EE101 (2017/2018)
C Programming & Software Engineering

Dr. Rui Lin and Dr. Mark Leach

Email: rui.lin/mark.leach@xjtlu.edu.cn

Rooms: EE512/EE510

Module Information

- Credit Value: 5 Credits

- How to Study :
  - Lectures
  - Lab Sessions
  - Private Study

- Assessments: **Continuous** – including Laboratory Exercises and Final Project
  - 3 Assignments (65% - 15, 15, 35% each)
  - 1 Final Project (35%)

Class Rules

- 1. Attend all your lectures and lab sessions (Attendance will be recorded through ICE).

- 2. Submit all of your course work on ICE (e-copy).

- 3. Observe the deadlines for your course work and our university policy applied for late submission.

- 4. Collusion and plagiarism are absolutely forbidden. University policy applied once caught.

Other Rules

- If you feel there is a large error in the marking of your coursework you have 1 week from the release of the marks to raise the issue….Module leaders decision is final!

- Deadlines are strict.

- No extensions without mitigating circumstance applications.

- Attend Lab sessions! (Attendance will be recorded through ICE).

Please Note The Following:

- 1. Students submitting the same or close to the same report for the assessment will be awarded "ZERO".

- 2. If we have doubts about the submitted assessments, students will be interviewed.

- 3. Cases of serious cheating will be reported to the "Academic Misconduct Working Group".

A Computer System in Office
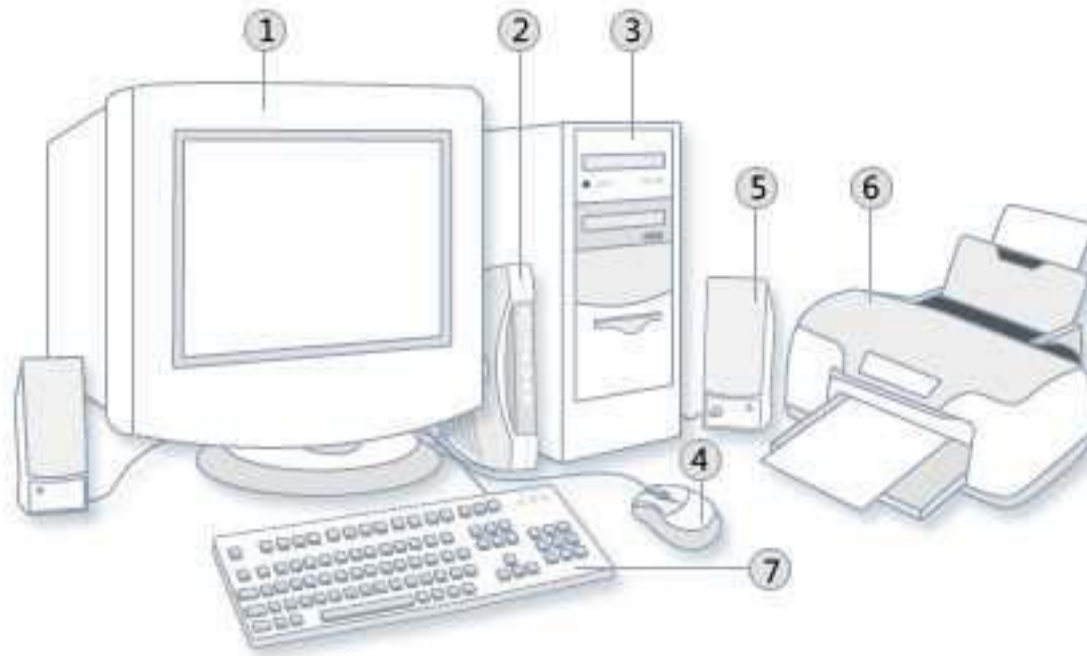
① Monitor   ③ System unit   ⑤ Speaker   ⑦ Keyboard

② Modem   ④ Mouse   ⑥ Printer

Elements of a Computer System

- The elements of a computer system include two major categories: hardware and software

- Hardware is the equipment used to perform the necessary computations (central processing unit, monitor, hard drives, memory card, keyboard, mouse, and etc.)

