Computer Programming

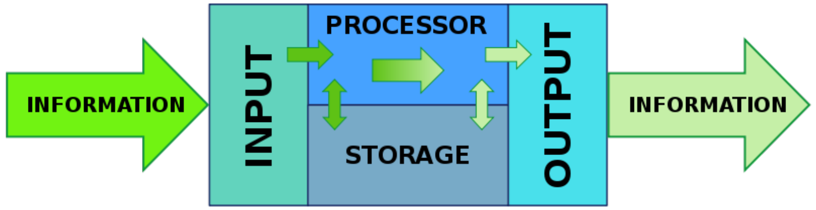
Introduction

Hung-Yun Hsieh

September 6, 2022

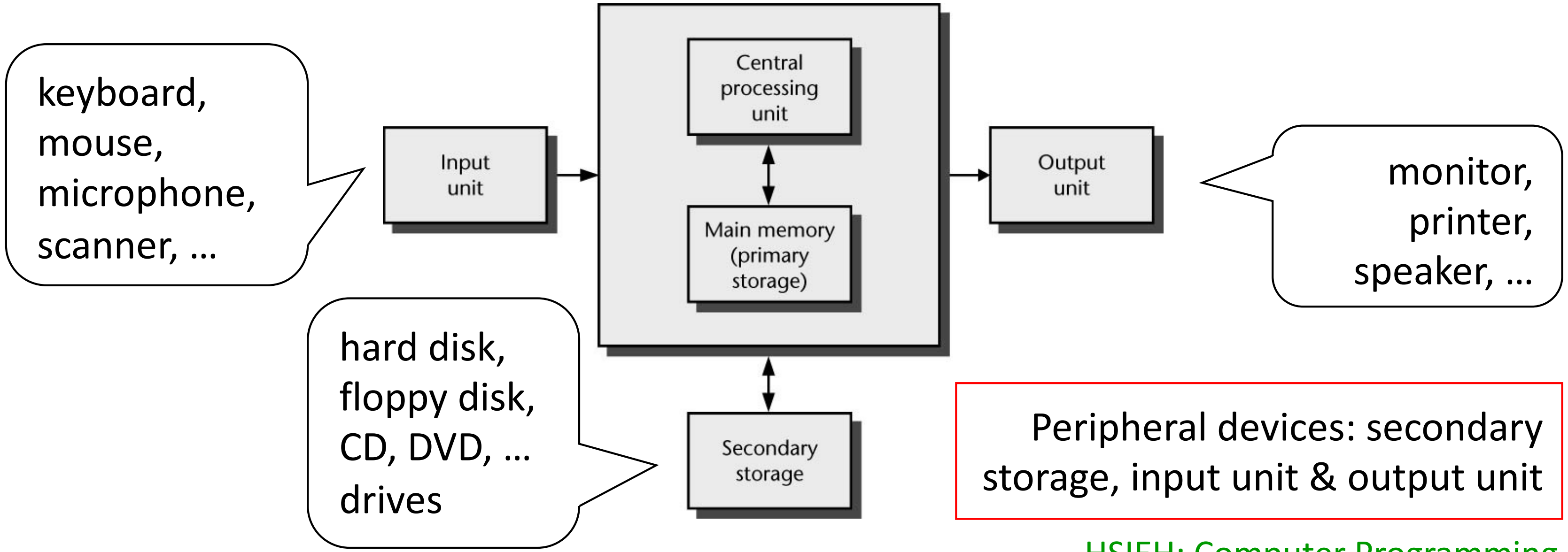
Abstraction of a Computer

- Computer
  - Device capable of performing computations and making logical decisions
  - Essentially everything that a computer does is related to information processing
  - ☞ Programmable to handle different tasks
- Capability of computers comes from…
  - Hardware
    - Physical devices of a computer that determine *what computers can do*
  - Software
    - Programs that run on computers to tell them *what to do*



① Hardware

- Information processing view
  - Information comes into the computer via the input unit
  - ☞ Information is stored in the memory
  - ☞ CPU reads instructions from memory to process information
  - Processed information is materialized via the output unit


keyboard, mouse, microphone, scanner, …
Input unit
Central processing unit
Main memory (primary storage)
Output unit
monitor, printer, speaker, …
hard disk, floppy disk, CD, DVD, … drives
Secondary storage

Peripheral devices: secondary storage, input unit & output unit

② Software

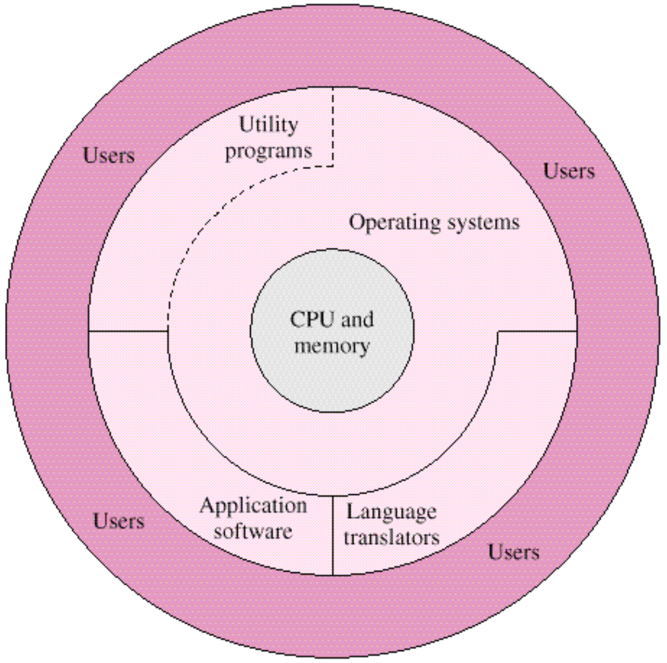
All computer programs (excluding firmware) require an OS to function

- Two groups
  - System software
    - Includes operating systems, system utility software and system development (language translation) software

resource management, *program loading & execution*, multi-tasking, disk access, …

    1. MS Windows, Unix, BSD, Linux, Android, macOS, iOS, …
    2. dir, copy, ls, mkdir, …
    3. C, C++, …
  - Application software
    - MS office, Photoshop, games, …
    - IE, Safari, Chrome, …
    - MATLAB, PSPICE, Cadence, …

