Evolution of Programming Languages (cont'd.)

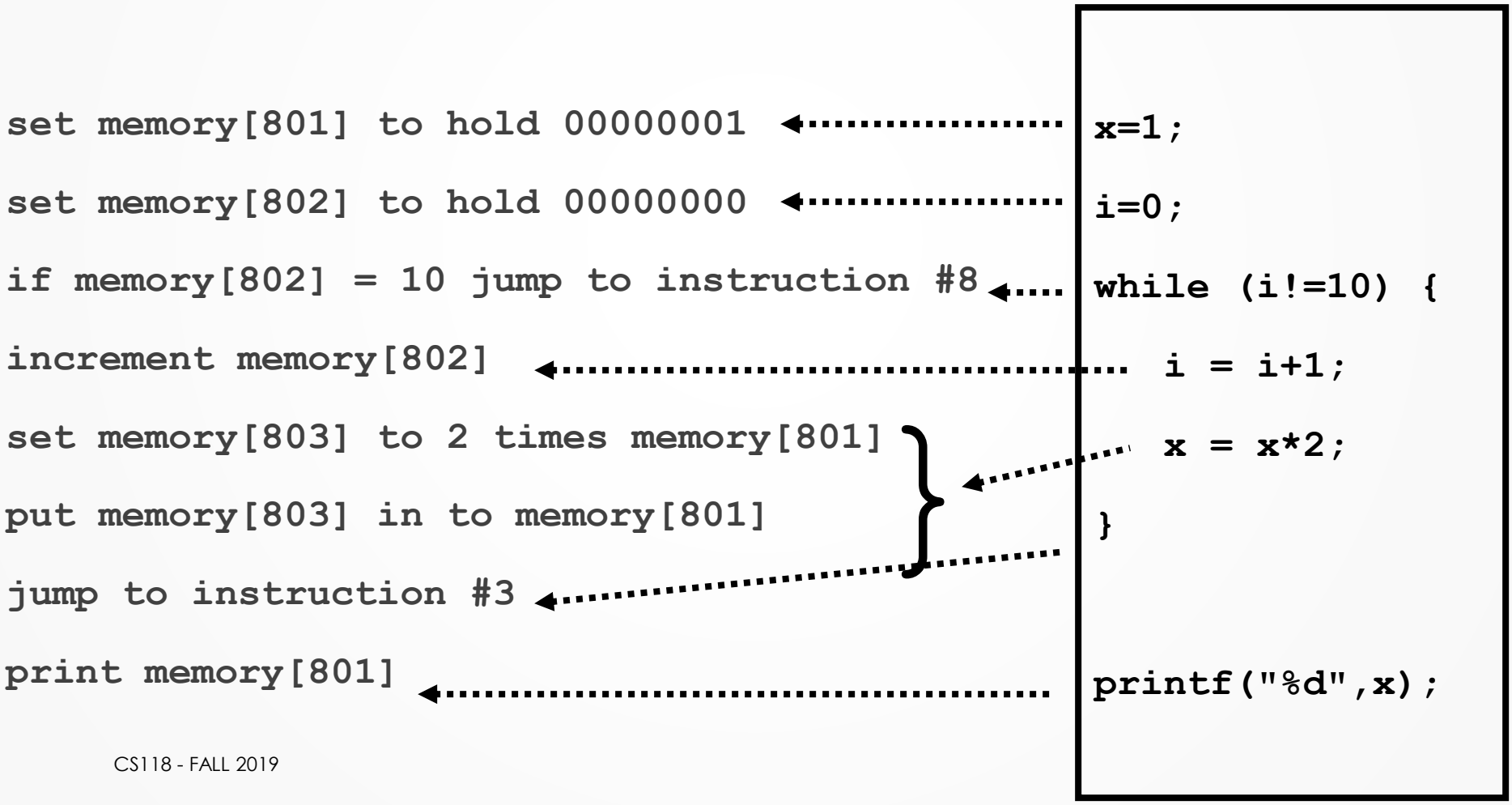
- High-level languages include Basic, FORTRAN, COBOL, Pascal, C, C++, C#, and Java
- **Compiler:** Translates a program written in a high-level language to machine language
- The equation wages = rate • hours can be written in C++ as:

