Computer Programming

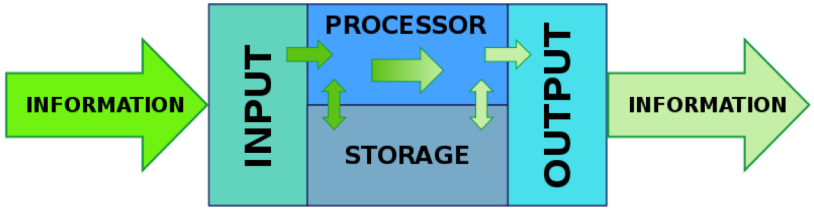
Introduction

Hung-Yun Hsieh
September 19, 2018

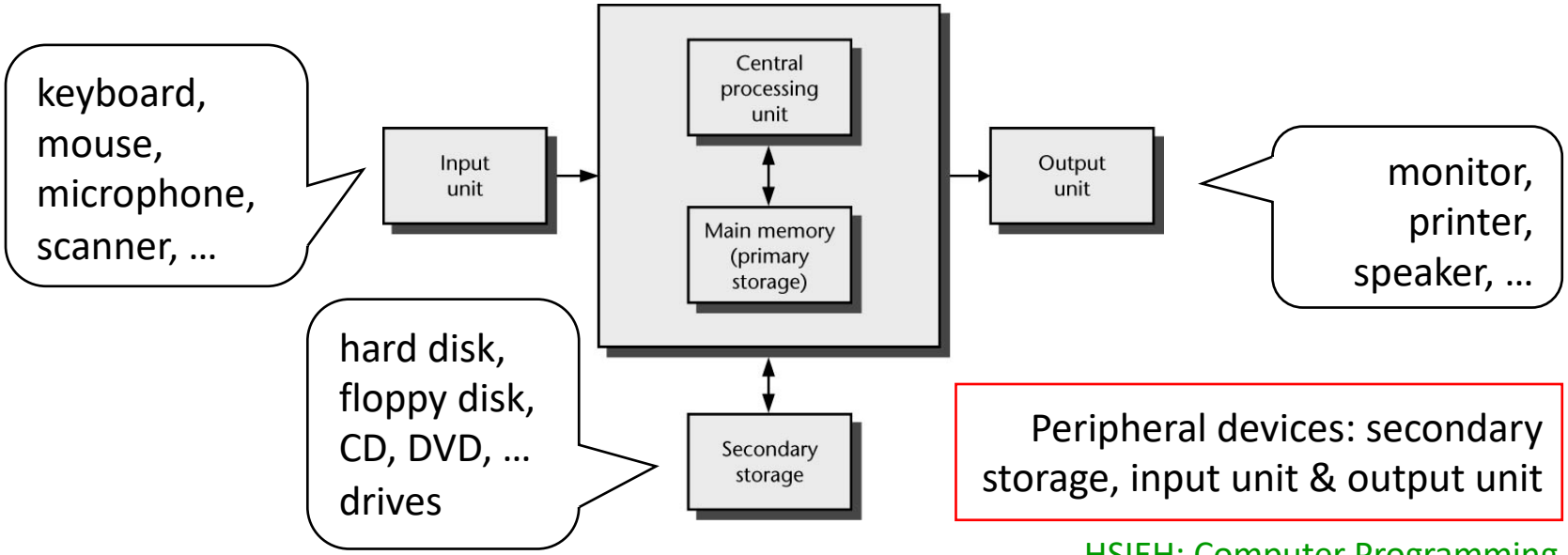
Abstraction of a Computer

- 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



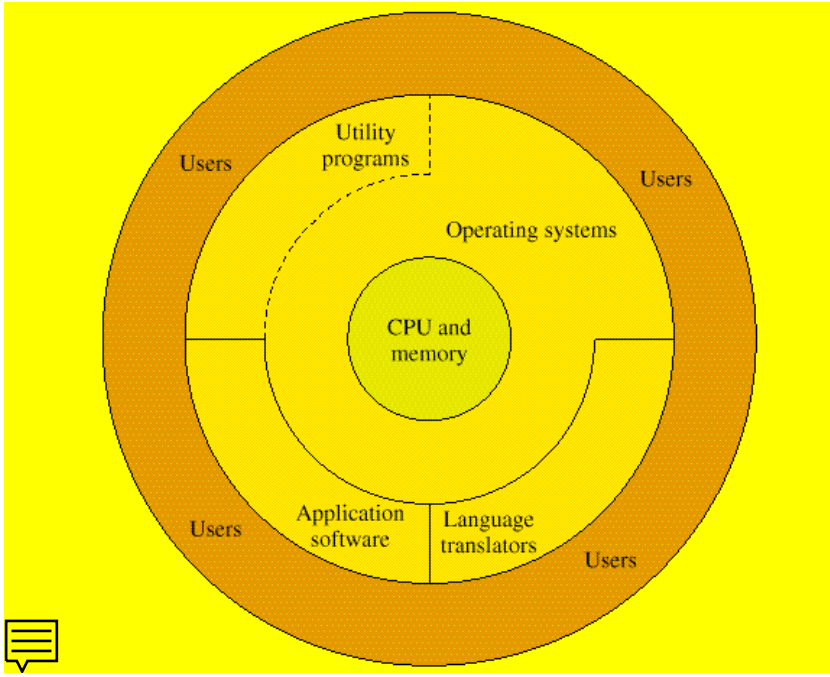
② Software

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

      computer resource management, program execution, multi-tasking, disk access, …

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



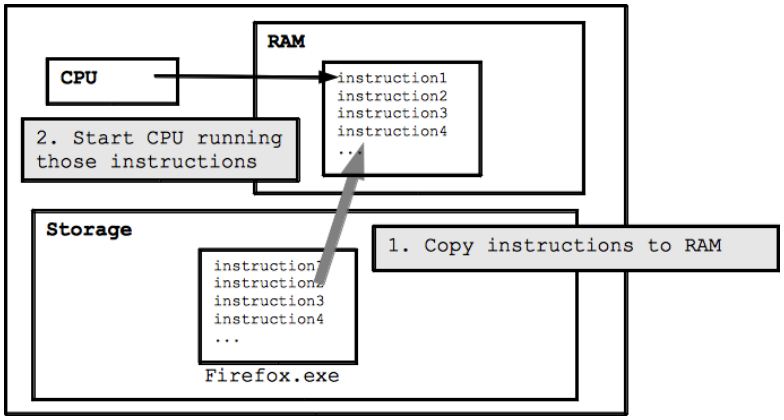
☞ Computer programs of your own design

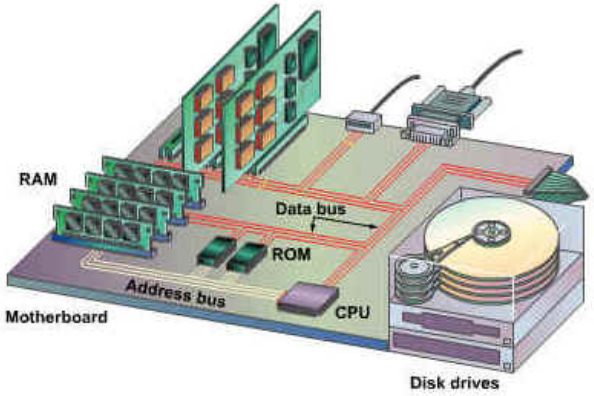
Computer Programming

