

# CPT109: C Programming & Software Engineering I

## Introductory Lecture

Dr. Xiaohui Zhu & Soon Phei Tin

Office: SD535 & SD531

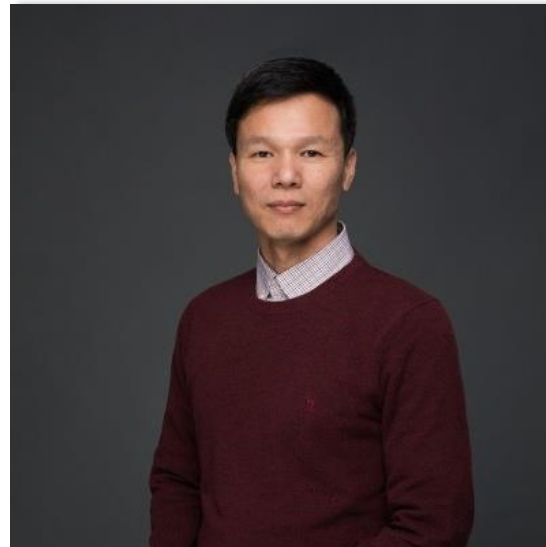
Email: [Xiaohui.zhu@xjtlu.edu.cn](mailto:Xiaohui.zhu@xjtlu.edu.cn),  
[Soon.Tin@xjtlu.edu.cn](mailto:Soon.Tin@xjtlu.edu.cn)

# Module Teachers

- Xiaohui Zhu



- Soon Phei Tin



# Today

---

In this lecture we will be looking at the following:

- Module Content
- Understanding what programming is
- Software Development Lifecycle

# Module Aim

Despite the popularity of newer languages such as C++ and Java the C language remains a core skill in the software business.

C is one of the most popular languages for programming embedded systems that are found in automobiles, DVD players, cameras and many other modern appliances.

This module aims to enable students to:

- Learn and use the C programming language
- Use the C language to solve real engineering problems
- Acquire fundamental software development skills covering program design, coding and testing.

# Why Do We Learn To Program?

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

Problem solving skills

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

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

# Module Learning Outcomes

Students completing the module successfully should be able to:

- A. demonstrate knowledge and understanding of the basic principles of the C programming language;
- B. demonstrate knowledge and understanding of the basic role and function of hardware and software components of a computer;
- C. demonstrate knowledge and understanding of the software development process;
- D. design, code, debug, test and document computer programs written in the ANSI C language to meet requirements supplied in a specification;
- E. design modular programs following the top-down function-oriented approach.
- F. analyse understand and modify existing code written in C;

# Module Information

- Credit Value: 5 Credits
- Module Delivery:
  - Lectures 2 Hours per week
  - Laboratory Sessions 2 Hours per week
  - Private Study
- Module Assessment
  - 2 Individual Assignments (15% and 50%)
  - 1 Group Project (35%)

# Class Rules

1. Attend all of your scheduled lectures  
(Attendance is recorded)
2. Submit all coursework on LMO (ecopy)
3. Complete all continuous online assesement on LMO
4. Coursework deadlines are strict (unless there are mitigating circumstances). University late submission policy will be applied.
5. Collusion and plagiarism are absolutely forbidden  
(Academic misconduct will be reported)

If we have doubts about submitted assessments you may interviewed



# Other Rules

- If you feel there has been 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!**
- **Attend lab sessions!**

# Resources

- In the resources section on the module learning mall page you will find various resources:
  - Quincy 2005 compiler installer
  - Books
  - Web resources
  - Assignment related help

# Reference Books

- Brian W. Kernighan & Dennis M. Ritchie, “The C Programming Language”, Prentice Hall, 1988
- Stephen Prata, “C Primer Plus”, Fourth Edition, SAMS Publishing, 2002
- Jeri R. Hanly & Elliot B. Koffman, “Problem Solving & Program Design in C”, Third Edition, Addison Wesley, 2002
- Ian Sommerville, “Software Engineering”, Seventh Edition, Pearson Education Limited, 2004

# Compilers

- The software you use to create your C program is called a **compiler**.
- Your actual C source code is simply a **text file** with the extension **.c**
- You can use any C compiler you like with **C90**:
  - Visual Studio 2013
  - Codeblocks
  - **Quincy 2005**
  - Or just write the code in Windows notepad (not recommended) and use the command line

# Compilers – **MAC users**

- There are a number of C compilers available for MAC users, you should be able to download **Xcode** from the app store.
- Install a Windows environment so that you can install Windows-based applications.