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

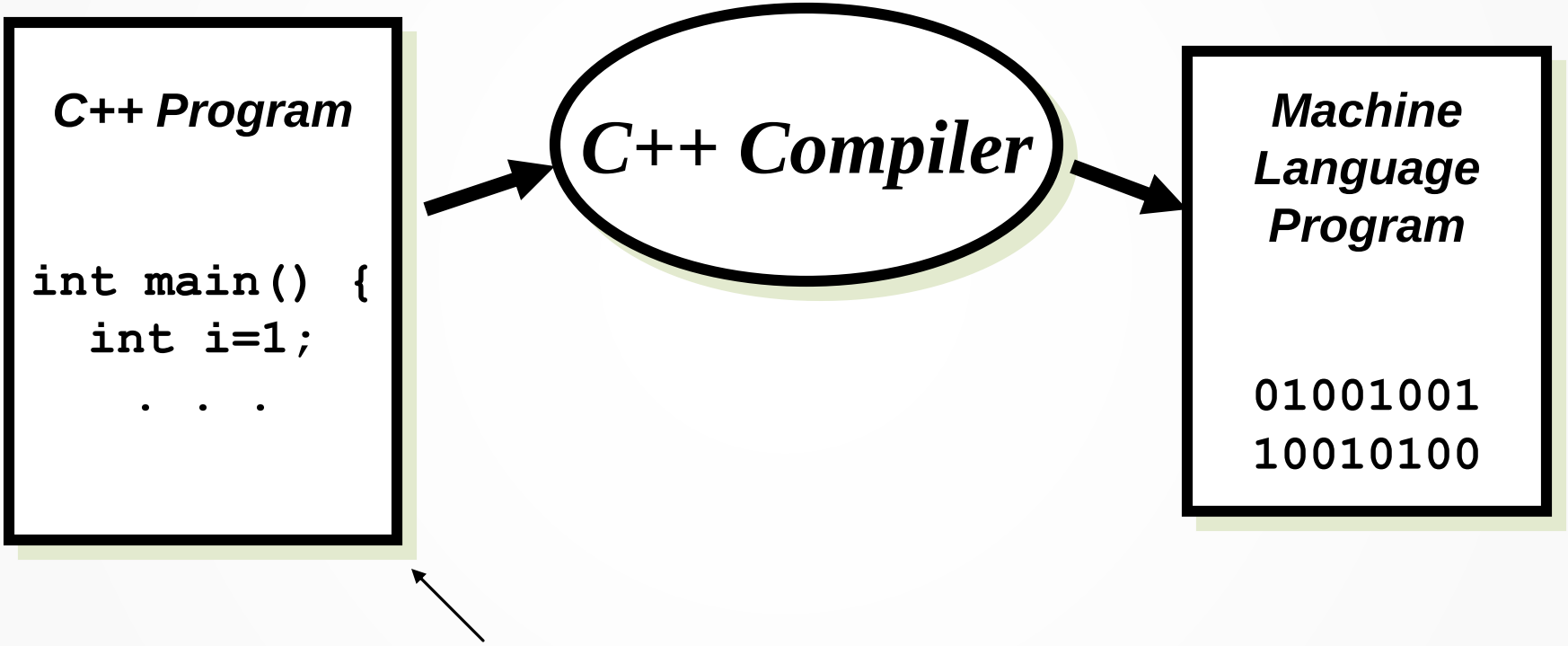
Assembly & Machine Language

Assembly Language	Machine Language
`ST 1,[801]`	`00100101 11010011`
`ST 0,[802]`	`00100100 11010100`
`TOP: BEQ [802],10,BOT`	`10001010 01001001 11110000`
`INCR [802]`	`01000100 01010100`
`MUL [801],2,[803]`	`01001000 10100111 10100011`
`ST [803],[801]`	`11100101 10101011 00000010 00101001`
`JMP TOP`	`11010101`
`BOT: LD A,[801]`	`11010100 10101000`
`CALL PRINT`	`10010001 01000100`

