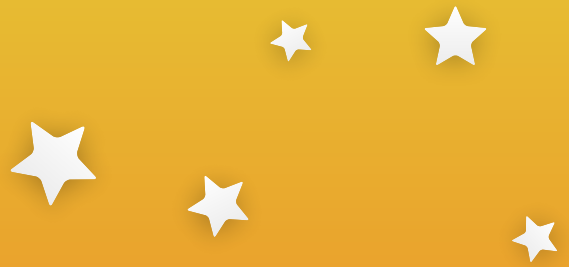


Lecture 1

Introduction

Labs for the Course

- Laboratory experiments (labs) are an integral part of the course
- One cannot learn to program without actually writing programs
  - Laboratory experiments for the course are specifically designed for the students to practice ‘writing programs’
- Come prepared before hand for the labs
  - Not possible to just turn up to the laboratory and expect to be able to complete the exercises

Integrated Development Environment(s)

- **CodeBlocks** – An open-source IDE
  - Download link: https://www.codeblocks.org/downloads/
    - Recommended to restart OS after installation
- **OnlineGDB** – an online web-based compiler for C/C++
  - Link: https://www.onlinegdb.com/online_c++_compiler
  - Some others as well
    - Link: https://onecompiler.com/cpp
    - Link: https://www.programiz.com/cpp-programming/online-compiler/
    - https://www.jdoodle.com/online-compiler-c++/
    - https://cpp.sh/

Course Handbooks

- Text Book(s):
  - “Problem Solving and Program Design in C” by Hanly & Koffman
  - “C++ Programming: From Problem Analysis to Program Design” by D.S. Malik
- Reference Book(s):
  - “C: How to Program” by Deitel & Deitel

Programming Languages

Three types of programming languages

1. Machine languages
   - Strings of numbers giving machine specific instructions
   - Programs written in machine language entirely consist of 1s and 0s
   - Example:
     - 00101010 000000000001 000000000010
     - 10011001 000000000010 000000000011

Programming Languages

Three types of programming languages

2. Assembly languages
   - English-like abbreviations representing elementary computer operations
   - Example:
     - LOAD BASEPAY
     - ADD OVERPAY
     - STORE GROSSPAY
   - Assembler, a computer software that translates a program written in assembly language into machine language

Programming Languages

Three types of programming languages

3. High-level languages
   - Statements are similar to everyday English
   - Example:
     - grossPay = basePay + overTimePay
     - Print “Your total pay is ” grossPay
   - Compiler, a computer software that translates a program written in a high-level language into machine language
   - Examples of high-level languages
     - Basic, FORTRAN, COBOL, Pascal, C, C++, C#, Java, Python

Programming Methodologies

Two popular approaches to programming

- Structured
- Object-oriented

Programming Methodologies

Structured programming:

- Dividing a problem into smaller subproblems
- Also known as top-down (or bottom-up) approach
- Stepwise refinement
- Modular programming

Programming Methodologies

Object-oriented programming (OOP):

- 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
- An object combines data and operations on the data into a single unit
- Must learn how to represent data in computer memory, how to manipulate data, and how to implement operations

C++ Language

- Without software, the computer is useless
- Software is developed with programming languages
- C++ is a programming language, evolved from C
- Suited for a wide variety of programming tasks
- Designed by Bjarne Stroustrup at Bell Laboratories in early 1980s
- Many different C++ compilers are available
- C++ programs were not always portable from one compiler to another
- In mid-1998, ANSI/ISO C++ language standards were approved
- Second standard called C++11 approved in 2011

Your First C/C++ Program

```
#include <stdio.h>
int main() {
        printf("My first C program.“);
        return 0;
}
```