# Issues Arising

- If you are having problems and want additional help please contact me, I will make time for you.
- If you want to raise concerns about the module, you have options:
  - Hopefully, you would first approach me (I don't bite).
  - You can contact the year-leader [sat.year2@xjtlu.edu.cn](mailto:sat.year2@xjtlu.edu.cn) (this includes e-learning issues).
  - Contact your year representative to report to Student Staff Liaison Committee (SSLC).
  - Speak to the head of department Dr. Hai-Ning Liang ([haining.liang@xjtlu.edu.cn](mailto:haining.liang@xjtlu.edu.cn)).
  - Speak to the Dean of the School Prof. Enggee Lim ([enggee.lim@xjtlu.edu.cn](mailto:enggee.lim@xjtlu.edu.cn)).

And so...



Let's begin...

# Typical Computer System





# Computer Systems

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

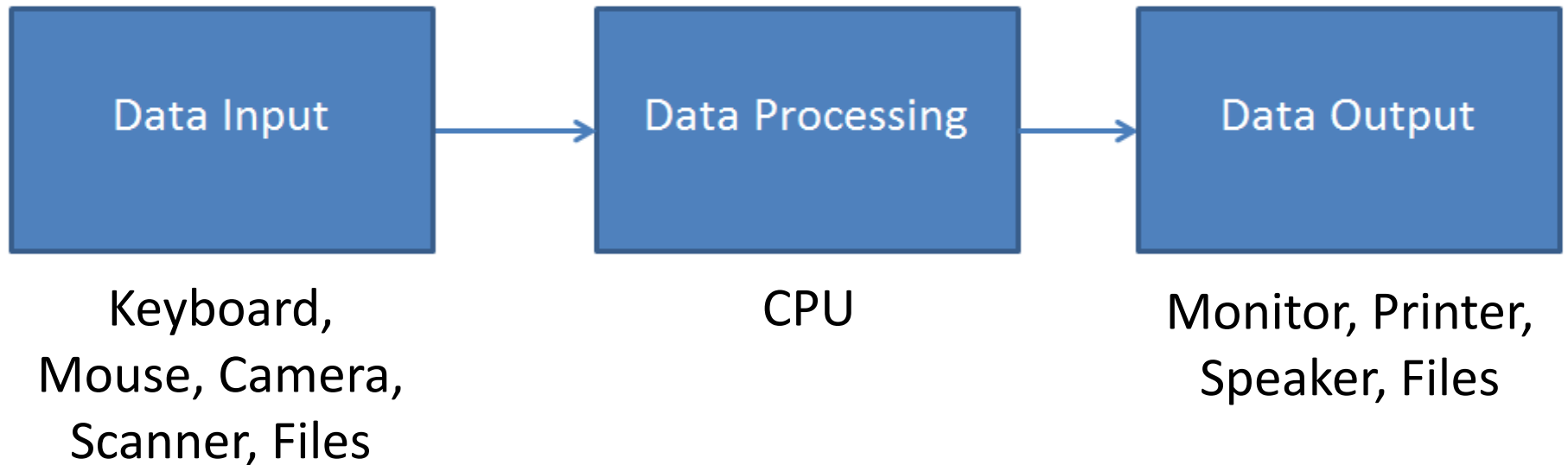
- **Hardware** is the equipment used to perform the necessary computations (central processing unit (CPU), monitor, hard drives, memory card, keyboard, mouse etc.)
- **Software** consists of the programs that enable us to solve problems with a computer (lists of instructions to perform)

# 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)

# Computers and Information



# Numbers in the Computer

Everything a computer does is based on numbers.

- In a computer all numbers are in binary, base 2.
- Convenient for computer because?...



- not for us – we like decimal, base 10 because?

# Bits, Bytes...and Nibbles

## Bits:

- bit – derived from the words **b**inary dig**it**
- smallest unit of data/information in a computer
- a bit is binary information:
  - Value 0 – OFF
  - Value 1 – ON

## Bytes:

- A byte is a basic unit of measurement of storage
- A byte is an ordered collection/pattern of bits
- A byte consists of 8 bits in modern computer systems  
e.g. 01100100 or 10011010

**Nibble:** Half a byte or 4 bits

# Working with number bases

Any number can be represented using any base

- Decimal system (base 10) uses 10 digits 0-9
- Binary system (base 2) uses 2 digits 0 and 1
- Octal system (base 8) uses 8 digits 0-7
- Hexadecimal system (base 16) uses ???