Equivalent C/C++ program



Compiler


C++ Program
int main() {
int i=1;
. . .
C++ Compiler
Machine Language Program
01001001
10010100
Created with text editor or development environment

Processing a C++ Program

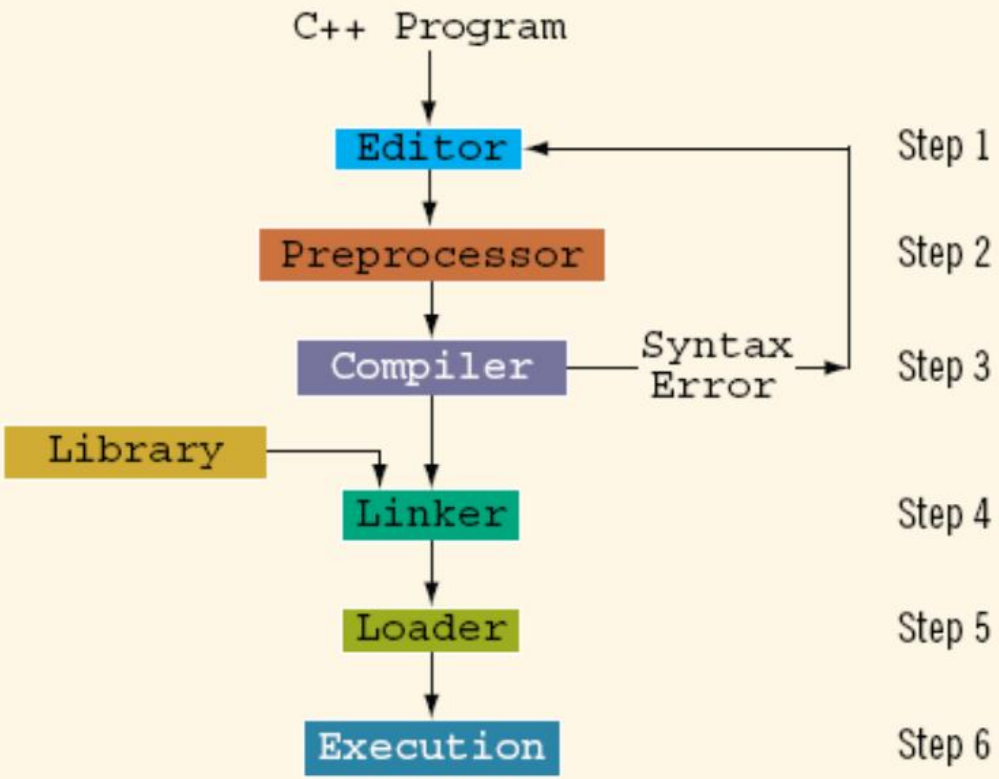
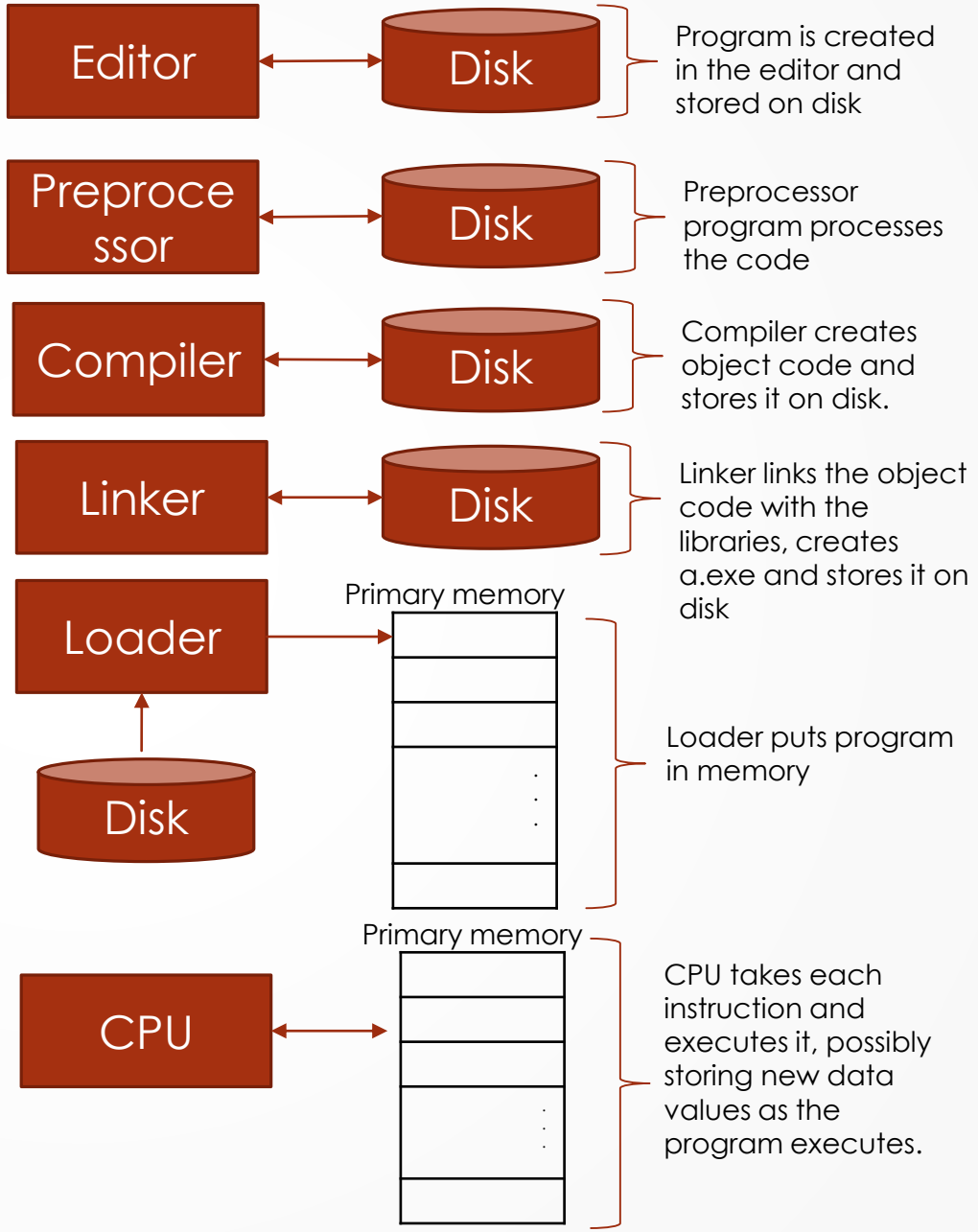