- Software consists of the programs that enable us to solve problems with a computer (lists of instructions to perform)

Computer Software

- Operating System is a collection of special computer programs that control the interaction of the user and the computer hardware and manage all computer resources (UNIX, Linux, DOS, Windows, MAC OS)

- Application Software is developed to assist computer users in accomplishing various tasks (Abaqus, Adobe Acrobat, AutoCAD, FireFox, LabView, MATLAB, Microsoft Office, PhotoShop)

Computer Information System

Data Input	→	Data Processing	→	Data Output

Keyboard, Mouse,

Camera, Scanner

CPU

Printer, Monitor

Speaker
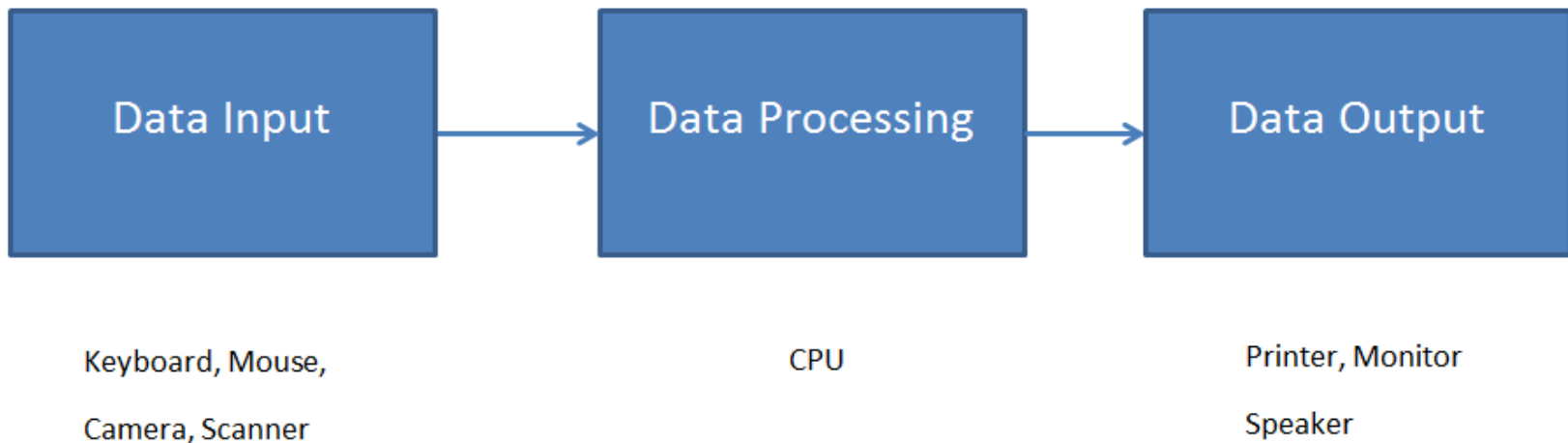
What is Programming?

- Computers are really dumb machines because they do what they are told to do.

- To solve a problem using a computer, you must express the solution to the problem in terms of the instructions of the particular computer

- Programming means telling a computer to do something as we wish.

Why Do We Learn Programming?

- Computer programming can become a career in itself. Learning how to program is the first step toward computer engineering, systems analysis, or database administration.

- Learning about programming will help you understand computers better in general.

- Programming can be very interesting and rewarding, as a hobby.

Software Engineering

- Preparedness ensures success, unpreparedness spells failure.

- 凡事预则立，不预则废 - 《礼记.中庸》

- Software development is more than programming, complex systems must be designed from the top down

Software Development Lifecycle

- 1. Specify the problem requirements

- 2. Analyze the problem

- 3. Design the algorithm to solve the problem

- 4. Implement the algorithm

- 5. Test and verify the completed program

- 6. Deliver to the customer

- 7. Maintain and update the program

Problem Specification

Specifying the problem requirement allows you to:

- 1. State the problem clearly and unambiguously.
- 2. Gain a clear understanding of what is required for its solution.
- 3. Eliminate unimportant aspects

- To achieve this goal you need more information from the person who posed the problem

Analysis

Analysing the problem involves identifying the following:

- 1. Inputs – the data you have to work with
- 2. Outputs – the desired results
- 3. Additional requirements or constraints on the solution.