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



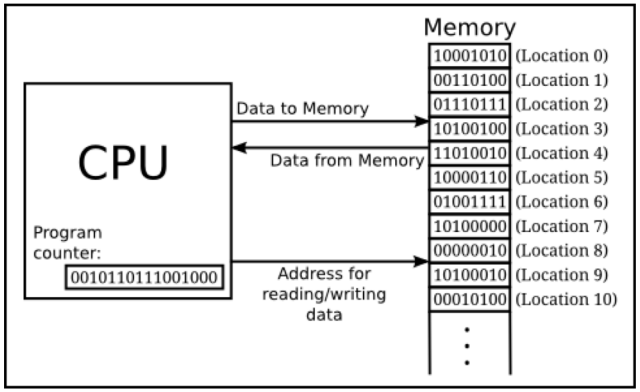


CPU Instructions

- 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

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	Immediate value

$r1 = $r2 + 350



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

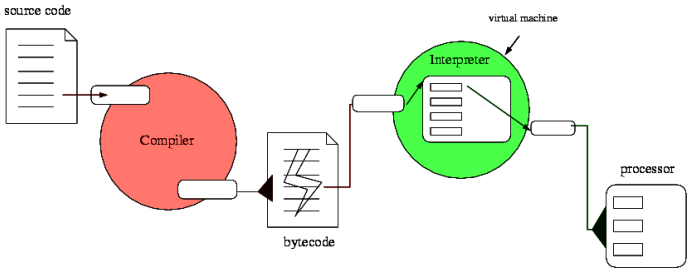
Machine code	Assembly code	Description
001 1 000010	LOAD #2	Load the value 2 into the Accumulator
010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
001 1 000101	LOAD #5	Load the value 5 into the Accumulator