# Working with number bases

Any numbers can be represented using any base

- Decimal system (base 10) uses 10 digits 0-9
- Binary system (base 2) uses 2 digits 0 and 1
- Octal system (base 8) uses 8 digits 0-7
- Hexadecimal system (base 16) uses 16 digits **0-9 and A-F**

**Hexadecimal is often used by programmers as it directly scales into binary unlike decimal.**

# Numbers in the Computer

Dec	Binary	Hex	Dec	Binary	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Hex easier to read than binary, one byte = 8 bits or 2 hex digits

**Decimal 75 = 01001011 = 4B**

You should be able to convert between the number bases.



# Number System of Base $b$

In general, if  $b$  is the base, we write in the number system of base  $b$  by expressing it in the form:

$$a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \dots + a_0 b^0$$

where  $a$  are the digits used in the number system of base  $b$  and  $n$  can be any natural number

## Examples:

In decimal ( $b=10$ ):

$$4327 = (4 \cdot 10^3) + (3 \cdot 10^2) + (2 \cdot 10^1) + (7 \cdot 10^0)$$

In binary ( $b=2$ )

$$011 - (3 \text{ in decimal}) = (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0)$$

In hexadecimal ( $b=16$ )

$$1E5 - 485 \text{ in decimal} = (1 \cdot 16^2) + (14 \cdot 16^1) + (5 \cdot 16^0)$$

# What is Programming

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

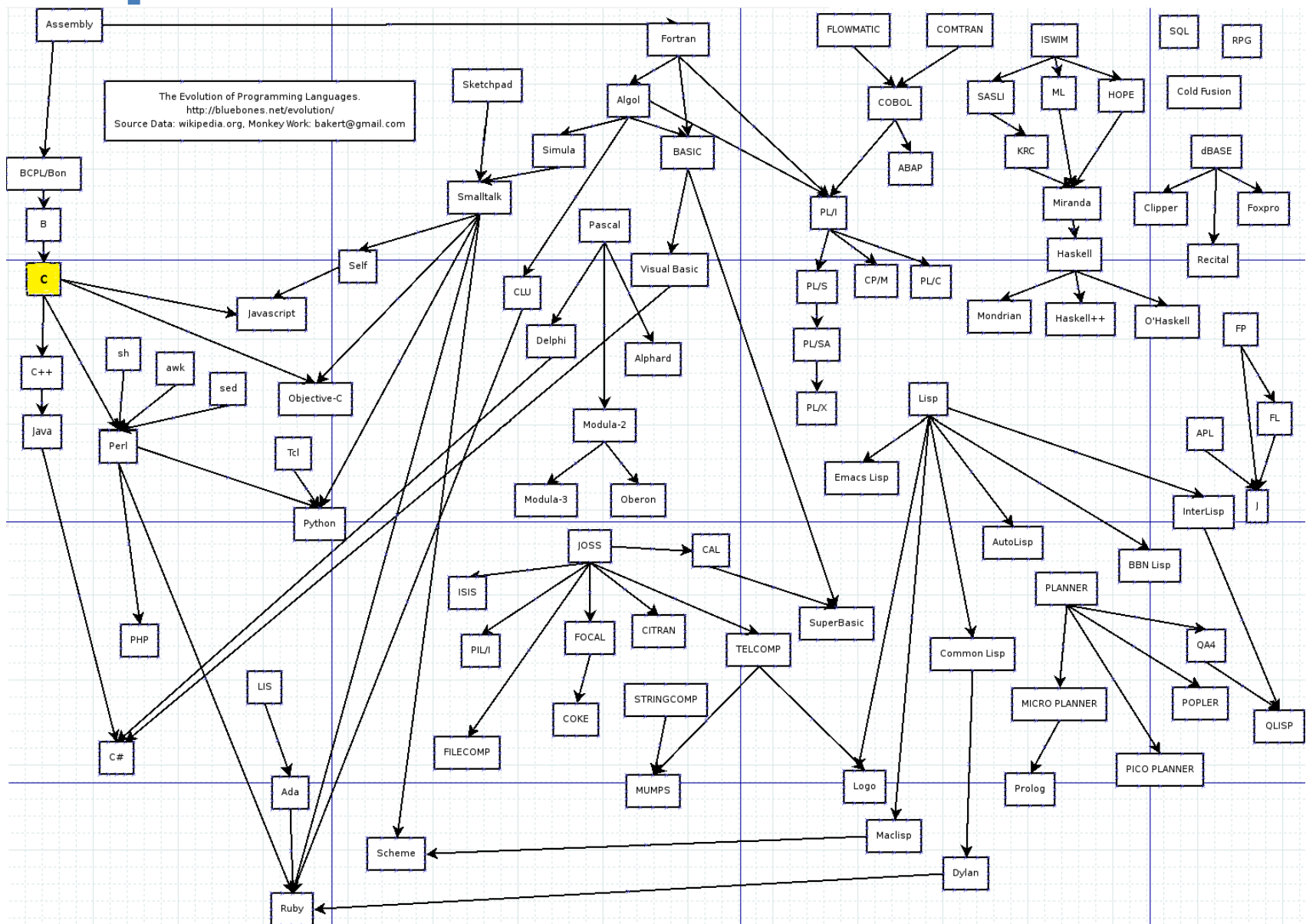
To solve a problem using a computer, you must express the solution to the problem in terms of the instructions of the particular computer

