

EEE102

C++ Programming and Software Engineering II

Lecture 1 Introduction

Dr. Rui Lin / Qing Liu

Rui.Lin / Qing.Liu@xjtlu.edu.cn

Room EE512 / EE516

Office hour: 2-4pm, Tuesday & Wednesday

/ Monday & Wednesday

Course Structure of EEE102

- **Allocation of Time:**

Lectures	Seminar	Tutorials	Lab/Practice	Private Study	Total
11			22	42	75

- **Assessment:**

4 assignments	15%, 15%, 35%, 35%
---------------	--------------------

Class Rules

- 1. Attend all your lectures and lab sessions.
- 2. Submit all your course work (e-copy) on ICE.
- 3. Observe the deadline for your course work, university policy applied for late submission.
- 4. Collusions and plagiarism are absolutely forbidden. University policy applied once caught.
 - Students submitting the same or close to the same report for the assessment will be awarded “ZERO” and reported to Registry for record.
- 5. If failed, resit will be a class room examination.

Course Materials

- Reference books:
 - H.M.Deitel and P.J.Deitel, “Small C++ How to program”, Prentice Hall, 2006.
 - S. Prata, “C++ Primer Plus, 5th ed.”, SAMS, 2005.
 - B. Eckel, “Thinking in C++, 2nd ed.”, Prentice Hall, 2002.
- Online resources:
 - The C++ Resources
 - <http://www.cplusplus.com/>
 - C++ tutorial for C users:
 - <http://www.4p8.com/eric.brasseur/cppcen.html>



What we will learn in this module?

- Software Engineering
- From C to C++
- Introduction to Classes and Objects
 - Functions, arrays and pointers with objects
- Advanced topics on classes
 - Class composition
 - Dynamic memory allocation
 - Operator overloading
 - Inheritance
 - Polymorphism
- Stream I/O in C++

Lecture 1 Introduction - Outline

- What is software engineering?
 - Software as an Engineering Product
 - The design model of software engineering – waterfall model
- Basic Principles for Software Design
 - Abstraction
 - Modularity
 - Information hiding
- C++ Programming Language and Object Oriented Programming
 - History of C and C++
 - What is object oriented programming (OOP)
 - A simple C++ program
 - Compilation process
 - Typical Structure of the Source Code

1.1 Introduction – Software Engineering

- SOFTWARE

- Basically, A set of instructions to a computer to perform specified computation, operation or control.
- A piece of software may consist of a number of programme modules.

- SOFTWARE ENGINEERING

- The establishment and use of sound engineering principles to cost effectively design and produce software that is reliable and works efficiently on real machines.

Introduction – Software Engineering

- Software as an Engineering Product ---- Business

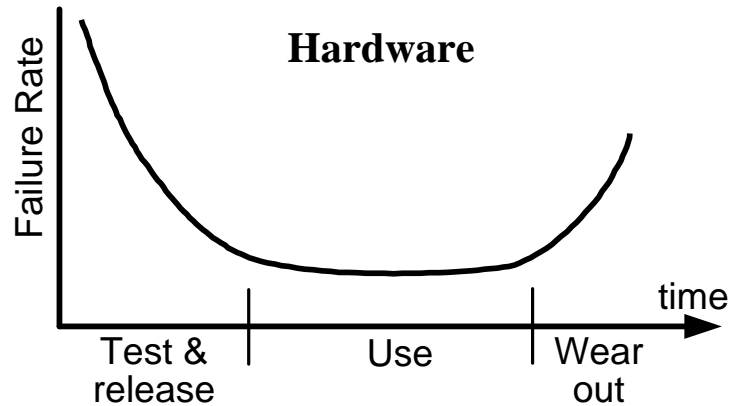
Need to use a set of tools, methods and techniques for the design of a software product.