010 0 001110	STORE 14	Store the value of the Accumulator in memory 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

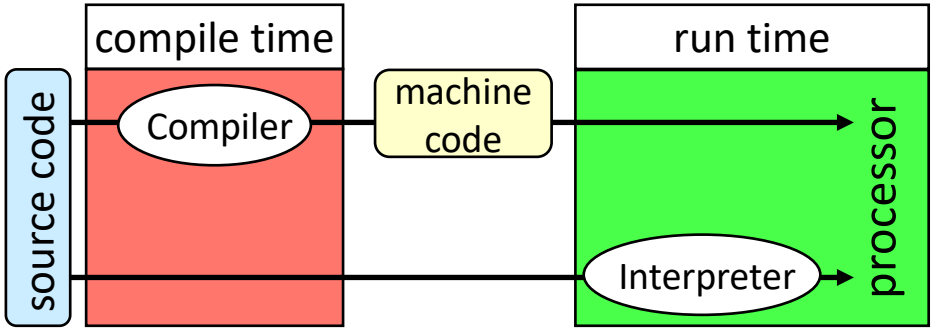
- High-level language
  - Similar to everyday English while using common mathematical notations
  - A single statement can accomplish complicated tasks performed by multiple CPU instructions
  - Many programming languages are created with specific purposes
    - Database processing, text processing, artificial intelligence, math operations

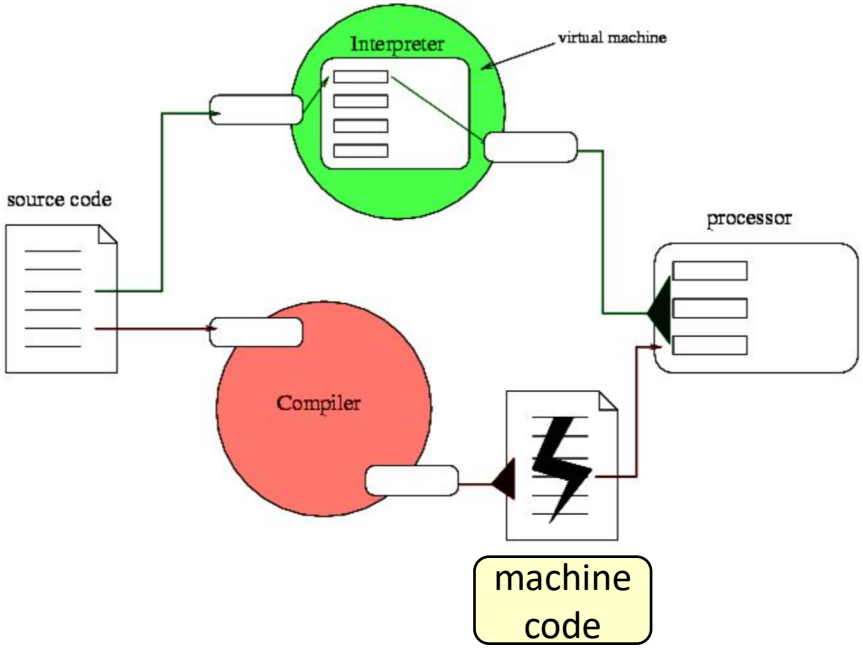
  - Translator
    - Compiler – convert to machine code before execution
    - Interpreter – directly execute high-level language programs