☞ Computer programs of your own design

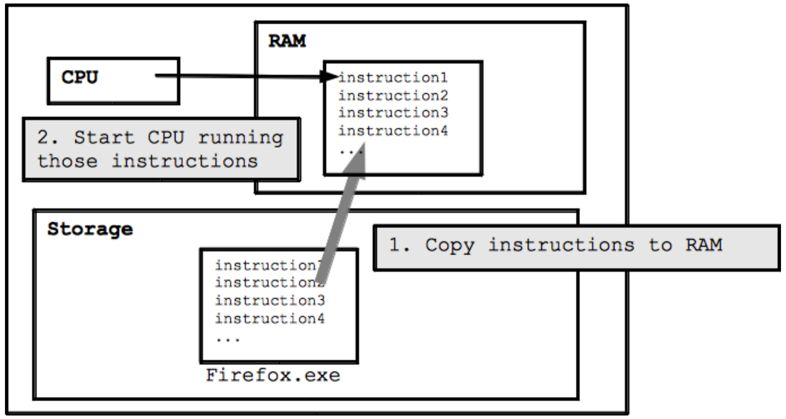


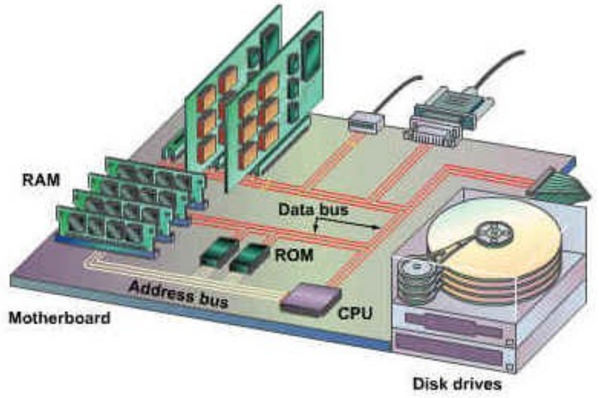
Computer Programming

Programming Language

Programming Language

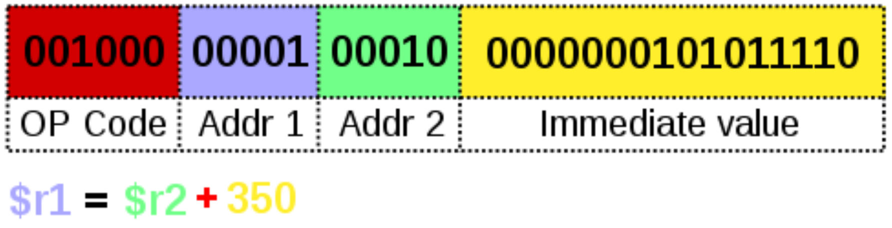
- Programming language
  - A programming language is a special language used to write computer programs
  - Programming languages have strict rules to prevent *translation errors* that could arise due to ambiguous interpretations
  - A computer program is stored in the memory in the form of *CPU instructions* to be executed by the CPU

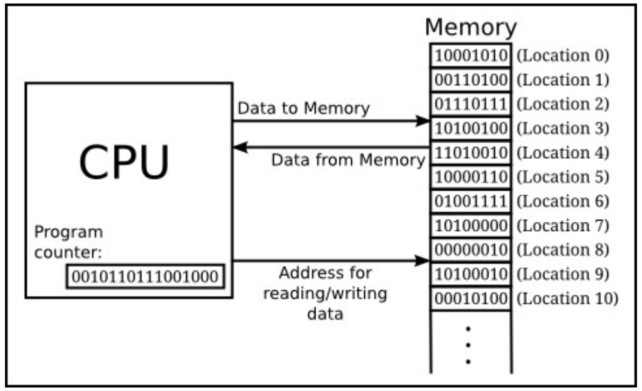




CPU Instructions

- Instruction set
  - ☞ The set of instructions that a CPU understands ("natural" language for a computer)
  - ☞ Machine code (binary code)
  - Most instructions have one or more opcode (to select the operation to perform) fields and other fields that may contain the operand(s)
  - Machine (processor) dependent: each processor has its own set of machine instructions





① Low-Level Programming Language

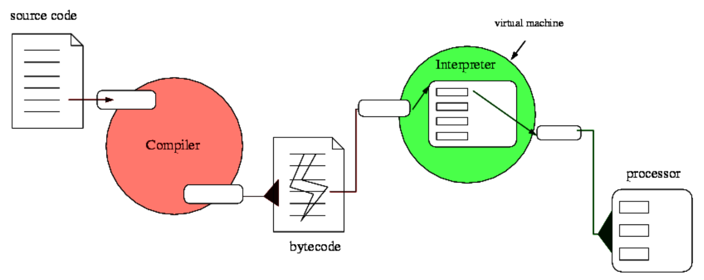
- Assembly language
  - English-like abbreviations representing elementary computer operations (CPU instructions)
    - ADD, LOAD, STORE, …
    - Assigns short names to instructions
  - Make reading "easier" to humans
  - Need to use the assembler to translate to CPU instructions

Machine code	Assembly code	Description
001 1 000010	LOAD #2	Load the value 2 into the Accumulator
010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
001 1 000101	LOAD #5	Load the value 5 into the Accumulator
010 0 001110	STORE 14	Store the value of the Accumulator in memory location 14
001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator
011 0 001110	ADD 14	Add the value of memory location 14 to the Accumulator
010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15
111 0 000000	HALT	Stop execution