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

# 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)
- Revised standard in 2011 (C11)
- Revised standard again in 2017 (C17)
- Bjarne Stroustrup of Bell Labs developed C++ in 1980

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



# Programming Evolution

## Assembly Language

```
DATA SEGMENT
    NUM1 DB 05
    NUM2 DB 06
    RESULT DB ?
ENDS

CODE SEGMENT
    ASSUME DS:DATA CS:CODE
START:
    MOV AX, DATA
    MOV DS, AX

    MOV AL, NUM1
    ADD AL, NUM2

    MOV RESULT, AL

    MOV AH, 0
    AAA

    MOV AH, 4CH
    INT 21H

ENDS
END START
```

## C Programming

```
main()
{
    int a=6, b=6, result;
    result=a+b;
}
```

# 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 – usually end in .h
  2. C source files – these have the extension .c

# Strengths of C

## 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

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

# Weaknesses of C

- C programs can be error-prone
- C programs can be difficult to understand.
- C programs can be difficult to modify.



# Example C Program

```
/*
 * 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);
}
```

Diagram labels and arrows:

- preprocessor directive** points to `#include <stdio.h>` and `#define KMS_PER_MILE 1.609`.
- constant** points to `1.609`.
- reserved word** points to `int`, `main`, and `{`.
- variable** points to `double` and `kms`.
- comment** points to `/* printf, scanf definitions */`, `/* conversion constant */`, and `/* Get the distance in miles. */`.
- standard identifier** points to `printf` and `scanf`.
- special symbol** points to `*` in `KMS_PER_MILE * miles` and `*` in `/*`.
- reserved word** points to `return`.
- punctuation** points to `(`, `)`, and `;`.