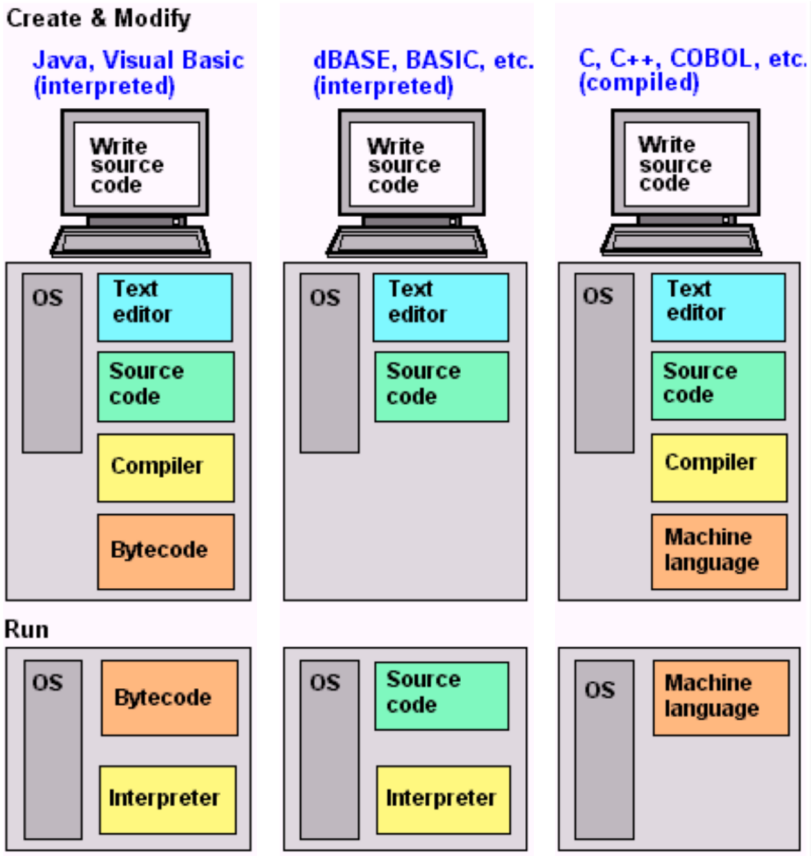
Compiler vs. Interpreter



- Compile time vs. run time behavior

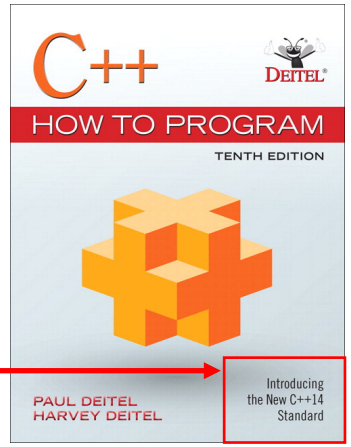






C and C++

- 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



C/C++ Descendents

- Objective-C
  - An object-oriented language based on C
  - The object syntax is derived from Smalltalk
- Java
  - An object-oriented language that derives much of its syntax from C and C++
  - A Java application can run on any computer architecture (Java virtual machine)
- C#
  - An object-oriented language designed for .NET platform
  - Built based on Java and C++
- ☞ Python