FIGURE 1-3 Processing a C++ program

Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Compilers

- Translate high-level language to machine language
- Check that the program obeys the rules

- **Source code**
  - The original program in a high level language
- **Object code**
  - The translated version in machine language

Linkers

- Some programs we use are already compiled
  - Their object code is available for us to use
  - **For example:** Input and output routines
- A **Linker** combines
  - The object code for the programs we write

    and
  - The object code for the pre-compiled routines (**of SDK**)

    into
  - The machine language program the CPU can run
- **Loader:**
  - Loads executable program into main memory
- The last step is to execute the program

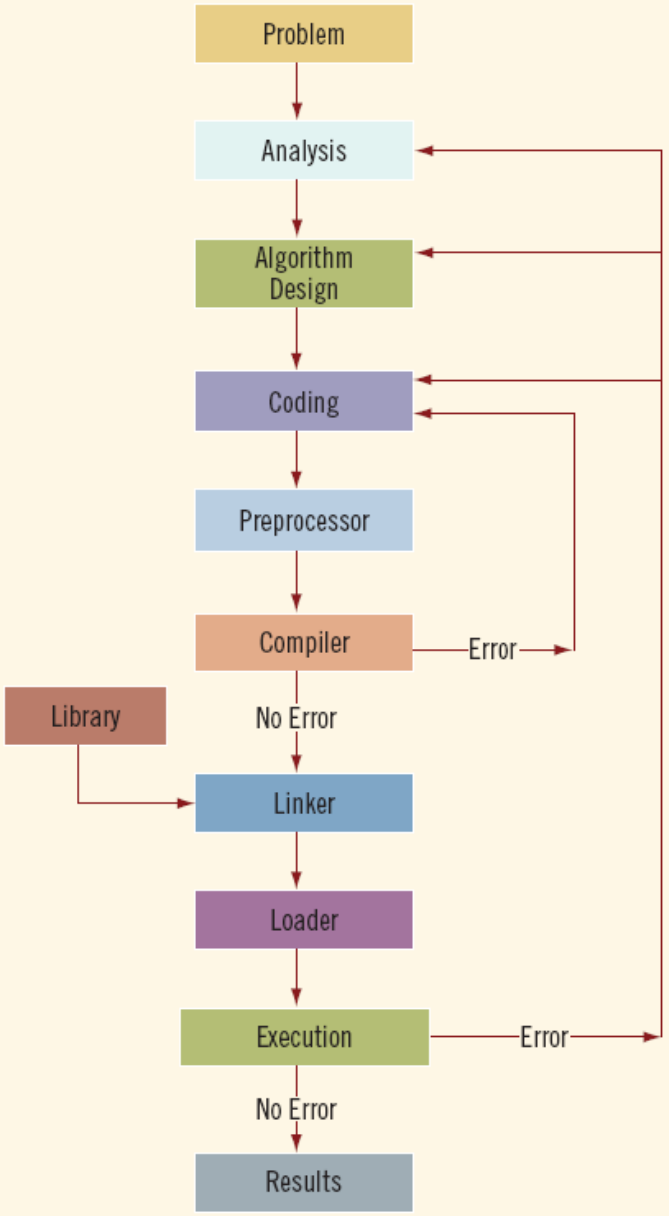


FIGURE 1-4 Problem analysis–coding–execution cycle

History of C and C++

- **History of C**
  - Evolved from two other programming languages
  - BCPL and B: "Typeless" languages
  - **Dennis Ritchie (Bell Lab):** Added typing, other features
- C is a programming language developed in the 1970's alongside the UNIX operating system
- C provides a comprehensive set of features for handling a wide variety of applications, such as systems development and scientific computation
  - 1989: ANSI standard/ ANSI/ISO 9899: 1990
- **History of C++**
  - Early 1980s: **Bjarne Stroustrup** (Bell Lab)
  - Provides capabilities for object-oriented programming
    - **Objects:** reusable software components
    - Object-oriented programs
- **Building block** approach" to creating programs
  - C++ programs are built from pieces called classes and functions
  - **C++ standard library:** Rich collections of existing classes and functions

Structured/OO Programming

- **Structured programming (1960s)**
  - Disciplined approach to writing programs
  - Clear, easy to test and debug, and easy to modify
  - e.g. Pascal: 1971: Niklaus Wirth
- **OOP**
  - "Software reuse"
  - "Modularity"
  - "Extensible"
  - More understandable, better organized and easier to maintain than procedural programming

Basics of a Typical C++ Environment

- **C++ systems**
  - Program-development environment
    - Integrated Development Environment (IDE)
  - Language
  - C++ Standard Library
- **C++ program names extensions**
  - .cpp (C Plus Plus)
  - .c (C)

The C++ Standard Library

C/C++ programs consist of pieces/modules called functions

- A programmer can create his own functions
  - **Advantage:** the programmer knows exactly how it works
  - **Disadvantage:** time consuming
- Programmers will often use the C/C++ library functions
  - Use these as building blocks
- Avoid re-inventing the wheel
  - If a pre-made function exists, generally best to use it rather than write your own
  - Library functions carefully written, efficient, and portable

Programming Style

C++ is a free-format language, which means that:

- Extra blanks (spaces) or tabs before or after identifiers/operators are ignored
- Blank lines are ignored by the compiler just like comments
- Code can be indented in any way
- There can be more than one statement on a single line
- A single statement can continue over several lines

Programming Style (cont.)

In order to improve the readability of your program, use the following conventions:

- Start the program with a **header** that tells what the program does
- Use **meaningful variable names and Camel notation**
- **Document** each variable declaration with a comment telling what the variable is used for
- Place each **executable statement** on a **single line**
- A segment of code is a sequence of executable statements that belong together
  - Use blank lines to separate different segments of code
  - Document each segment of code with a comment telling what the segment does.