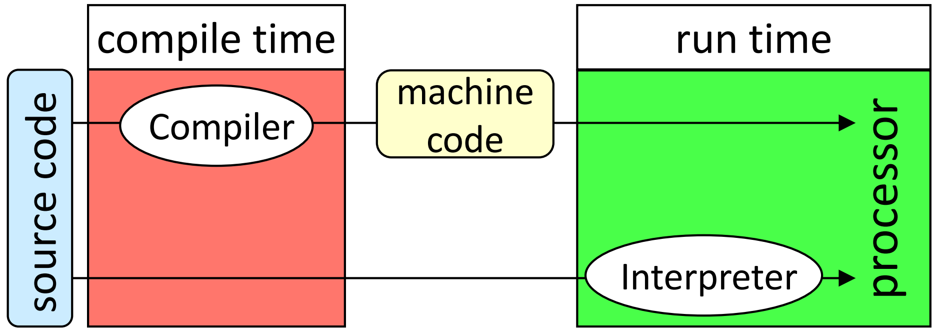
② High-Level Programming Language

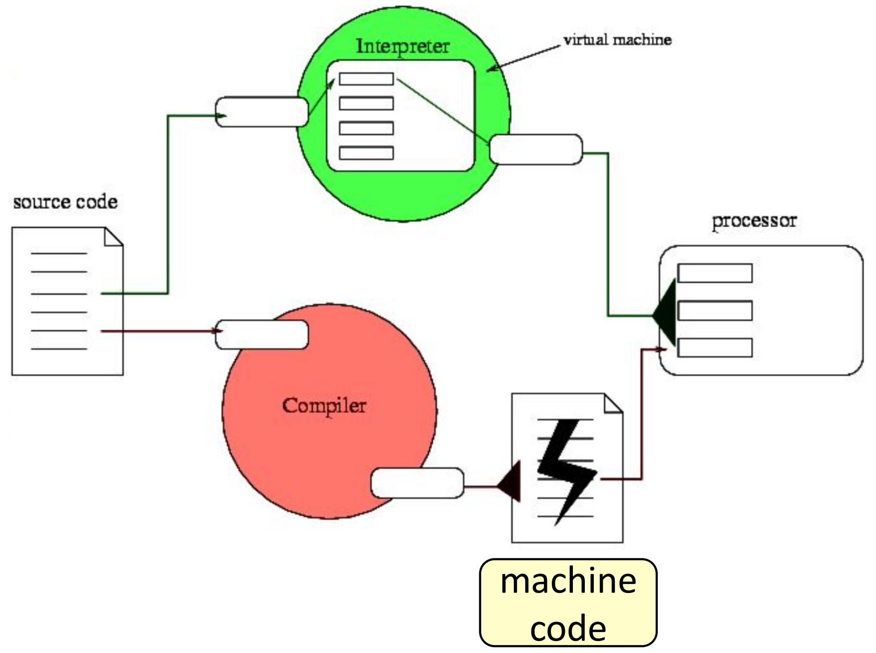
- High-level language
  - Similar to everyday English while using common mathematical notations
  - A single statement can accomplish complicated tasks performed by multiple CPU instructions
  - Many programming languages are created with specific purposes
    - Database processing, text processing, artificial intelligence, math operations
  - Translator
    - Compiler – convert to machine code before execution
    - Interpreter – directly execute high-level language programs

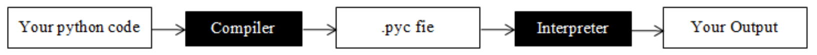
Compiler vs. Interpreter

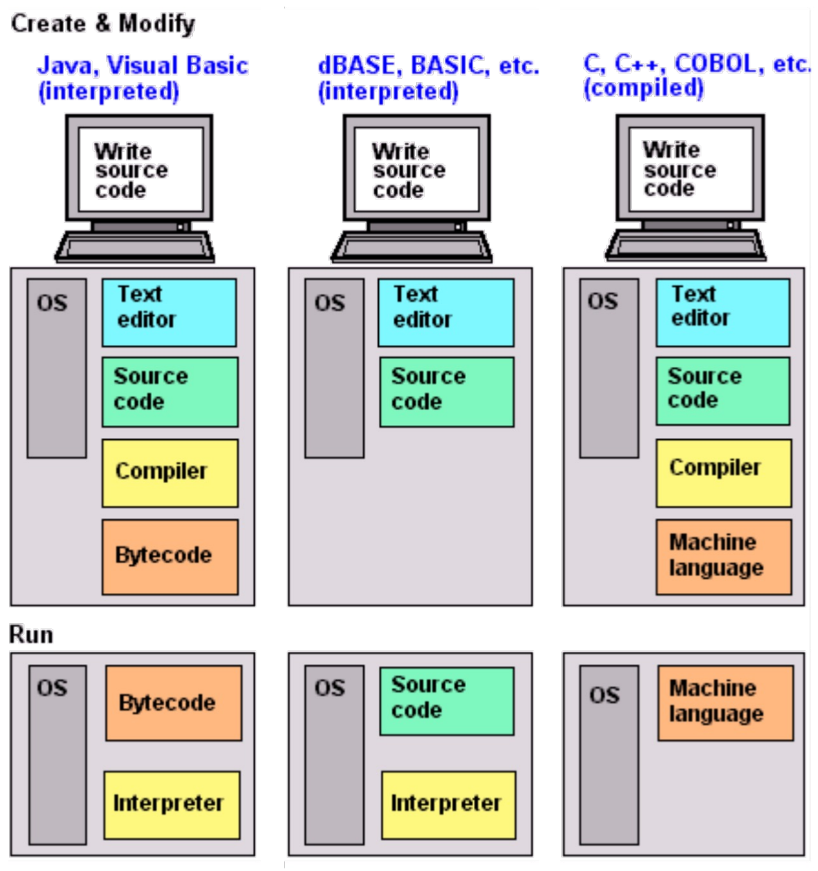


- Compile-time vs. run-time behavior





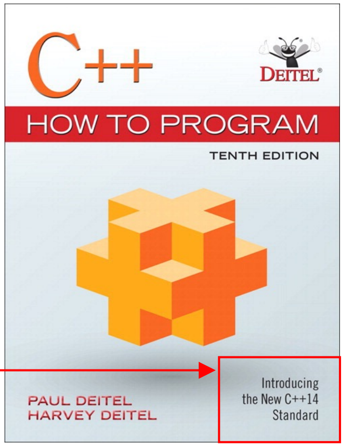




C and C++

The C language has evolved from C90, C99, C11 (C1X), …

- C and C++ are languages that grew in increments
  - 1972 – C was created at the Bell Labs and evolved from two other languages: BCPL and B for writing OS (Unix)
  - 1985 – C with classes (C++) was officially released at the Bell Labs for object-oriented features
  - 1990 – ANSI standard of C
  - 1998 – ANSI standard of C++ (aka C++98)
  - Evolving standards: C++11, C++14, …
- C++ is a hybrid language
  - C++ allows programmers to use new features without throwing away old C code
  - C-like style and object-oriented style can co-exist



Elements of a Programming Language

- What constitutes a programming language?
  - Keyword
  - Identifier
  - Operator
  - Punctuation mark
  - Syntax

Token (Keyword, Identifier, Operator, Punctuation mark)

Language Element	Description
Keywords	Words that have a special meaning. Keywords may only be used for their intended purpose.
Identifiers	Words or names (identifiers) defined by the programmer. They are symbolic names that refer to variables or programming routines.
Operators	Operators perform operations on one or more operands. An operand is usually a piece of data, like a number.
Punctuation Marks	Punctuation characters that mark the beginning or ending of a statement, or separate items in a list.
Syntax	Rules that must be followed when constructing a program. Syntax dictates how keywords and operators may be used, and where punctuation symbols must appear.