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

- Output:

```
My first C program.
```

Executing a C/C++ Program

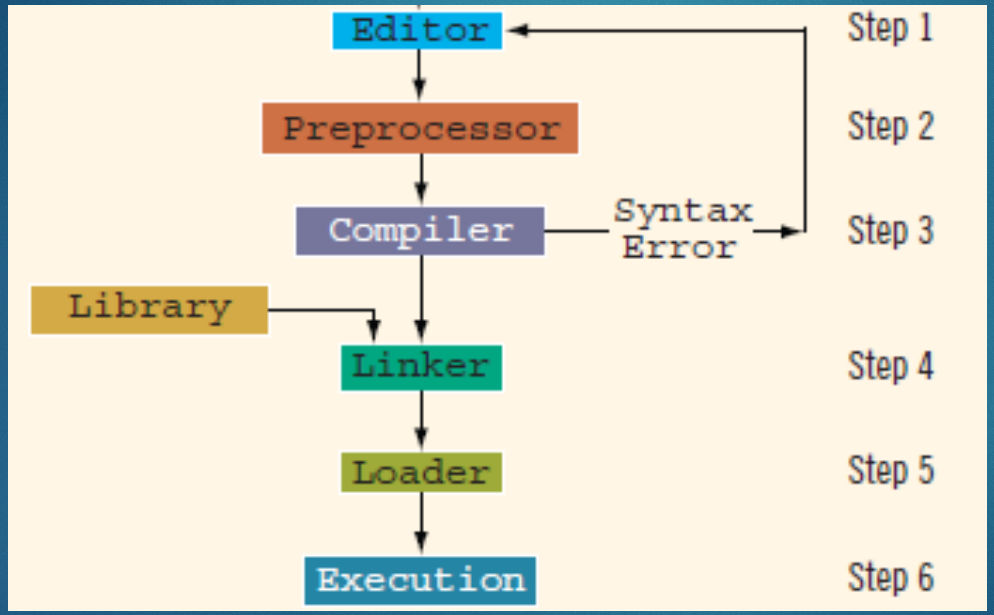
- To run/execute a C/C++ program:
  - Use an **editor (IDE)** to create a source program in C language
    - Preprocessor directives begin with # and are processed by a preprocessor
  - The **compiler** checks whether the program obeys the rules (syntax is correct), then translate it into machine language (object program)
  - The **linker** combines object program (compiled code) with other programs provided by the **SDK** (C standard libraries) to create executable code (.exe file)
  - The **loader** copies executable program into main memory and execute the program
  - Most of the IDEs do all this with a **Build** or **Rebuild** command/button

Executing a C/C++ Program

- Run code through compiler
- If compiler generates errors, carefully look at the code and remove errors
- Run code again through compiler
- If there are no syntax errors, compiler generates equivalent machine code
- Compiler does not guarantee that the program will run correctly
- Linker links machine code with system resources
- Once compiled and linked, loader places the program into main memory for execution