- Pre-planned (specifications)
- Designed (away from computer)
- Constructed (coding)
- Tested
- User manual for information and maintenance

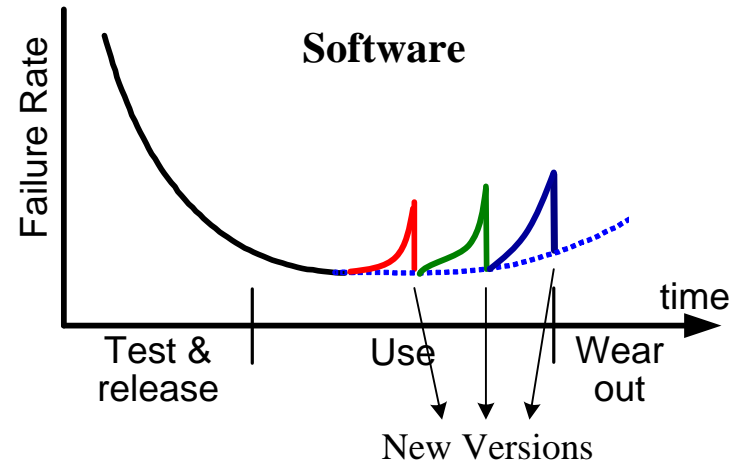
Introduction – Software Engineering

- Important to Engineering Students ?
 - Simulation and Modelling
 - Final year project, research project in future for optimum design and simulation of complex systems.
 - Design of embedded systems
 - Multimedia systems or control systems.
 - Development of Commercial Software
 - Where there is a computer, there must be software
 - Job aspects ---- Lots of advertisement requires knowledge in C/C++

Lifecycle and characteristics of engineered products



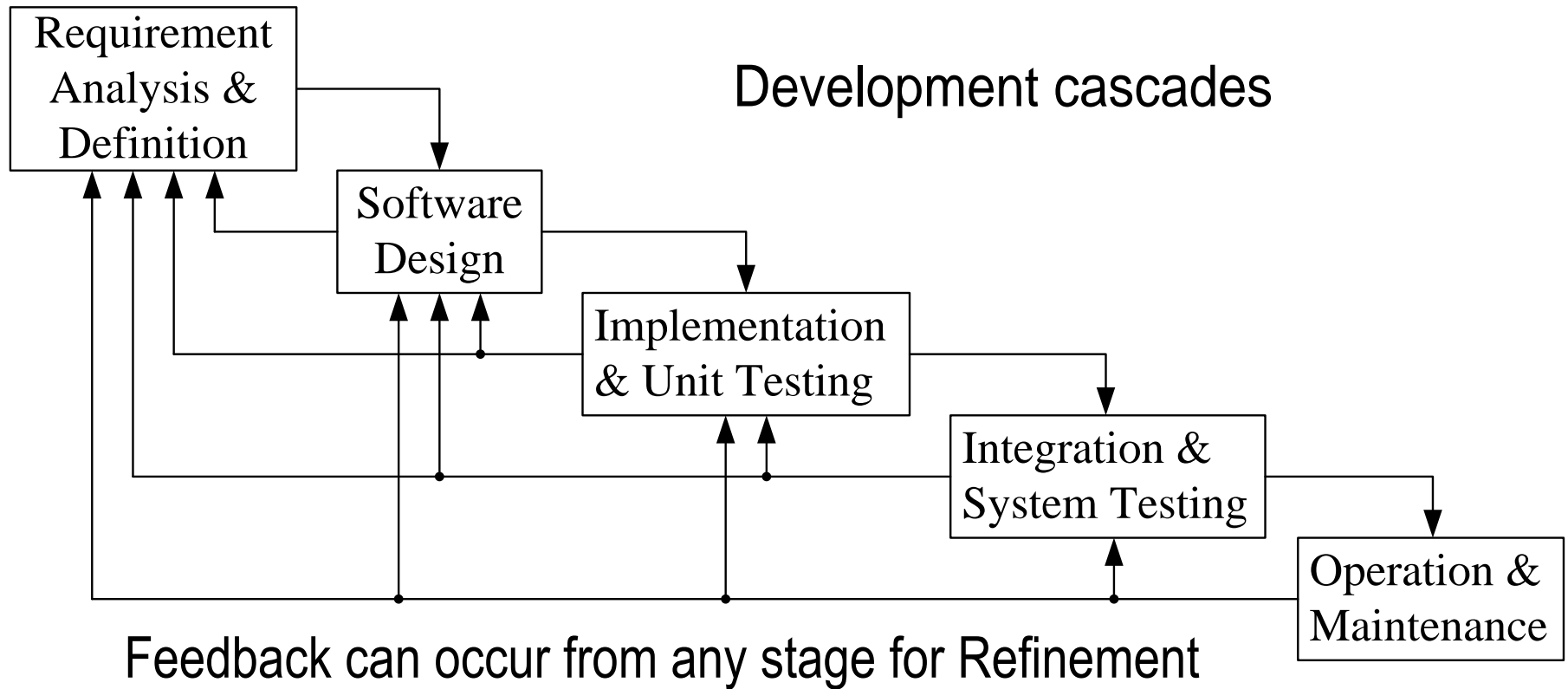
- Product
- Manufactured
- Physical system
- Wear out
- Spare part to replace
- Instructions for use