Algorithms Design

Designing the algorithm:

- 1. Develop a list of steps called an algorithm to solve the problem.
- 2. Verify that the algorithm solves the problem as intended.

- N.B: Writing the algorithm is often the most difficult part of the problem solving process. Coding the algorithm is simple once you know the programming language syntax. Do not attempt to solve every detail of the problem at the beginning.

Implementation

Implementing the algorithm

- 1. Involves writing the algorithm as a program

- 2. You must convert each step of the algorithm into one or more statements in a programming language

Testing and Verification

- 1. Test the completed program to verify that the program works as desired. Do NOT rely on just one test case.

- 2. Run the program several times using different sets of data, to make sure it works correctly for every situation provided in the algorithm.

Maintenance

Maintaining and Updating

- 1. Improve a program by removing the previously undetected errors and keep it up-to-date as government regulations or company policies change.

- 2. Many organizations maintain a program for five years or more, often after the programmers who coded it have left or moved on to other positions.

Waterfall Lifecycle with feedback
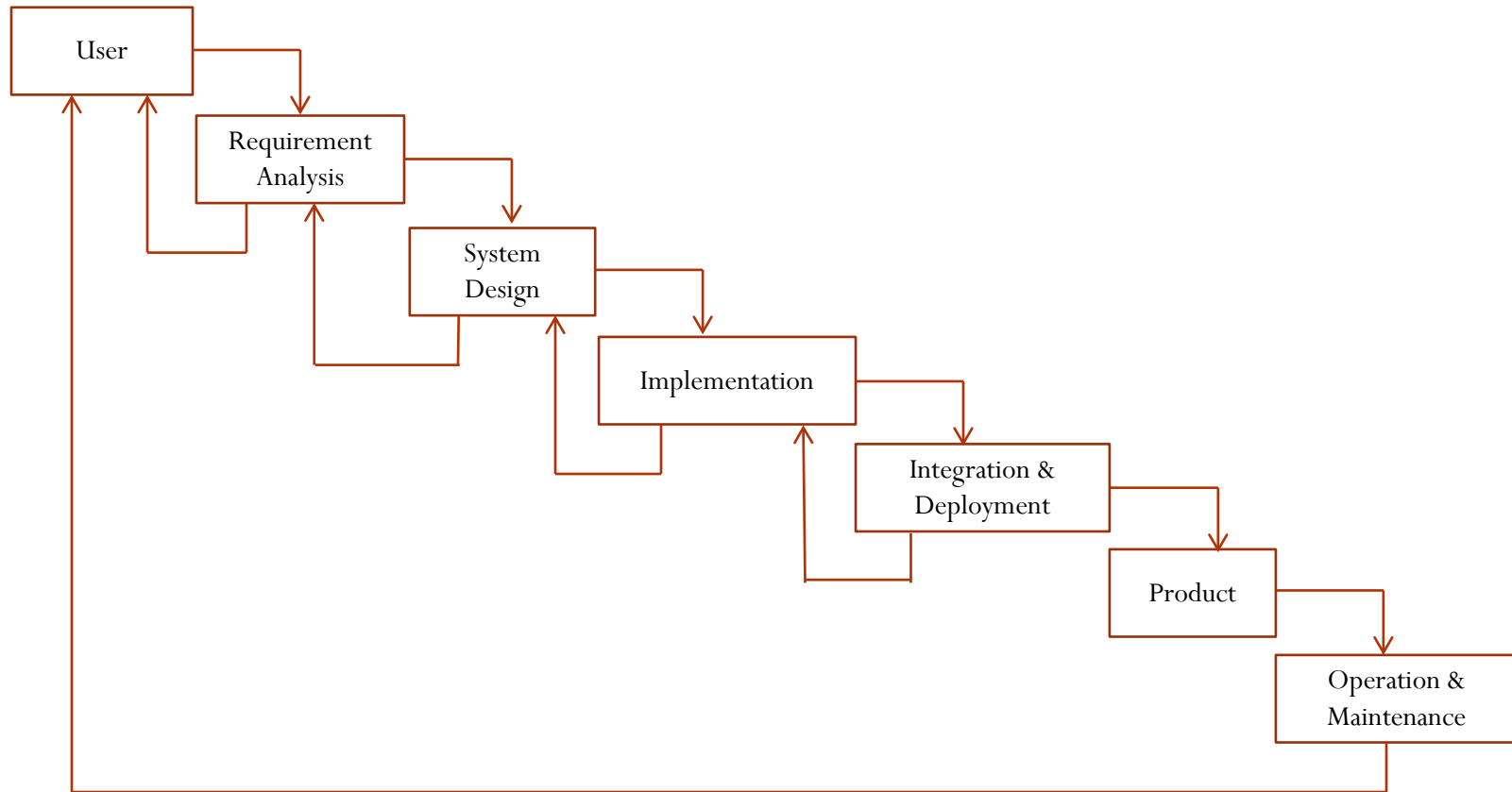
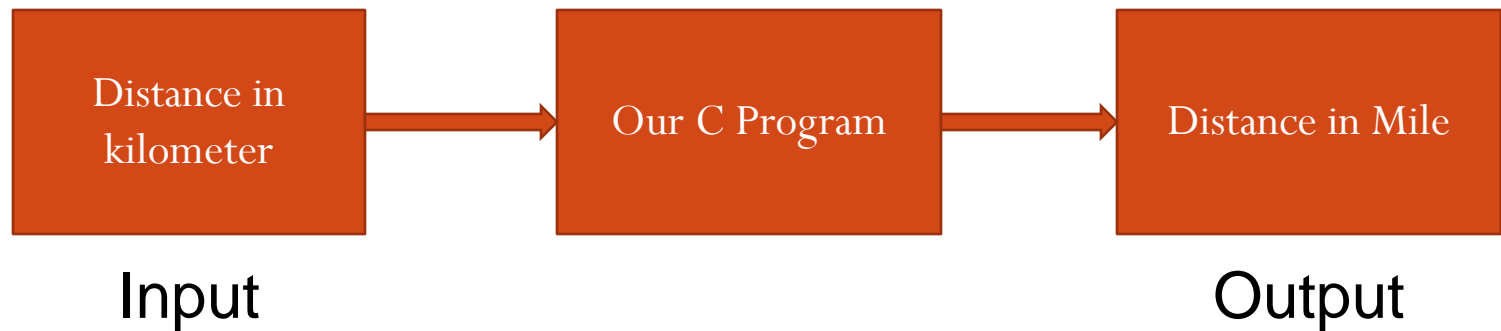# A Simple Example of SDL (1)

Step 1 - Problem Specification
To design a program that can realize the length unit conversion from kilometers to miles

A Simple Example of SDL (2)

Step 2 - Analysis

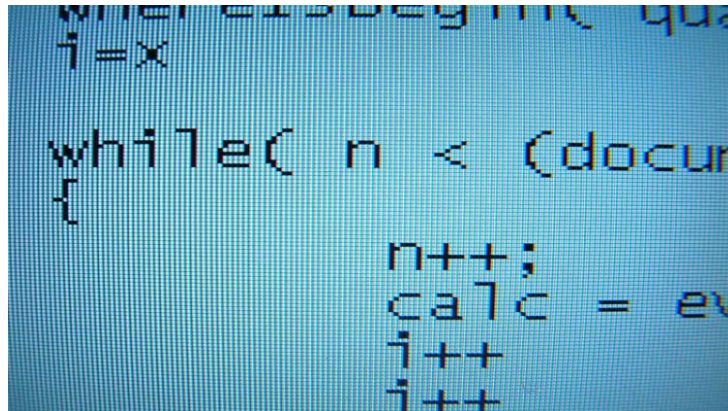Input   Output

A Simple Example of SDL (3)

Step 3 - Algorithm Design

$$1 \text{ mile} = 1.609 \text{ kilo-meters}$$

- *miles = kms / 1.609;*

Step 4 - Implementing Algorithm in *C*

Step 5 - Testing

Test the program using different data sets, for example, Xiao-Ming's every day workout in a week.