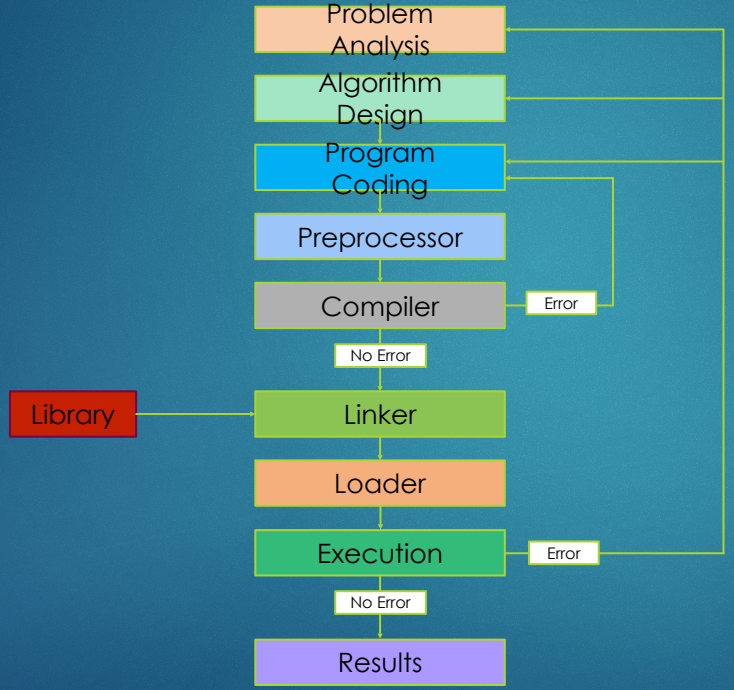
Executing a C/C++ Program

- Flowchart of how to execute a C++ program

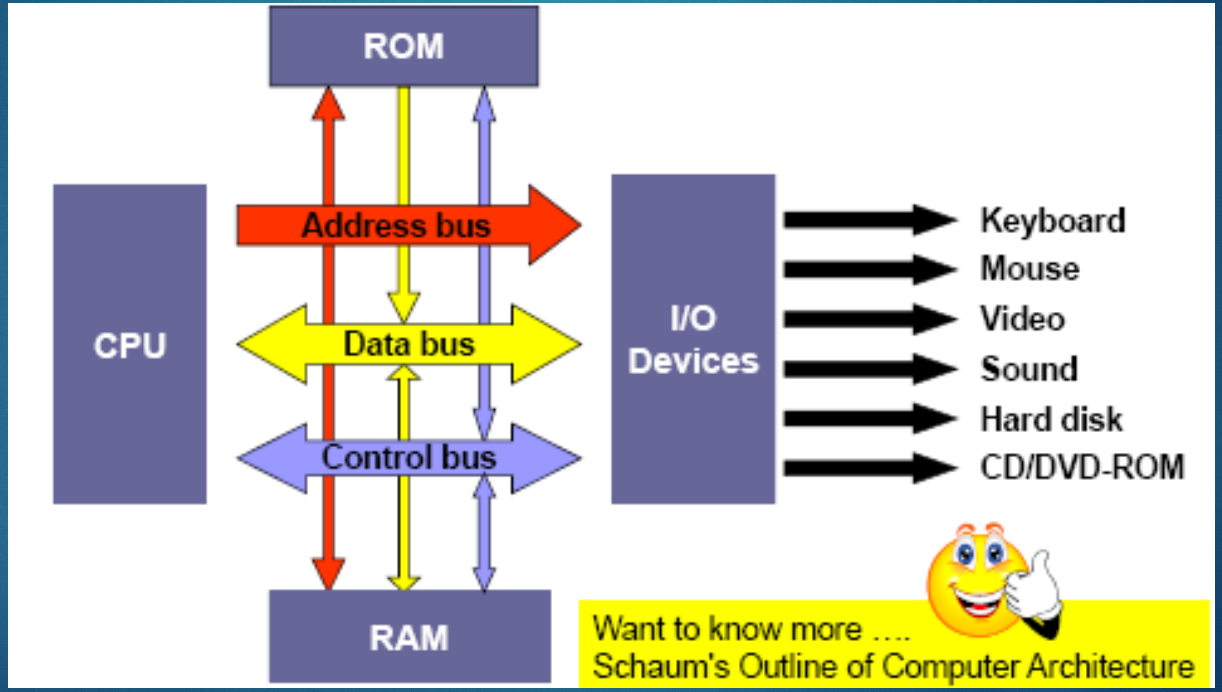


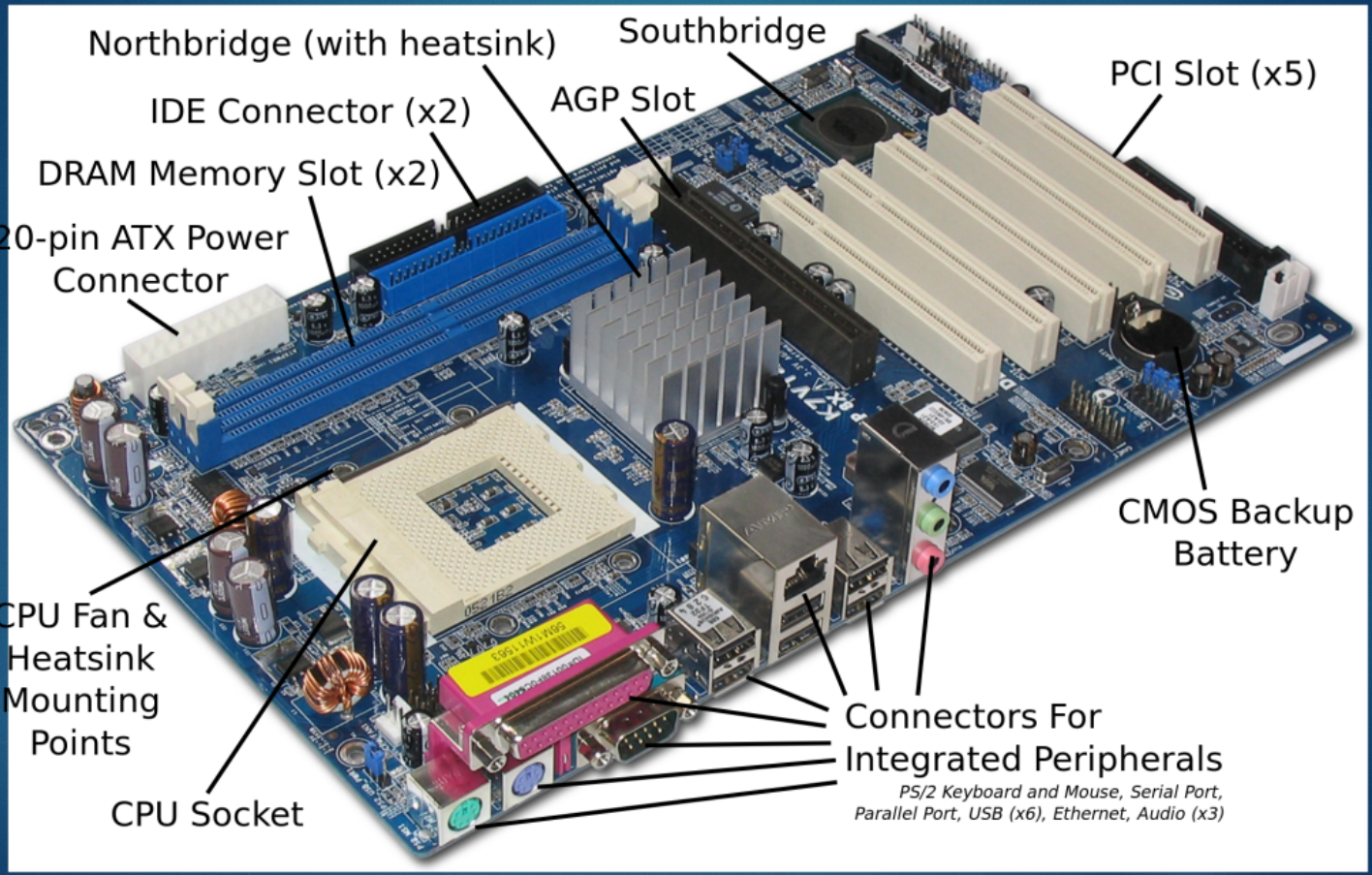
Developing a C/C++ Program

- Flowchart of how to develop a C++ program



What's Inside a Computer




Northbridge (with heatsink)
Southbridge
IDE Connector (x2)
AGP Slot
PCI Slot (x5)
DRAM Memory Slot (x2)
20-pin ATX Power Connector
CMOS Backup Battery
CPU Fan & Heatsink Mounting Points
CPU Socket
Connectors For Integrated Peripherals
PS/2 Keyboard and Mouse, Serial Port, Parallel Port, USB (x6), Ethernet, Audio (x3)

COMPUTER!

Some computerized devices		
Automobiles	Blu-ray Disc™ players	Building controls
Cable boxes	Desktop computers	Credit cards
CT scanners	GPS navigation systems	e-Readers
Game consoles	Lottery systems	Home appliances
Home security systems	MRIs	Medical devices
Mobile phones	Parking meters	Personal computers
Optical sensors	Printers	Robots
Point-of-sale terminals	Servers	Smartcards
Smart meters	Televisions	Smartphones
Tablets	TV set-top boxes	Thermostats
Transportation passes	ATMs	Vehicle diagnostic systems

3P's

- Programme
  - A set of rules for your computers to follow in order to achieve a goal
- Programmer
  - A person who produces the program
  - A person who makes a living from producing programs
- Programming language
  - A way for a programmer to write about the set of rules

Writing vs. Programming

- Setting the theme
- Structuring
- Writing
- Proof-reading

- Defining the problem
- Planning the solution
- Coding the program
- Testing the program

- **Documenting the program**
  - commenting

What is Computer Programming?

- A ***way to solve problems*** using a computer
- How?
  - Through a sequence of very clear and defined steps that aim to solve the problem -> this is called an **algorithm**
- Algorithms are not only used for computer programming, they are a ***general concept*** for problem solving

Defining The Problem

- Input
- Problem to be solved
- Output

Planning The Solution

- Algorithms
  - Ways of solving the same problem
  - Some fast; some slow
  - Some long; some short
- Formats
  - Pseudo code
  - Flow chart
  - Decision Tables