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;
}
```

☞ Transformation from the *human-readable* source code to *machine-executable* machine code (binary code)

Elements of a Programming Language

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

Token

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

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.

☞ Library

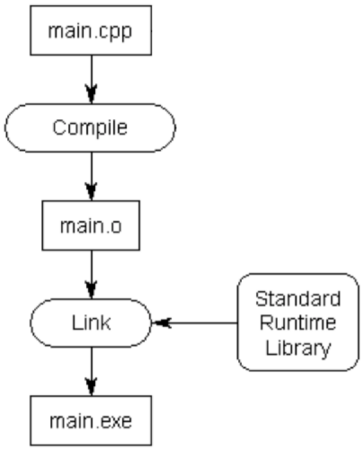
Library

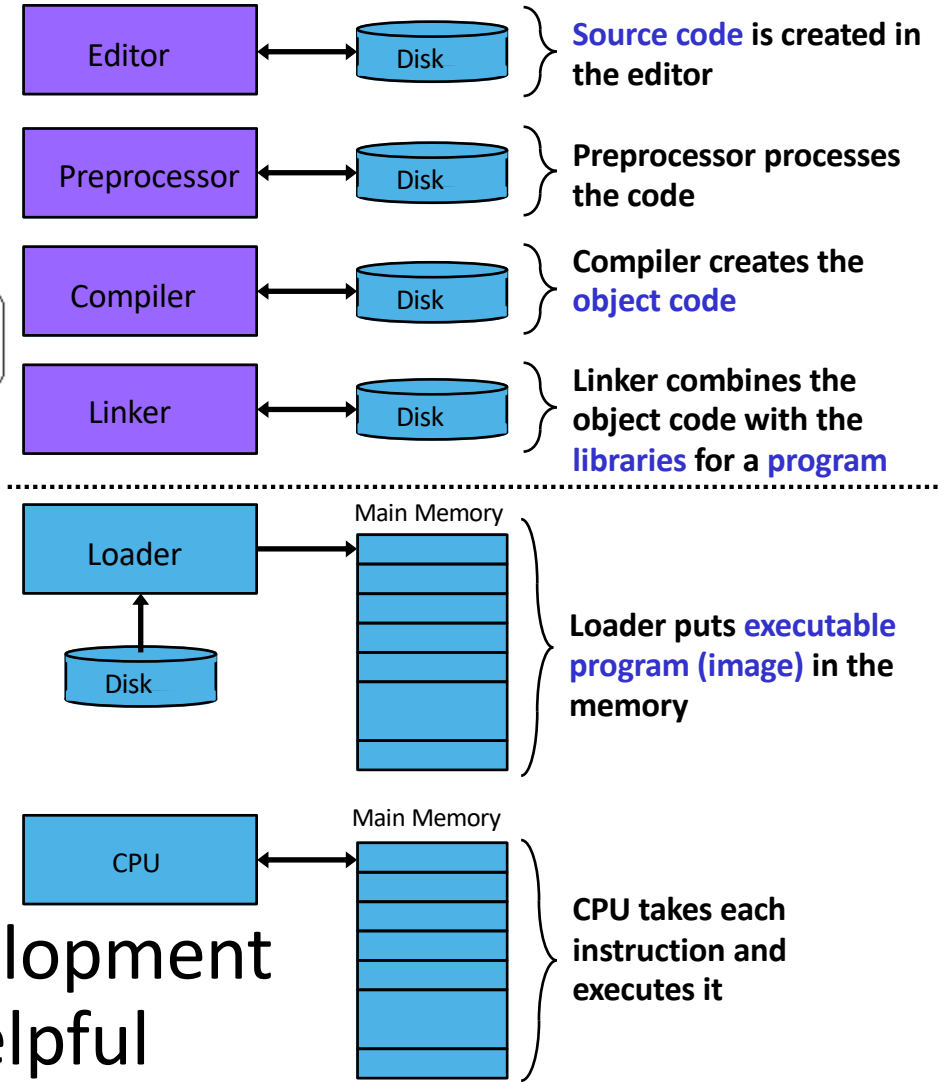
- Function and library
  - Functions are "self-contained" modules of machine codes that accomplish some well-define task
  - Functions usually "take in" data, process it, and "return" a result
  - A library contains a collection of functions (among others) for use by programmers
- Standard library
  - Standard library is provided to the programmers as part of the language
  - ☞ *e.g. third-party library*
  - Built-in functions: math calculation, string handling, I/O processing, memory management, …

Creating a Program

A library consists of object codes of pre-compiled functions to execute complicated routines of tasks

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



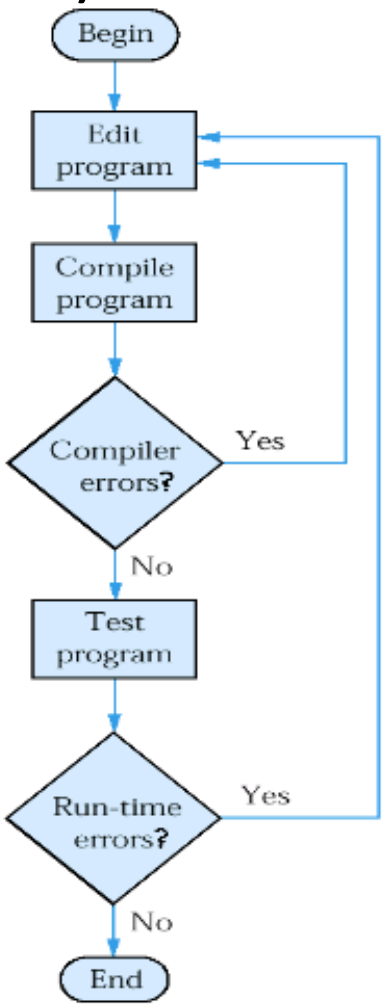


Refer to as the "build" process for creating a machine-executable program

☞ A good integrated development environment (IDE) is helpful

Programming Environment

- Integrated development environment (IDE)
  - Edit, debug, and compile (build)
- Choice of development environment
  - ☞ Code::Blocks
    http://www.codeblocks.org
  - ☞ CLion (free for student license)
    https://www.jetbrains.com/clion/
  - ☞ (Orwell) Dev-C++ (MS-Windows only)
    http://orwelldevcpp.blogspot.tw
- Note the IDE and the compiler are not necessarily tied / bundled together
  - ☞ GCC C++ compiler (TDM-GCC on Windows)



Computer Programming

First C++ Program

First C++ Program ("C"-Style I/O)