Step 7 - Maintenance

Xiao-Ming can run like Forest Gump crossing the continent of North America, and he is not satisfied with your program any more. It is not accurate enough.

1 mile = 1.609 344 kilometers

Low-Level versus High-Level Programming Languages

- **Low-Level Languages:** Assembly & Machine Languages – are specific to machine architectures; closer to machines than "problems".

- **High-Level Languages:** They are used to write programs that are independent of the machine architectures on which they will be executed. Examples of high-level languages are Fortran, Pascal, C, C++, Java, Ruby, and etc.

C Language

- Short History:

- - Designed by Dennis Ritchie at Bell Labs in 1972 called "C" from the 2nd letter from BCPL (Basic Combined Programming Language).

- - used to create the UNIX operating system.

- - Draft ANSI C standard in 1980

- - ANSI C standard adopted in 1989

- - ISO C (C90) standard adopted in 1990 (same as ANSI C)

- - Joint ANSI/ISO committee revised the standard (C99)

- - Bjame Stroustrup of Bell Labs develops C++ in 1980

The Evolution of Programming Languages

The Evolution of Programming Languages.
http://bluebones.net/evolution/
Source Data: wikipedia.org, Monkey Work: bakert@gmail.com
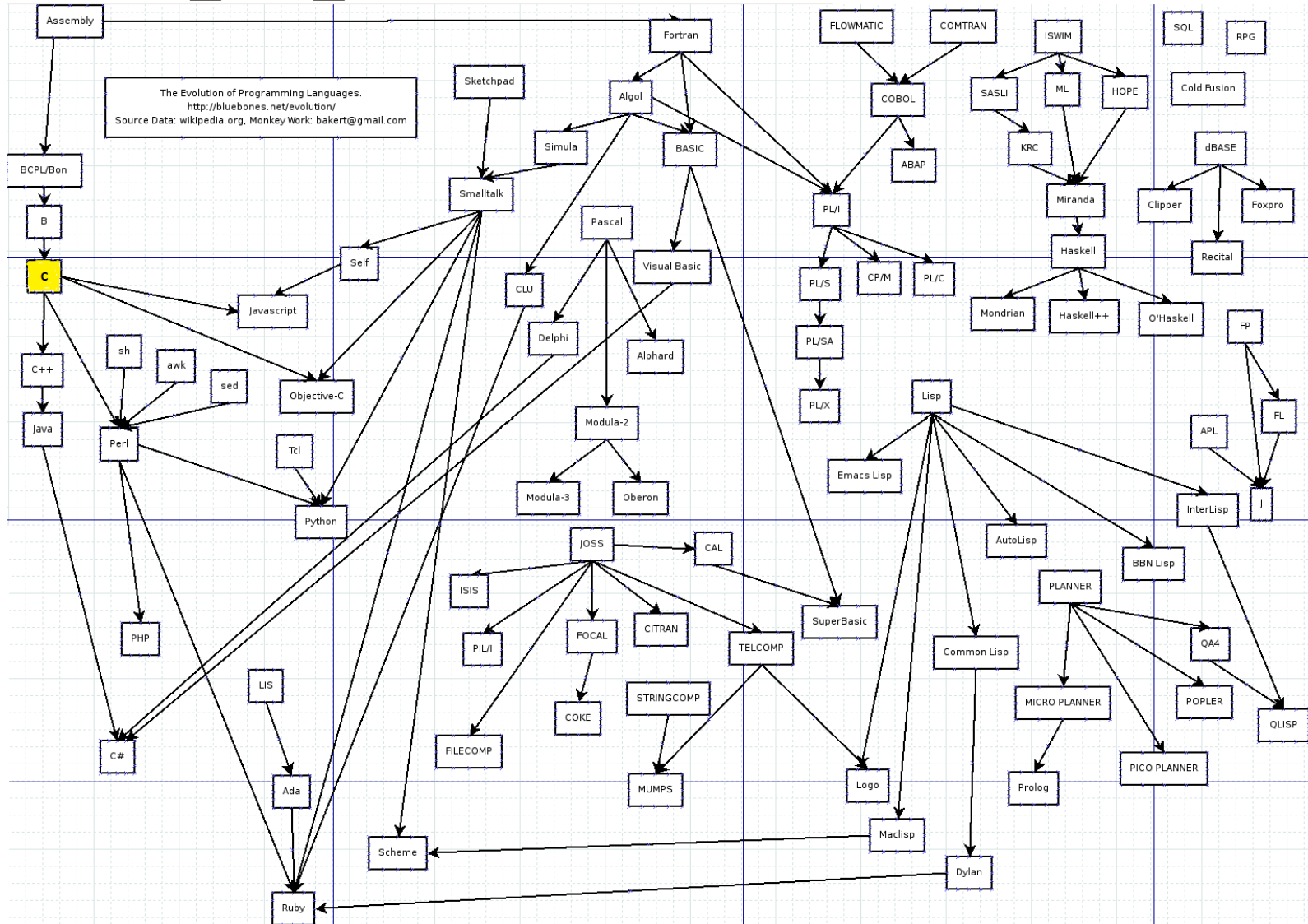
Strengths of C Language