C++ keywords

- Keywords appear in **blue** in Visual C++
- Each keyword has a predefined purpose in the language
- Do not use keywords as variable and constant names!!

- We shall cover most of the following keywords in this class:

bool, break, case, char, const, continue, do, default, double, else, extern, false, float, for, if, int, long, namespace, return, short, static, struct, switch, typedef, true, unsigned, void, while

Structure of a C++ Program

A C++ program is a collection of definitions and declarations:

- data type definitions
- global data declarations
- function definitions (subroutines)
- class definitions
- a special function called
  - **main()** (where the action starts)

General form of a C++ program


```
// Program description
#include directives
int main()
{
      constant declarations
      variable declarations
      executable statements
      return 0;
}
```

Includes

- The statement: **#include <myfile.h>** inserts the contents of the file **myfile.h** inside your file before the compiler starts
- Definitions that allow your program to use the functions and classes that make up the standard C++ library are in these files.
- You can include your own file(s):

```
#include "myfile.h"
```

C++ compiler directives

- Compiler directives appear in **blue** in Visual C++
- The **#include** directive tells the compiler to include some already existing C++ code in your program
- The included file is then linked with the program
- There are two forms of #include statements:

```
#include <iostream> //for pre-defined files
#include "my_lib.h"   //for user-defined files
```