Tokens are atomic items of a language -- each significant *lexical chunk of the program* is represented by a token

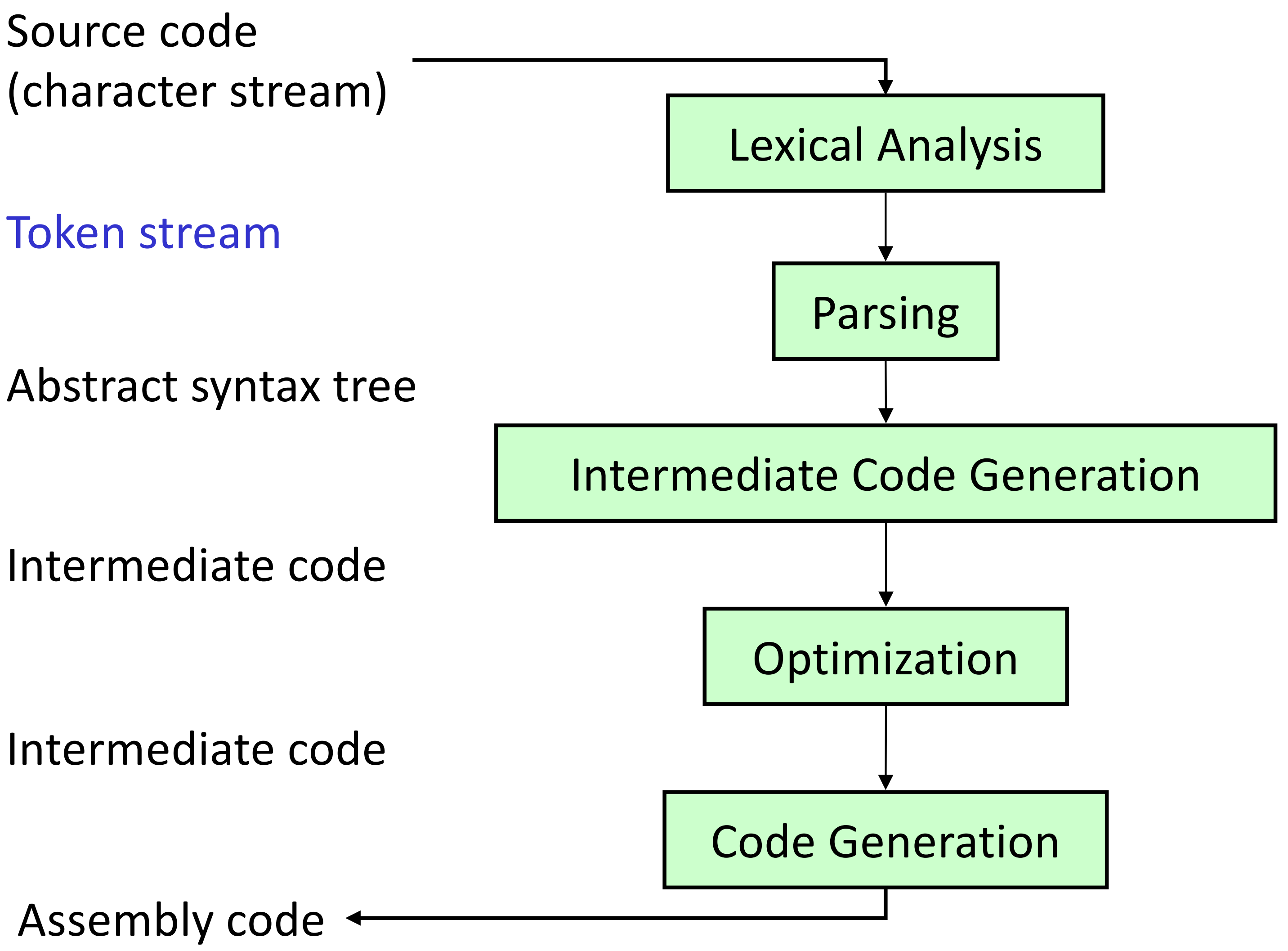
C++ (Source) Code

```
#include <iostream> // header file for std::cin
/*
The program entry
*/
int main( )
{
    float a, b;
    std::cin >> b;
    if (b==0) a=b;
    else      a=1/b;
    std::cout << "a is " << a;
    return 0;
}
```

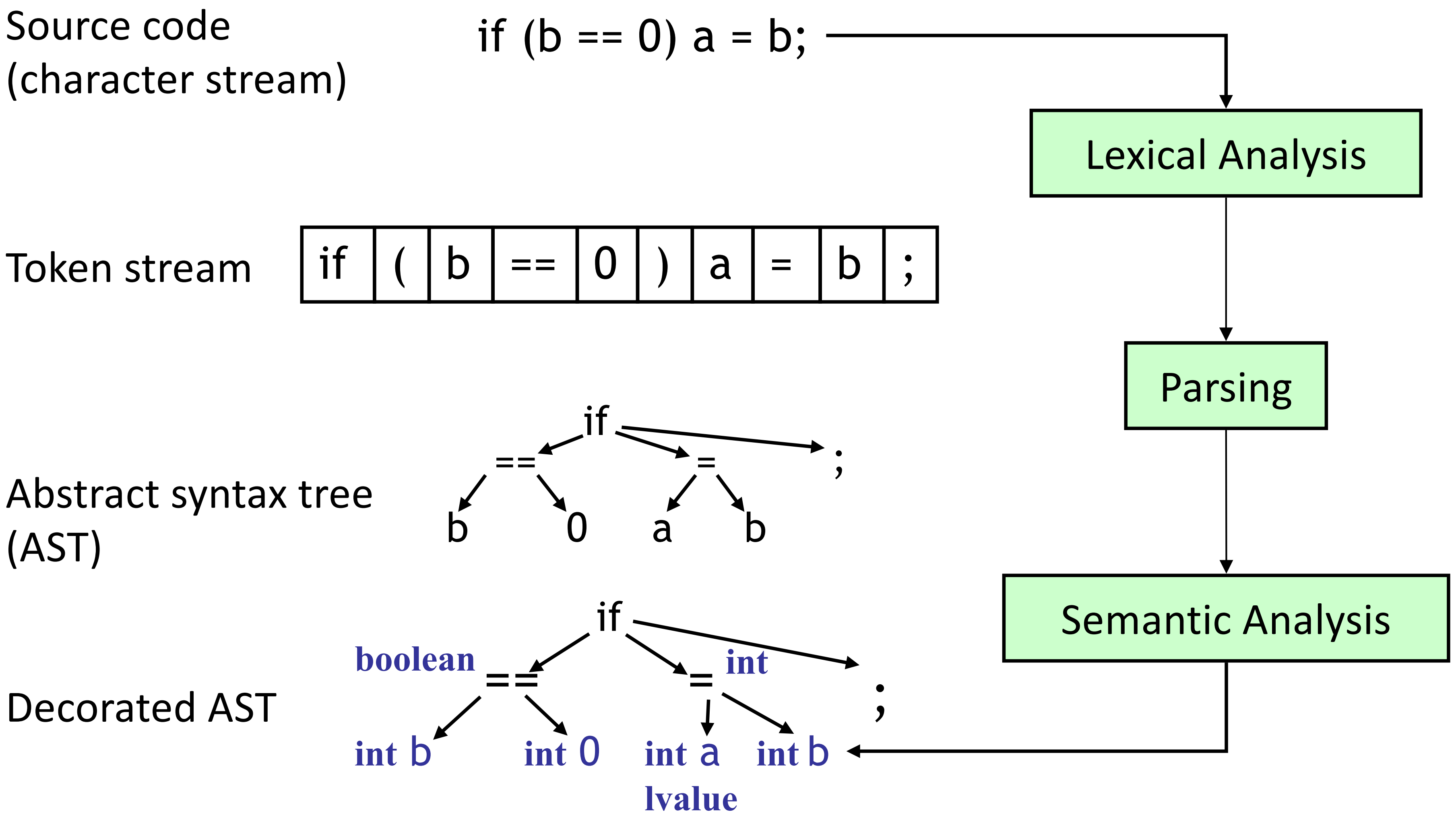
Language Element	**Description**
Keywords	Words that have a special meaning. Keywords may only be used for their intended purpose.
Identifiers	Words or names (identifiers) defined by the programmer. They are symbolic names that refer to variables or programming routines.
Operators	Operators perform operations on one or more operands. An operand is usually a piece of data, like a number.
Punctuation Marks	Punctuation characters that mark the beginning or ending of a statement, or separate items in a list.
Syntax	Rules that must be followed when constructing a program. Syntax dictates how keywords and operators may be used, and where punctuation symbols must appear.