# Building an Executable

## Compiling:

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

## Linking:

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

# 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.

# Algorithm 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: 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

## 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.

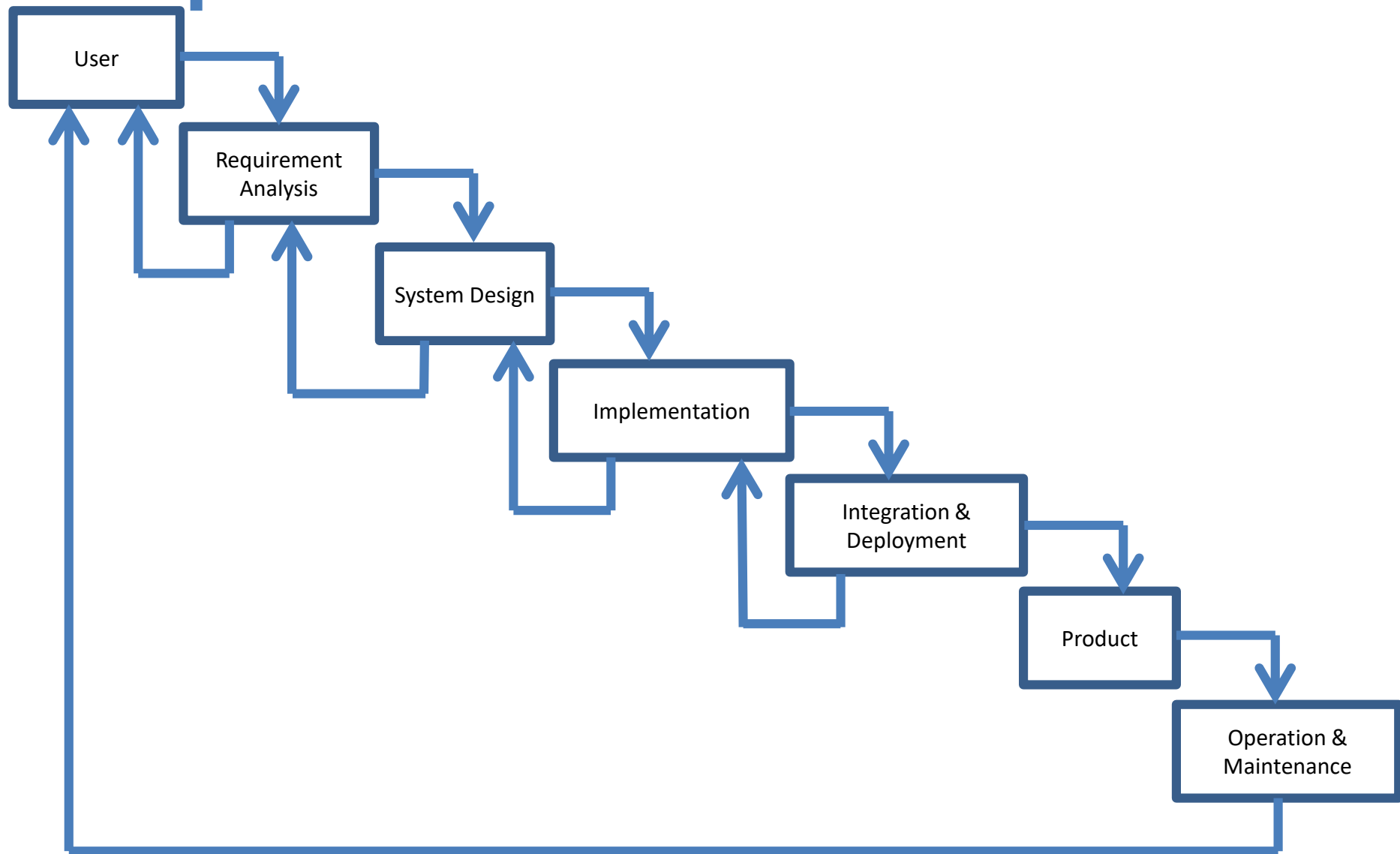


# Maintenance

## Maintenance and Updating

- 1. Improve a program by removing previously undetected errors and keep it up-to-date with changing government regulations or company policies.
- 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 + Feedback



# Simple SDL Example (1)

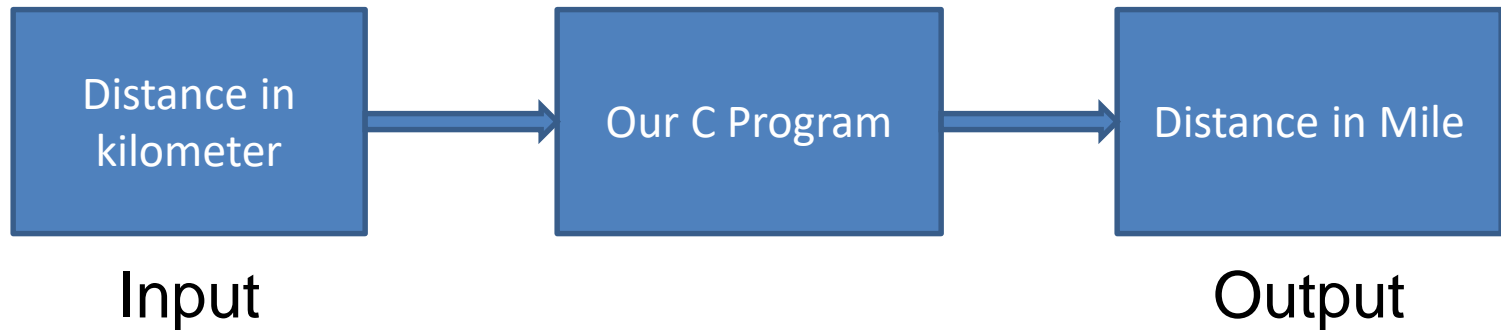


## Step 1 - Problem Specification

To design a program that can realize the length unit conversion from kilometers to miles