Algorithm

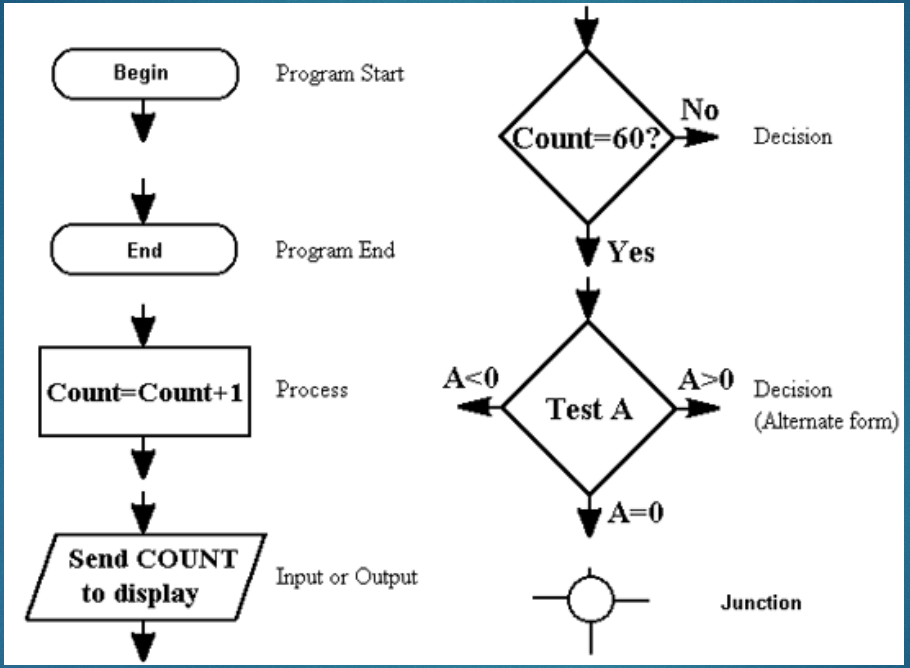
- The order of the actions in an algorithm is important
  - Example: algorithm to go to work every morning
    - Get out of bed;
    - Take a bath;
    - Get dressed;
    - Eat breakfast;
    - Take the bus to work.
- If I carry out the same actions in a different order
    - Get out of bed;
    - Get dressed;
    - Take a bath;
    - Eat breakfast;
    - Take the bus to work.
- Then, I will still get to work, however soaking wet

So order is important

Pseudo Code , Flowchart and decision table

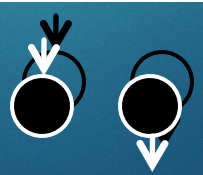
- However, programming languages can be complex for students that are starting to learn how to program
- The focus in programming fundamentals is also to learn how to solve a problem through a computer program
- Therefore, we will introduce the use of pseudo codes and flowcharts to describe algorithms (solutions of the proposed problems) and after that, we will learn how to translate them to a programming language, like C.
- Decision Tables are used to define clearly and concisely the word statement of a problem in a tabular form

Flow chart Symbols



Flow Lines

Flow Lines



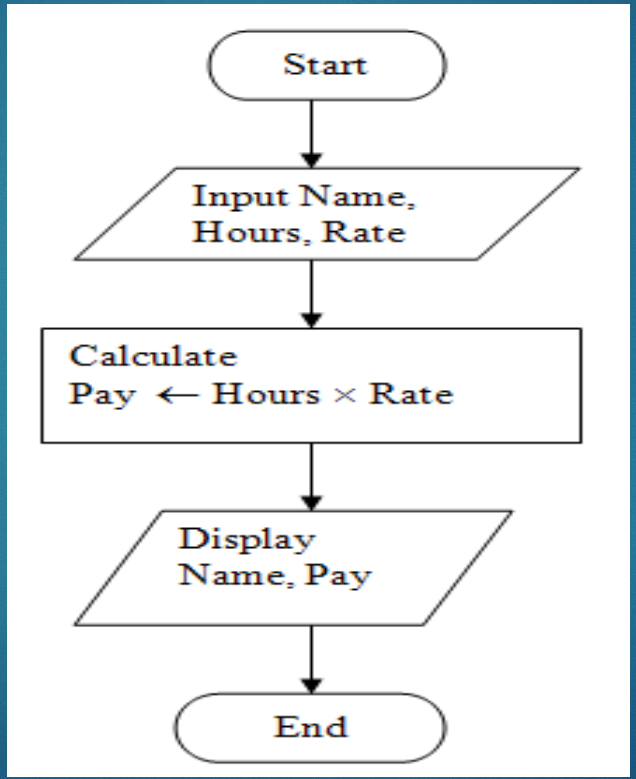
Connectors

Connectors

Pseudo Code

- Example: The following set of instructions forms a detailed algorithm in pseudo-code for calculating the payment of person.
  - Begin
  - Input the three values into the variables Name, Hours, Rate.
  - Calculate Pay = Hours * Rate.
  - Display Name and Pay.
  - End