☞ Compiling the *human-readable* source code to *machine-readable* object code (binary code)

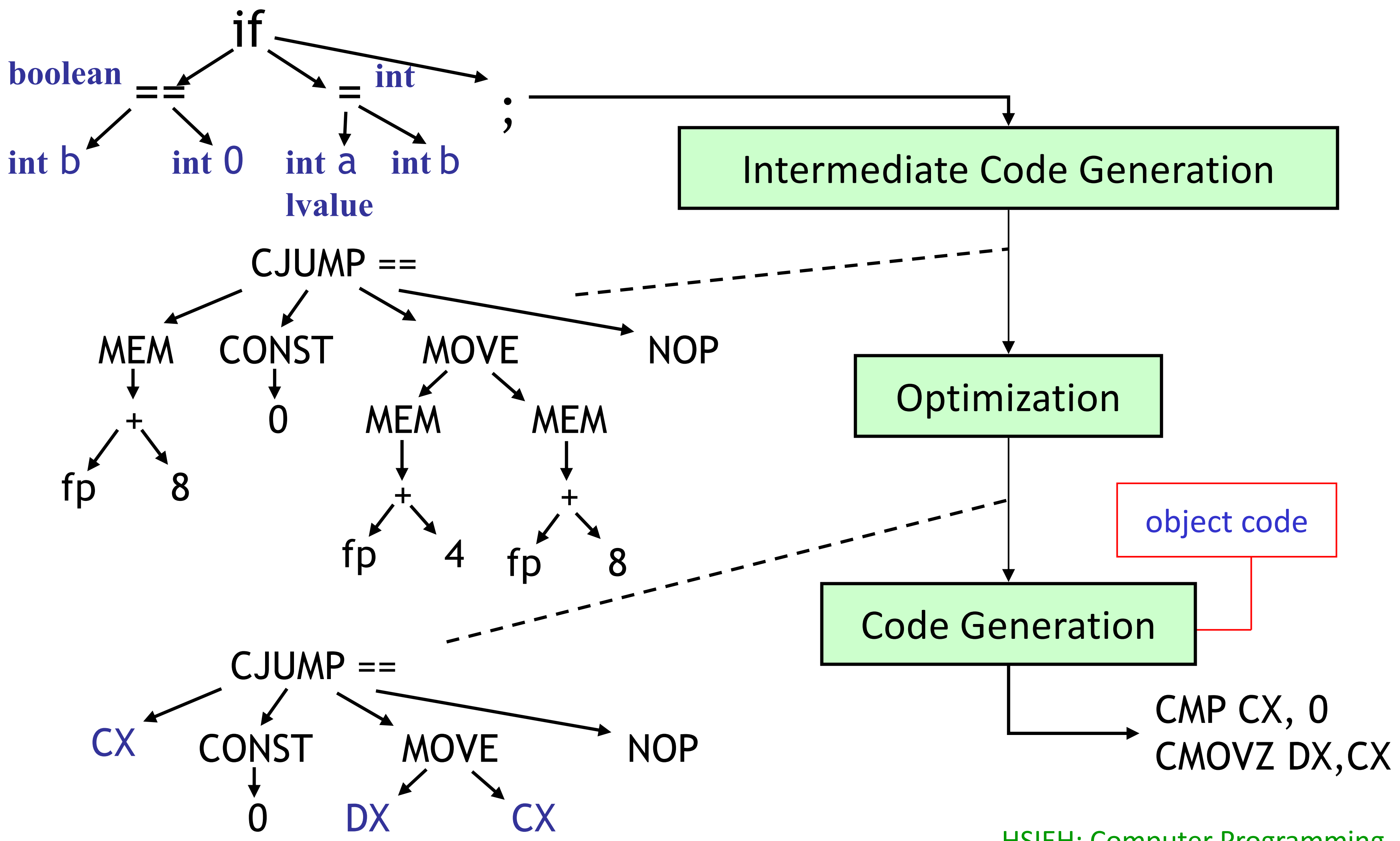
Compiling the Source Code



An Example on Compilation (1/2)



An Example on Compilation (2/2)



```
CMP CX, 0
CMOVZ DX,CX
```

Computer Programming

First C++ Program

First C++ Program ("C"-Style Output)

```
/*
 This is my first C++ program!
 It shows a message on the console.
 */
#include <cstdio>    // header for the printf() function

// the main() function
int main( )   // program entry
{
    printf("This is my first C++ program!");
    return 0;
}
```