- Product
- Developed
- Logical system
- Does not wear out
- No spare part to replace
- Instructions for installation and use



1.2 The Waterfall model of software design



2. Basic Principles for Software Design

- Abstraction
 - Important for structured programming
- Modularity
 - For efficient management and test
- Information hiding (coupling and cohesion)
 - To reduce the interference between modules

2.1 Abstraction

When a program becomes big, it is more difficult to handle it.

- Problem:
 - It is important to understand the whole problem for which you are going to produce a software package; However, human brains can only understand part of a complicated system at one time.
- Solution:
 - Use of Structured programming where we divide the whole software package into smaller pieces (modules). Each time we concentrate on a specific part of the programme.
- How to divide?

Top Level of Abstraction

Always start from the customer's requirements -- Describe the major actions that a programme needs to perform, just forget all the details associated with each major job.

- Example:
 - A programme will **read** information of all employees in an organisation from a disk file, **sort** the names in alphabetic order and **display** the information on the screen. It also **outputs** the sorted information into a disk file with another file name.

Second Level of Abstraction

Then progressing into the major tasks and produce a list

- Major Tasks:
 - Read in information from file
 - Sort names
 - Display information on monitor
 - Write sorted information to another file

Tips: Regard each of the tasks as a black box.

Lower Level of Abstraction

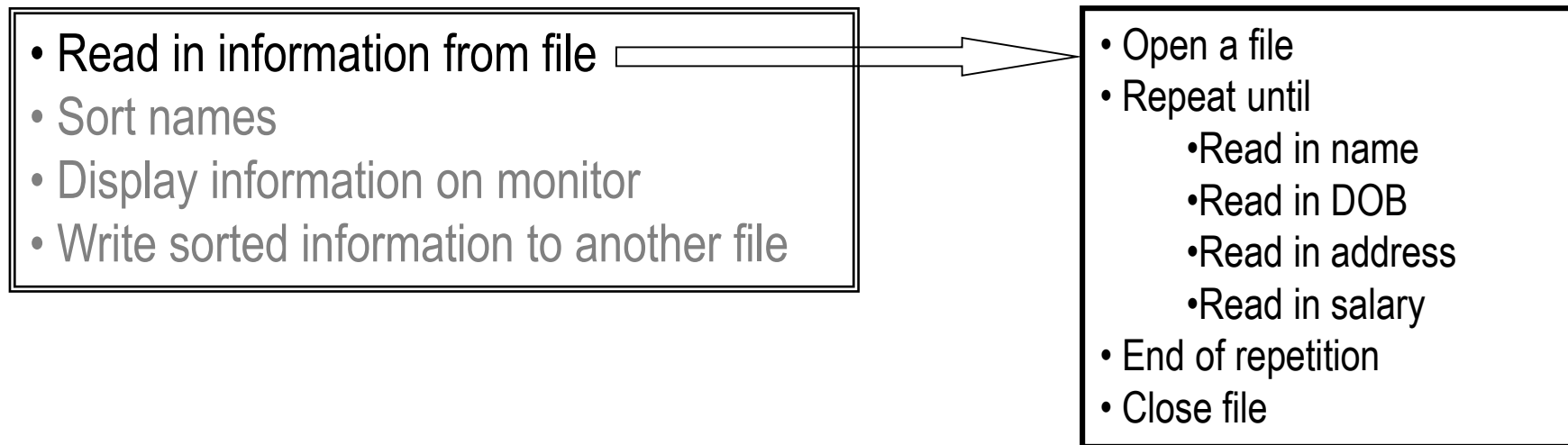
Progressing into more details that how a specific task should be done. You may need to break a major task into several smaller ones depending on the problem you are working on