Flowcharts


Start
Input Name,
Hours, Rate
Calculate
Pay ← Hours × Rate
Display
Name, Pay
End

Decision tables

Taking Decision on Discount	Decision Rule Numbers							
	R1	R2	R3	R4	R5	R6	R7	R8
Condition Statements								
Less than 50 units ordered	Y	Y	Y	Y	N	N	N	N
Cash on Delivery	Y	Y	N	N	Y	Y	N	N
Wholesale Outlet	Y	N	Y	N	Y	N	Y	N
Actions Taken								
Discount Rate 0%				X				
Discount Rate 2%		X	X					X
Discount Rate 4%	X					X	X	
Discount Rate 6%					X			

Flow Chart another Example

(b) Pseudo code

Place 0 in sum

Place 0 in counter

Enter first number

DOWHILE the number is not equal to 999

Add number to sum

Add 1 to counter

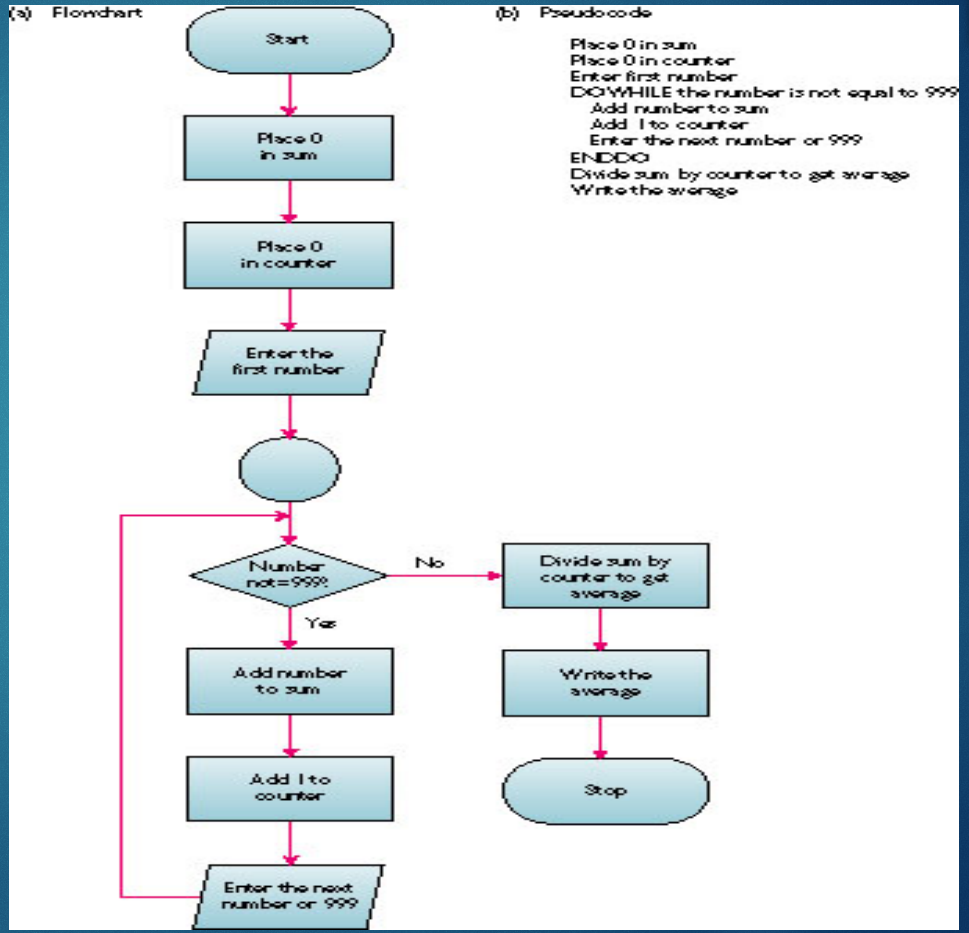
Enter next number or 999

ENDDO

Divide sum by counter to get average

Write the average

Accept series of numbers and display the average



Coding The Program

Polly Huang, NTU EE

- Sentence by sentence, word by word
- Detailed description in the chosen language

Testing The Program

Polly Huang, NTU EE

- Translation – compiler
  - Translates from **source** module into **object module**
  - Detects syntax errors
- Link – linkage editor (linker)
  - Combines **object module** with libraries to create **load module**
  - Finds undefined external references (run-time errors)
- Debugging
  - Run using data that tests all statements
  - Logic errors

Distinction

Polly Huang, NTU EE

- Source code
  - Your creation
- Object modules
  - Machine code of your creation
- Load modules
  - Machine code of pre-installed functions
- Executable programs
  - Combination of your and pre-installed machine code