More precisely, use

```
std::printf("This is my first C++ program");
```

```
This is my first C++ program!
```

```
/*
 This is my first C++ program!
 It shows a message on the console.
 */
#include <cstdio> // header for the printf()

// the main() function
```

Comments

- Comments
  - Explain programs to other programmers
  - For your future reference
  - Ignored by the compiler
  - Single-line comment
    - Begin with `//`
  - Multi-line comment
    - Begin with `/*`
    - End with `*/`

Valid comment:

```
////////// comment //////
```

Valid comment:

```
/***********************
 * comment
 ***********************/
```

Problematic comment:

```
/* /* comt1 */  cmt2 */
```

(nested comment)

☞ It is good practice to always write comments so you will not forget why you wrote codes this way

C++ Programming Style

- C++ has strict rules (e.g. case sensitive) but also allows some free writing styles
  - ☞ Use white space characters for formatting
- White space characters
  - Newline character ("Enter" key), space, and tab
  - Ignored by the compiler
- Writing style
  - Indentation
  - {} alignment

Note that operators and symbols cannot be broken by the white space characters (e.g. `/*`, `*/`, `//`, and `<<`)

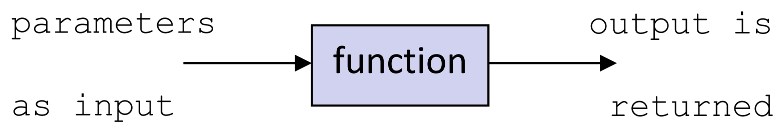
```
int
main(
){printf("This is my first C++ program!"); return 0;}
```

A Simpler Version

```
int main( )
{

    return 0;
}
```

```
int main( )
{


}
```

The `main` Function



- Programming using C++
  - C++ allows you to "program" the computer to do what you want – by writing "functions"
  - ☞ A function (you name it) can accept parameters, perform some tasks, and then return the results
  - ☞ Irrespective of the functions you want, every C++ program starts with the function called `main`
  - ☞ The `main` function needs to return an integer value to the caller (OS shell) when it finishes the execution

```
int main ( )
{
   statement 1;
   …
   return 0;
}
```

Returns an integer to the caller (OS shell)

No input parameters to the `main` function

Detailed instructions to be performed by a function are enclosed in { }

Keywords `return` and `int`

- `return` statement
  - One of several ways to exit a function
  - When used at the end of main
    - The value `0` indicates to the caller (OS) that the program has terminated successfully
  - ☞ If omitted, a value of `0` is returned automatically
- Data type `int`
  - Used for data with the integer type
    - Integer values: 1, 2, -1, 0,…
    - Cannot be used for fractions (2.9, -1.33, …)
  - ☞ In the program, it indicates that the value returned by the function `main()` is an *integer*

```
// the main() function

int main( ) // program entry
{
    printf("This is my first C++ program!");
    return 0;
}
```

C++ Keyword

- Keyword
  - A word reserved by C++ for a specific use
  - ☞ Cannot be used as variable and function names

Keywords common to C and C++

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

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

A C++ Program with Two Functions

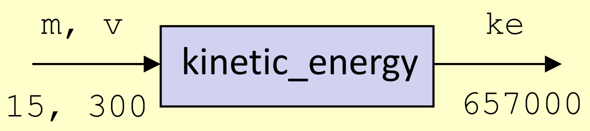
Function declaration

Function invocation

Function definition

```
#include <cstdio>
double kinetic_energy(int, double);
int main( )
{
    double energy;                                      // variable
    energy = kinetic_energy(15, 300);                   // function_call
    printf("The value of the kinetic energy is: %.3f", energy);
    return 0;
}
// user-defined function
double kinetic_energy(int m, double v)
{
    double ke;
    ke = 0.5*m*v*v;
    return ke;
}
```

Don't worry about the statements inside each function for now -- we will explain them later in the class



Before invoking a function, the function needs to be declared for the compiler to do error checking

First C++ Program ("C++"-Style Output)

```
/*
 This is my first C++ prog
 It shows a message on the
 */
#include <iostream> // hea

// the main() function
int main( )   // program en
{
    std::cout << "This is
    return 0;
}
```

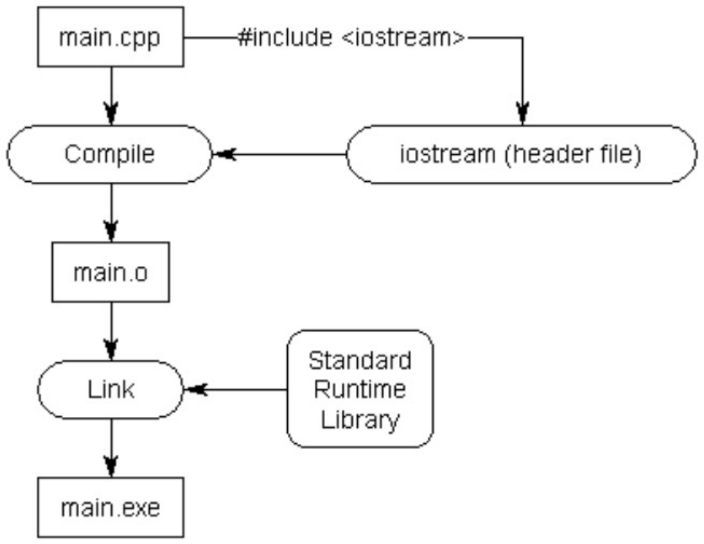
```
/*
 This is my first C++ program!
 It shows a message on the…
 */
#include <cstdio> // header…

// the main() function
int main( )   // program entry
{
    printf("This is my …");
    return 0;
}
```

```
std::cout << "This is my first C++ program!";
```

```
This is my first C++ program!
```

Using Functions

- Function declaration
  - Most programs use (call) functions provided by the standard library or third-party library
  - The compiler requires *function prototypes (declarations)* be provided for syntax checking (e.g. # of parameters)
  - Prototypes of functions are usually stored in header files
  - The header file (source code) is used by the *compiler* at compile time (preprocessing)
  - The library (binary code) is used by the *linker* at link time for extracting the actual implementation of the function



Library

- Library and functions
  - A library consists of <u>object codes</u> of *pre-compiled functions* to execute complicated routines of tasks
  - The list of functions (and their prototypes) is maintained in a *header file* to be included in the source code
- Standard library
  - Standard library is provided to the programmers as part of the language
  - Built-in functions: math, string, I/O, time, …
  - ☞ *cf. library created by you – the programmer*
  - ☞ *cf. third-party library – neither by you nor by the user*

```
/*
 This is my first C++ program!
 It shows a message on the console.
 */
#include <cstdio> // header for the printf()

// the main() function
```