# Simple SDL Example (2)

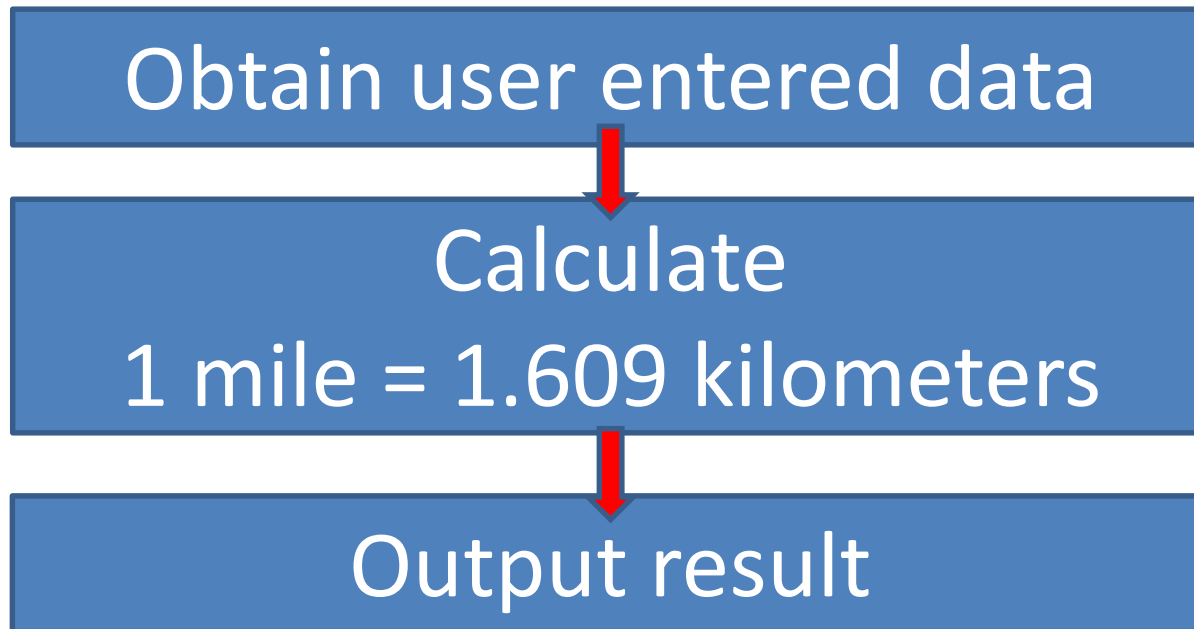
## Step 2 - Analysis



Determine types of variables (will discuss these next week)

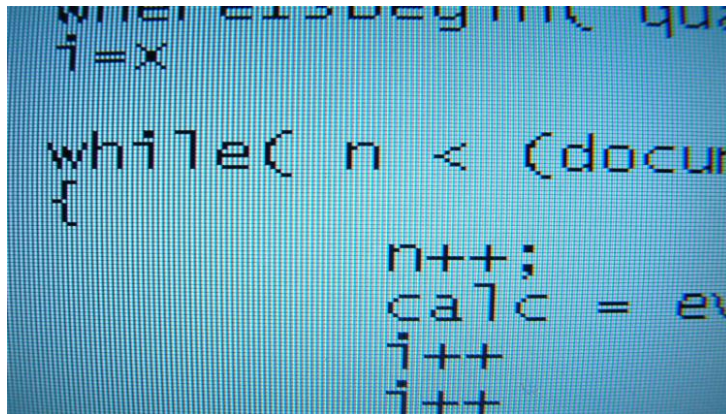
# Simple SDL Example (3)

## Step 3 – Algorithm Design



# Simple SDL Example (4)

## Step 4 – Implementation



```
while( n < (document  
{  
    n++;  
    calc = ev  
    i++;  
    i++;
```

## Step 5 – Testing

Test the program using different data sets, for example, 1km 50km 100km, 1.1km, a.

# Simple SDL Example (4)

## Step 7 – Maintenance

Requirement for more accuracy

1 mile = 1.609 344 kilometers

---

# **Your First Program!**

---



# Your first C program (1/3)

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf("Programming in C is fun.\n");
```

```
    return 0;
```

```
}
```

**Q.** What does this C program do?

# Your first C program (2/3)

This C program consists of two parts:

- A **preprocessor directive** (begins with #)
- The **main** function.

## Preprocessor directives

- Are commands that give instructions to the C preprocessor, whose job it is to modify the text of a program before it is compiled.
- Begin with the # character
- Two commonly used directives
  - #include
  - #define

# Your first C program (3/3)

The **main()** function:

- Every program has a main function
- C program execution begins with the main function

# 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 programming language.



**Thank you for your attention 😊**

**See you in the laboratory...**