- Task 1: **Read** in information from file
 - Open a file
 - Repeating the following for each employee
 - Read in name
 - Read in DOB
 - Read in address
 - Read in salary
 - Until information read for all staff
 - Close file

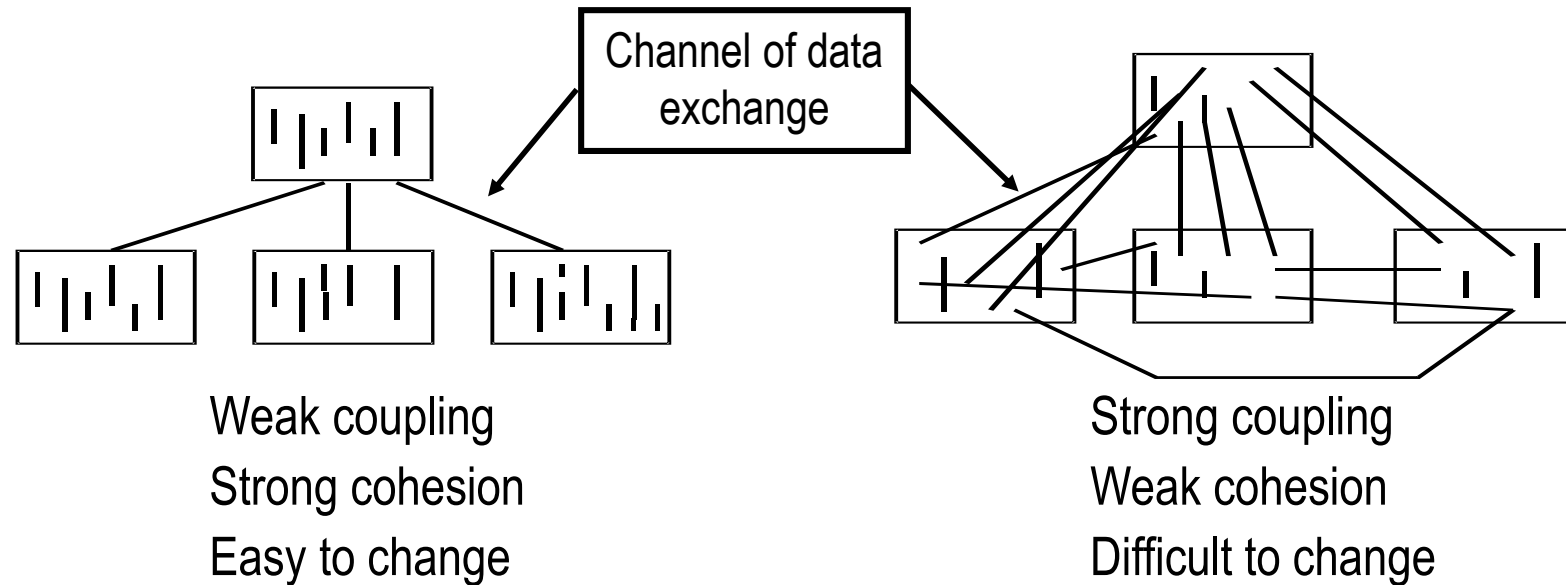
2.2 Modularity and Information Hiding

A programme (piece of software) consists of a number of sub-programmes (modules).

- ✓ Ideally, Each module perform only one simple task.
- ✓ Modules should be reusable when needed.
- ✓ Information contained in a module should not be accessed from another module. It can only be exchanged through module interface.