Documenting The Program

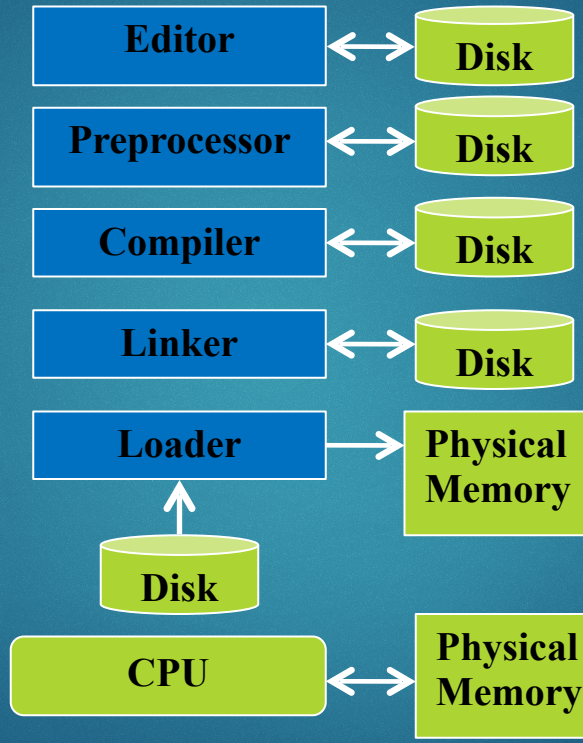
Polly Huang, NTU EE

- Comments within source code
- In plain English
- Convenient for
  - The programmer him/herself who needs to read the program later
  - Somebody else who needs to read the program

C Program Development Environment

Phases of C Programs:

- Edit
- Preprocessor
- Compile
- Link
- Load
- Execute



Program is created in the editor and stored on disk.

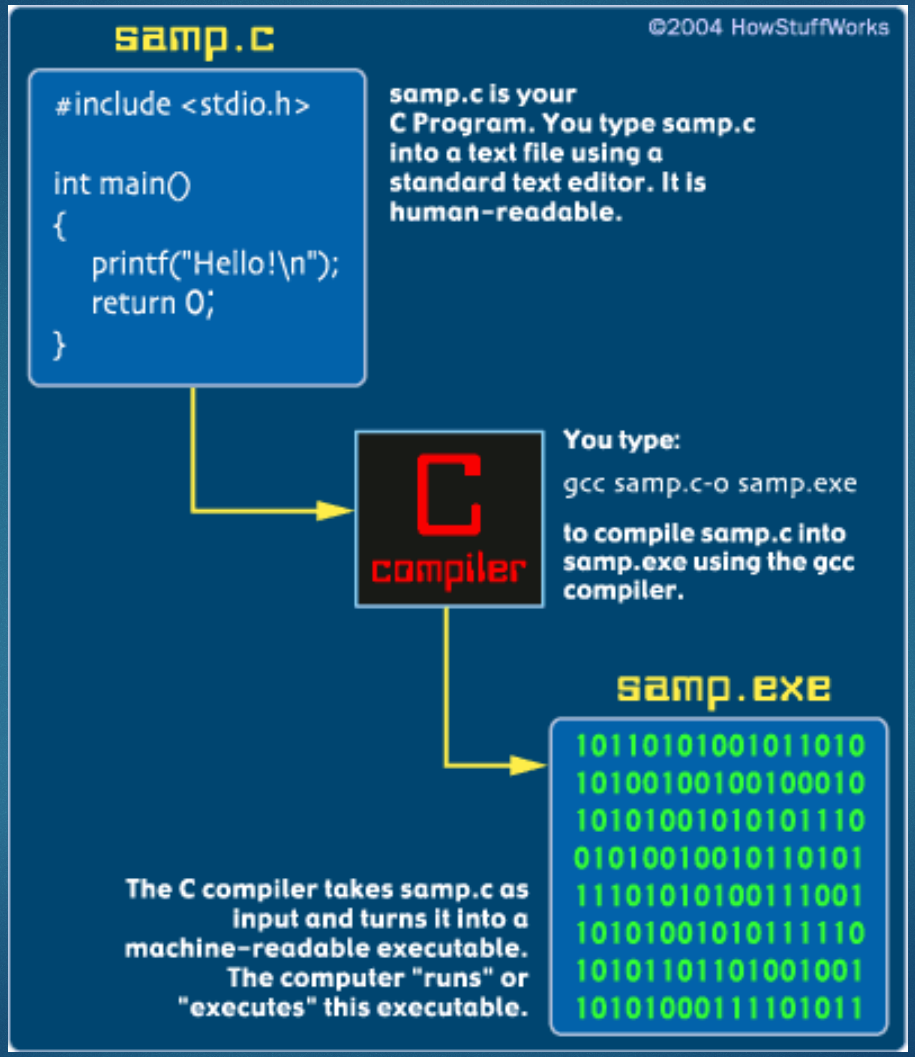
Preprocessor program processes the code.