Efficiency:

- Allows you to fine-tune your programs for maximum speed or most efficient use of memory.

Portability:

- Programs written in C can be recompiled and run on various computer systems (ANSI Standard) and microcontrollers.

Powerful and Flexible

- 1. Used by game programmers and scientists
- 2. The UNIX operating system is written in C.
- 3. Even some languages compilers are written in C

Weaknesses of C Language

- C programs can be **error-prone**

- C programs can be **difficult to understand**.

- C programs can be **difficult to modify**.

What is a C Program?

- A C program consists of one or more files, known as source files, which are ordinary text files.

- There are two types of source files:

- 1. Interface or header files – there usually end in *.h*

- 2. C source files – these have the extension *.c*

Example: a C program

```c
/*
 * Converts distances from miles to kilometers.
 */
#include <stdio.h>                  /* printf, scanf definitions  */
#define KMS_PER_MILE 1.609  /* conversion constant        */

int
main(void)
{
        double miles,   /* distance in miles
               kms;     /* equivalent distance in kilometers */

        /* Get the distance in miles. */
        printf("Enter the distance in miles> ");
        scanf("%lf", &miles);

        /* Convert the distance to kilometers. */
        kms = KMS_PER_MILE * miles;

        /* Display the distance in kilometers. */
        printf("That equals %f kilometers.\n", kms);

        return (0);
}
```

Labels (annotations):
- comment
- standard header file
- preprocessor directive
- constant
- reserved word
- variable
- standard identifier
- comment
- special symbol
- reserved word
- punctuation
- special symbol

Building the Executable File

Compiling:

- Converting C statements into binary (machine) code   the *.obj* (object) files

Linking:

- Stitching together various object *(.obj)* files with existing library files (*.lib*) needed to generate a self-contained executable file (*.exe*)

Reference Books

- 1. Brian W. Kernighan & Dennis M. Ritchie, "The C Programming Language", Prentice Hall, 1988

- 2. Stephen Prata, "C Primer Plus", Fourth Edition, SAMS Publishing, 2002

- 3. Jeri R. Hanly & Elliot B. Koffman, "Problem Solving & Program Design in C", Third Edition, Addison Wesley, 2002

- 4. Ian Sommerville, "Software Engineering", Seventh Edition, Pearson Education Limited, 2004

What's Next?

- Practice in lab sessions!

- Learn to use a compiler

- Enter the C code from the examples to familiarize yourselves with the **compilation procedure**.

- In next week we will start learning the **Syntax** of C programing language.