Preprocessor

- Preprocessor directives
  - Processed by the preprocessor before compiling
    - A line begins with #
  - `#include <cstdio>` ← *cf.* `#include <stdio.h>`
    - Tells the preprocessor to put the content of the header file `<cstdio>` here in the source code
    - Search the file `cstdio` in *system-defined directories*
  - `#include "cstdio"`
    - Similar to the case of using `<cstdio>`, but the search starts from the current directory (the directory the source file to be compiled is in) *before* searching the system-defined directories

☞ Built-in header files are usually included using `<>`, while user-supplied header files are usually included using `""`

Preprocessing of the Source Code

```
/*
 This is my first C++
 It shows a message on
 */
#include <iostream> //

// the main() function
int main( )   // progra
{
    std::cout << "This
    return 0;
}
```

```
// -*- C++ -*-
//===--------------------------- iostream ----------------------------===//
//
//                     The LLVM Compiler Infrastructure
//
// This file is dual licensed under the MIT and the University of Illinois
// Open Source Licenses. See LICENSE.TXT for details.
//
//===--------------------------- iostream ----------------------------===//

#ifndef _LIBCPP_IOSTREAM
#define _LIBCPP_IOSTREAM

#include <ios>
#include <streambuf>
#include <istream>
#include <ostream>

namespace std {

extern istream cin;
extern ostream cout;
extern ostream cerr;
extern ostream clog;
extern wistream wcin;
extern wostream wcout;
extern wostream wcerr;
extern wostream wclog;

}  // std
```

```
This is my first C++ program!
```

C++ Statement vs. Directive

- Statement
  - Instruct the program to perform an action
  - All statements end with a semicolon (;)
  - It is possible to write many statements per line or write a single statement that takes many code lines

```
printf("This is my first C++ program!"); return
0;
```

```
#include \
<cstdio>
```

Unless the line is "escaped" with the backslash symbol \

- Directive
  - Instruct the preprocessor to perform an action
  - Preprocessor directives do not end with ;
  - Preprocessor directives *extend only across a single line*

The `printf` Function

- Standard I/O library
  - The library provides functions for input/output operations such as reading from an input unit (e.g. keyboard) and writing to an output unit (e.g. console)
  - Prototypes of functions in this library is stored in the header file `cstdio`
- `printf()`
  - The function accepts a string as an input parameter (argument) and shows the string to the standard output (console)
  - A string is specified by enclosing the characters in " "
  - ☞ Include the header file `cstdio` before the function `printf()` is called in the program

The `cout` Object

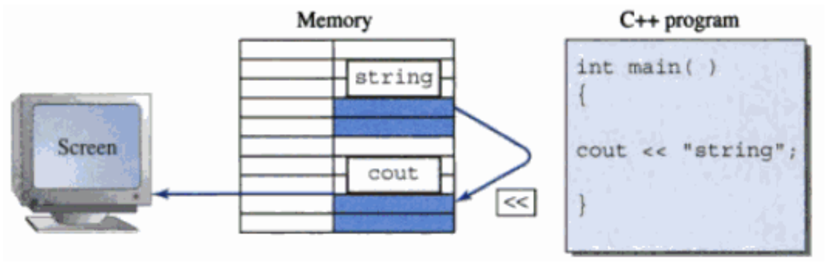
- `cout`
  - `cout` is the name of an object – just like the variable name
  - ☞ Is it a C++ keyword?
  - An object is a self-contained entity that consists of both data and procedures to manipulate the data
  - ☞ Function $\rightarrow$ black box, object $\rightarrow$ "smart" black box
  - ☞ It is "connected" to the standard output (screen)
  - ☞ Data sent to the `cout` object will be displayed in the appropriate form on the standard output (i.e., screen)
  - ☞ How is data sent to `cout` for display?

The << Operator

- Stream insertion operator <<
  - In C++, input and output are represented as a stream of characters
  - The right operand is inserted into the left operand
    - The operator "points" in the direction of where the data goes
  - Example
    - `cout << "string";`
    - The above C++ statement inserts the string `string` into the `cout` object, which will then display the string to the screen

☞ The `cout` object is provided by the standard library

- Need to include the header file `iostream` to tell the compiler (declare) what the identifier `cout` is



```
int main( )
{
cout << "string";
}
```

```
#include <iostream> // header for std::cout
using namespace std;

// the main() function
int main( ) // program entry
{
    cout << "This is my first C++ program!";
```

Namespace

- Namespace
  - Namespace allows the global scope of naming (variables, objects, functions, ...) to be divided in "sub-scopes", each one with its own name
  - Each namespace defines a scope in which identifiers are kept
  - `std::`
    - Specifies an identifier that belongs to "namespace" `std`
    - C++ standard library puts all of its entities within the `std` namespace
  - `std::cout`
    - The standard output stream object `cout` resides in the namespace `std`

`::` is the scope resolution operator

Variables (objects) declared in the file `iostream` are put in the namespace `std`

First C++ Program Revisited

```
/*
 This is my first C++ program!
 It shows a message on the console.
 */
#include <iostream> // header for std::out
using namespace std;

// the main() function
int main( )   // program entry
{
    cout << "This is my first C++ program!";
    return 0;
}
```

Alternatively, use `using std::cout;` to bring `cout` to the current scope

Specification of the namespace to use

It can be placed inside the main body

Directly specify the use of the `cout` object declared in namespace `std`

```
This is my first C++ program!
```