```
/*
 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!
```

Comments

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

// the main() function
```

- 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

Preprocessor

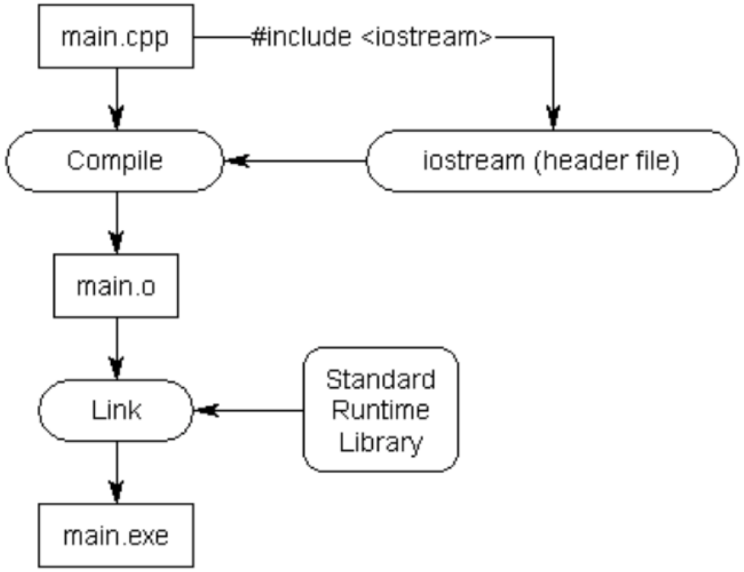
```
/*
 This is my first C++ program!
 It shows a message on the console.
 */
#include <cstdio> // header for the printf()

// the main() function
```