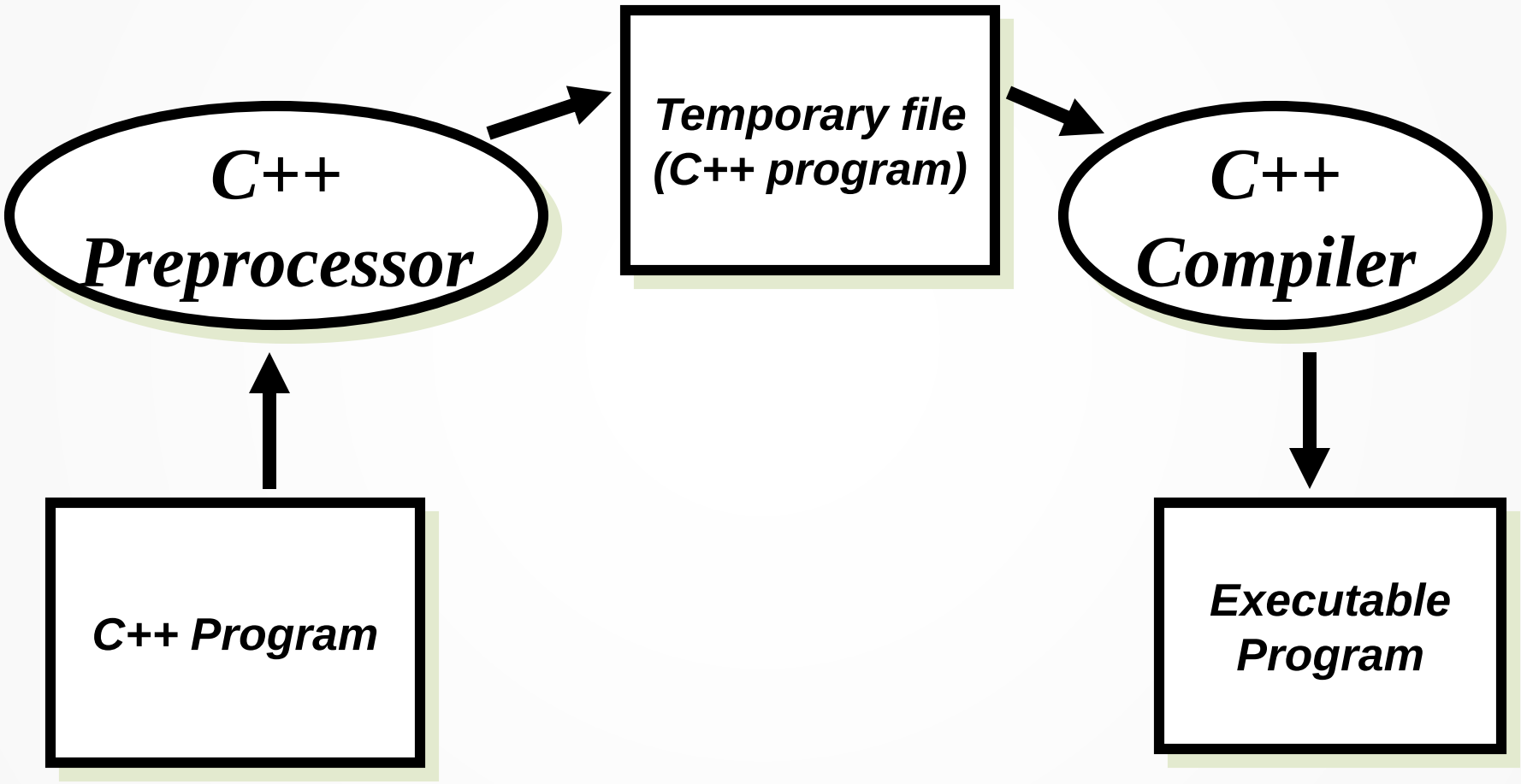
The Preprocessor

- Lines that start with the character **'#'** are special instructions to a preprocessor
- The preprocessor can replace the line with something else:
  - **include:** replaced with contents of a file
- Other *directives* tell the preprocessor to look for patterns in the program and do some fancy processing

C++ Preprocessor

- C++ Compilers **automatically invoke a preprocessor** that takes care of #include statements and some other special directives
- Definitions that **allow your program** to use the functions and classes that make up the standard C++ library are in these files
- You don't need to do anything special to run the preprocessor - it happens automatically

Preprocessing



Preprocessor Directives

- Preprocessor directives: Begin with #
- Processed before compiling
- # include
- # define

Some common include statements

- Basic I/O: iostream.h
  - Provides functionality of input and output
- I/O manipulation: iomanip.h
  - Format's the input and output
- Standard Library: stdlib.h
  - Functions for memory allocation, process control, conversion etc.
- Time and Date support: time.h
  - Functionality of time manipulation
- Mathematics support: math.h
  - Functionality of basic mathematical functions

Basics of a Typical C++ Program Environment

Common Input/output functions

- cin
  - Standard input stream
  - Normally keyboard
- cout
  - Standard output stream
  - Normally computer screen
- cerr
  - Standard error stream
  - Display error messages

I/O Streams and Standard I/O Devices

- **I/O:** sequence of bytes (stream of bytes) from source to destination
  - Bytes are usually characters, unless program requires other types of information