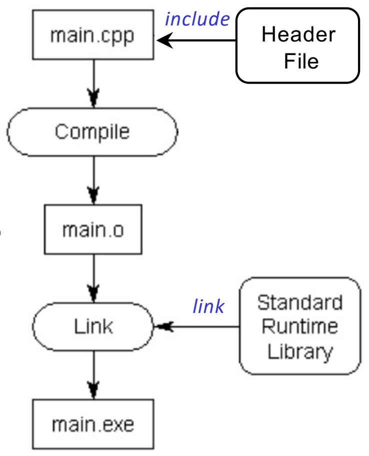
Creating a Program

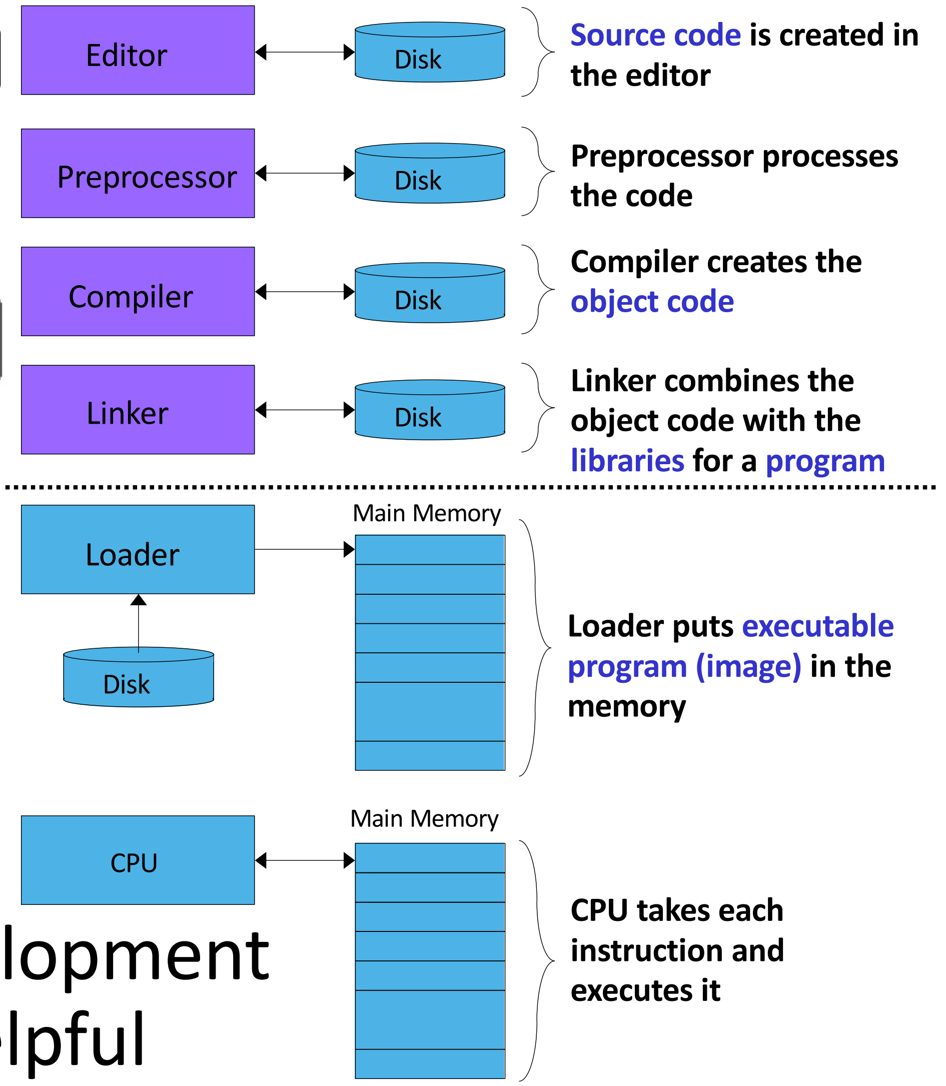
Most IDEs use a *Project* file to keep information of source files, libraries, as well as compiling and linking options needed for creating a program

- Phases
  - Edit
  - Preprocess
  - Compile
  - Link
  - Load
  - Execute

Refer to as the "build" process for creating a machine-executable program

☞ A good integrated development environment (IDE) is helpful

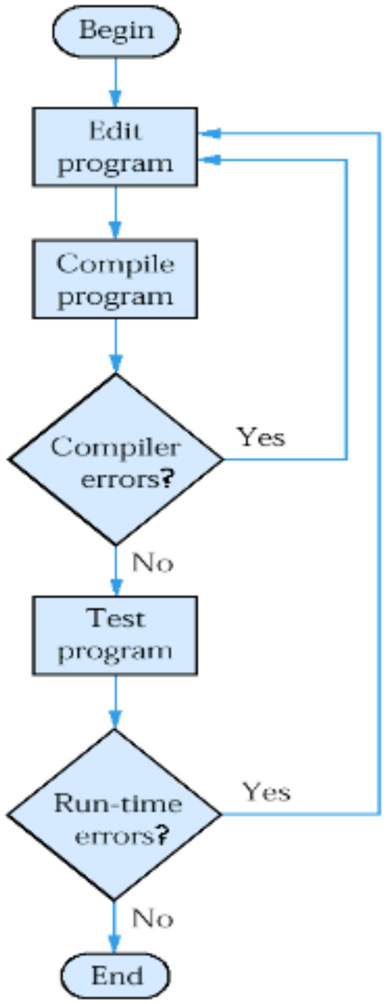




Editor
Disk
Source code is created in the editor
Preprocessor
Disk
Preprocessor processes the code
Compiler
Disk
Compiler creates the object code
Linker
Disk
Linker combines the object code with the libraries for a program
Main Memory
Loader
Disk
Loader puts executable program (image) in the memory
Main Memory
CPU
CPU takes each instruction and executes it

Programming Environment

- Integrated development environment (IDE)
  - Edit, debug, and compile (build)
- Choice of development environment
  - ☞ CLion (cross platform) (free for students)
    https://www.jetbrains.com/clion/
  - ☞ Visual Studio Code (cross platform)
    https://https://code.visualstudio.com
  - ☞ Code::Blocks (cross platform)
    http://www.codeblocks.org
  - ☞ Dev-C++ (Windows only)
    https://github.com/Embarcadero/Dev-Cpp/releases
- Note the IDE and the compiler are not necessarily tied / bundled together
  - ☞ GCC C++ compiler (TDM-GCC on Windows)



More on the Compiler

Clang based on LLVM is an *alternative to GCC* for C/C++ and Objective C/C++

- GNU compiler collection (GCC)
  - GCC includes front ends for C and C++ (among others), as well as libraries for these languages (libstdc++,...)
  - GCC has been adopted as the standard compiler by many modern Unix-like OS, including Linux and the BSD family
  - MinGW includes a port of GCC and tools (e.g. assembler and linker) for Windows (Win32)
  - MinGW-w64 supports both 32-bit and 64-bit programs
- TDM-GCC
  - TDM-GCC combines the GCC toolset, MinGW and MinGW-w64 to create an open-source alternative to Microsoft's compiler (for any version since Windows 98)

We will discuss more on 32-bit and 64-bit programs later

Review

- Programming language
  - Compiler vs. interpreter
  - Translation from the editable source code to an executable program
- First C++ program
  - Comment, statement, and directive
  - Standard library and header file