- Preprocessor directives
  - Processed by the preprocessor before compiling
    - A line begins with #
  - `#include <cstdio>`
    - 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 is in) *before* searching the system-defined directories

☞ Declarations for functions provided in the standard library are scattered across many different header files

Function and Header File

- Header file
  - Functions have well-defined interfaces (prototypes)
    - Function name, input parameters, output, and performed task
  - Header file contains prototypes (declarations) of functions to help the compiler check if a function is correctly used
  - The header file (source code) is used at compile time (preprocessing) for checking
  - The library (binary code) is used at link time for extracting the actual code of the function



The `main` Function

```
// the main() function

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

- 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

```
int main ( )
{
   statement 1;
   statement 2;
   …
   return 0;
}
```

No input parameters to the `main` function

Returns an integer to the caller (OS shell)

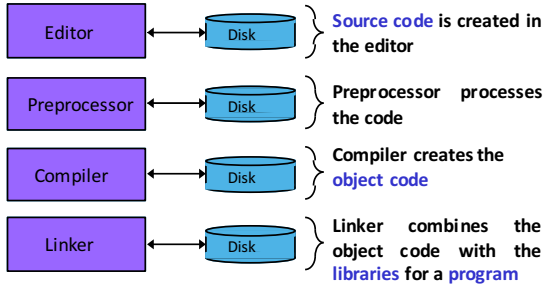
Detailed instructions to be performed by a function are enclosed in **{ }**

Keywords `int` and `return`

- 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

- `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

The `printf` Function



- Standard library
  - C/C++ provides many (standard or built-in) functions to facilitate programming
  - The binary code is stored in the *library* (to be included during linking)
  - The declaration is stored in the *header* (to be included before compiling)
- `printf()`
  - The function accepts a string as an argument and writes the string to the standard output (console)
  - A string is specified by enclosing the characters in " "
  - The header file to include for `printf()` is `cstdio`

Function declaration allows the compiler to know whether the function call syntax is correct or not

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;}
```

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

First C++ Program ("C++"-Style I/O)

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

// the main() function
int main( )   // program e
{
    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;
}
```

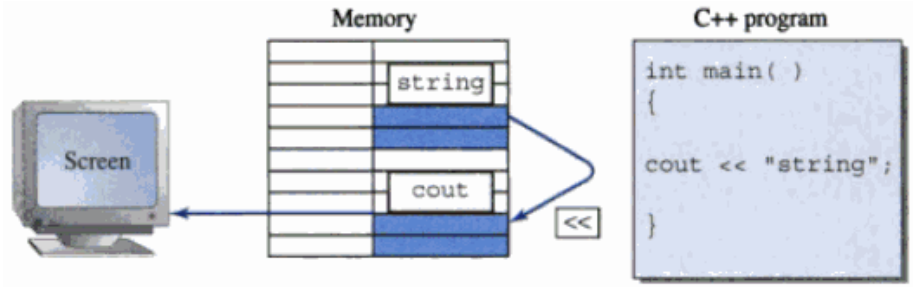
This is my first C++ program!

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 → black box, object → "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
  - Value to right (right operand) inserted into 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 tell the compiler that `cout` will be used in the program



Namespace

```
#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 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` defined 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;
}
```

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!
```

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