- **Stream:** Sequence of characters from source to destination
- **Input stream:** Sequence of characters from an input device to the computer
- **Output stream:** Sequence of characters from the computer to an output device

I/O Streams and Standard I/O Devices (cont'd.)

- Use iostream header file to extract (receive) data from keyboard and send output to the screen
- Contains definitions of two data types:
  - istream: input stream
  - ostream: output stream
- Has two variables:
  - cin: stands for common input
  - cout: stands for common output

I/O Streams and Standard I/O Devices (cont'd.)

- To use **cin** and **cout**, the preprocessor directive #include <iostream> must be used
- Variable declaration is similar to:
  - istream cin;
  - ostream cout;
- **Input stream variables:** type istream
- **Output stream variables:** type ostream

Basics of a Typical C++ Program Environment

- Insertion operator & extraction
- Input stream object
  - >> (stream extraction operator)
    - Used with std::cin
    - Waits for user to input value, then press Enter (Return) key
    - Stores value in variable to right of operator
    - Converts value to variable data type
- Use **using namespace std;** to reduce typing work

```
cin >> variable >> variable ...;
```

Basics of a Typical C++ Program Environment …

- Standard output stream object
  - std::cout
  - "Connected" to screen
  - <<
    - Stream insertion operator
    - Value to right (right operand) inserted into output stream
- **Namespace**
  - **std::** specifies that entity belongs to "namespace" **using binary scope resolution operator(::)**
  - **std::** removed through use of **using** statements
- **Escape characters: **
  - Indicates **"special"** character output

Questions