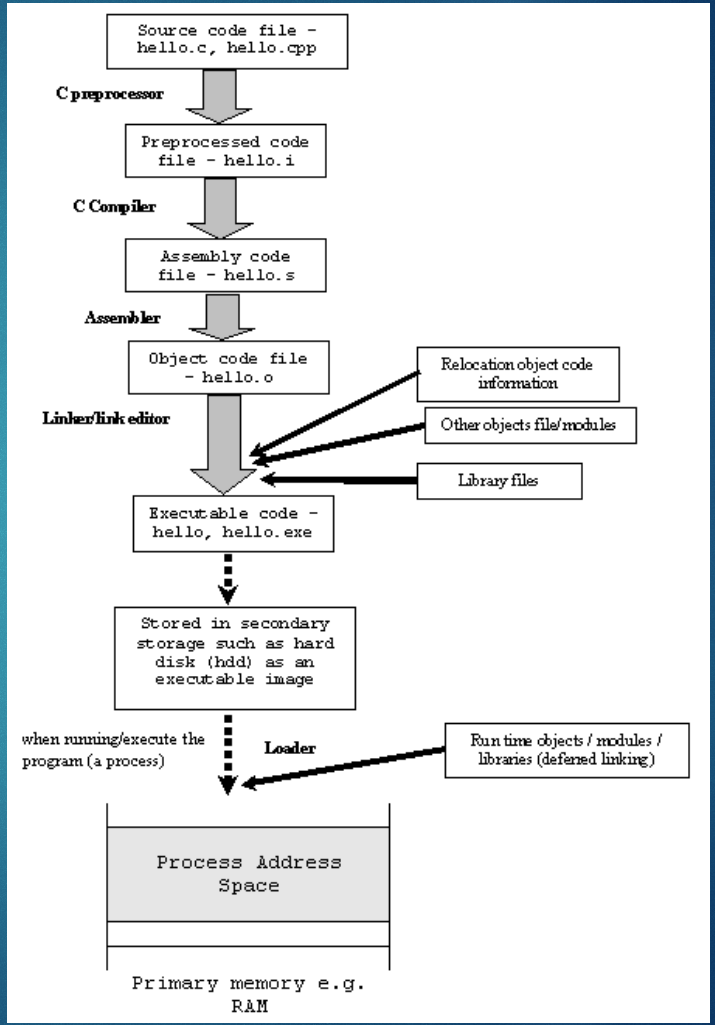
Compiler creates object code and stores it on disk.

Linker links the object code with the libraries

Loader puts program in memory.

CPU takes each instruction and executes it, possibly storing new data values as the program executes.


samp.c
©2004 HowStuffWorks
#include <stdio.h>
int main()
{
printf("Hello!\n");
return 0;
}
samp.c is your C Program. You type samp.c into a text file using a standard text editor. It is human-readable.
C compiler
You type:
gcc samp.c-o samp.exe
to compile samp.c into samp.exe using the gcc compiler.
samp.exe
10110101001011010
10100100100100010
10101001010101110
01010010010110101
11101010100111001
10101001010111110
10101101101001001
10101000111101011
The C compiler takes samp.c as input and turns it into a machine-readable executable. The computer "runs" or "executes" this executable.


Source code file - hello.c, hello.cpp
C preprocessor
Preprocessed code file - hello.i
C Compiler
Assembly code file - hello.s
Assembler
Object code file - hello.o
Relocation object code information
Other objects file/modules
Library files
Linker/link editor
Executable code - hello, hello.exe
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM

Compilers vs. Interpreters

- Compilers
  - Compile several machine instructions into short sequences to simulate the activity requested by a single high-level primitive
  - Produce a machine-language copy of a program that would be executed later

Compilers vs. Interpreters

- Interpreters
  - Execute the instructions as they were translated
  - An interpreter reads a code statement, converts it to one or more machine language instructions, and then executes those machine language instructions.
  - It does this all before moving to the next code statement in the program.
  - Each time the source program runs, the interpreter translates and executes it, statement by statement.

Thanks