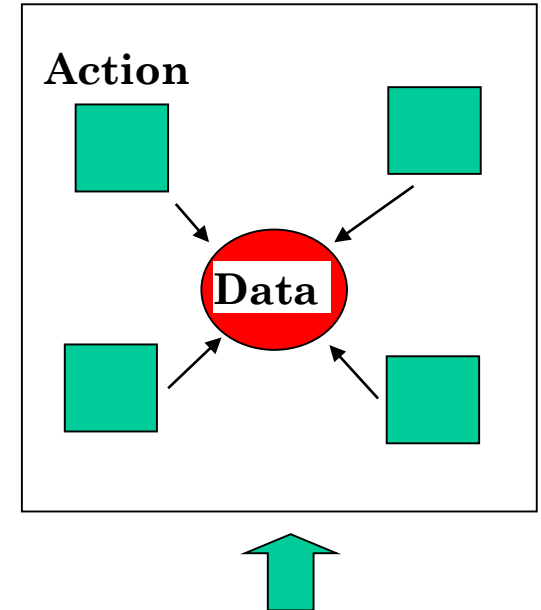
Module Content and Communication between Modules



- **Coupling:**
 - Is a measure of the amount of interaction between modules. Less is better.
- **Cohesion:**
 - Is a measure of the amount of interaction between action and information within a module. More is better

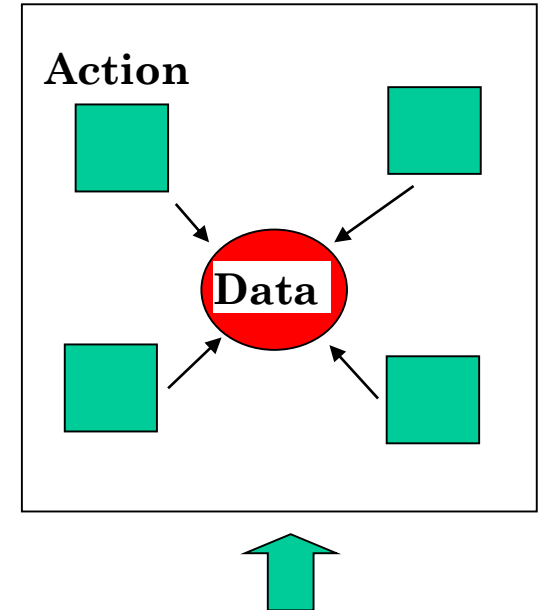
Types of cohesion

- Coincidental (undesirable):
- Logical: Example: Output text to screen for user
 Output line to printer
 Output data to file
- Temporal: Example: Clear screen
 Read data from a file
 Display on screen
- Communicational: Actions acting on common data are grouped together.
 Example: Convert and print the price in British pounds
- Functional (desirable):
 A module performs a well-defined action on a group of data.



Types of cohesion

- Coincidental (undesirable):
- Logical: Example: Output text to screen for user
 Output line to printer
 Output data to file
- Temporal: Example: Clear screen
 Read data from a file
 Display on screen
- Communicational: Actions acting on common data are grouped together.
 Example: Convert and print the price in British pounds
- Functional (desirable):
 A module performs a well-defined action on a group of data.



3. C++ Programming Language

- What is Programming?
- Why do we learn Programming?
- Programming languages
 - Low-level: Assembly & Machine Languages – are specific to machine architectures; closer to machines than “problems”
 - High-level: 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, C, C++ and Java.
- C and C++

3.1 History of C and C++

- C
 - Designed by Dennis Ritchie at Bell Labs 1972
 - 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)
- C++
 - Bjarne Stroustrup of Bell Labs develops C++ in 1979 (C with Classes).
 - In 1983, the name of the language was changed from C with Classes to C++
 - ISO/IEC 14882:1998 (C++98) published in 1998
 - A corrected version ISO/IEC 14882:2003 published in 2003.

3.2 C++ and Object Oriented Programming

- C++ is developed from C
 - Introduce object-oriented programming (OOP) features to C.
 - It offers classes, which provide the features commonly present in OOP languages: abstraction, encapsulation, modularity, inheritance, and polymorphism
- Object Oriented Programming
 - Object – a data structure consisting of data fields and methods together with their interactions – to design applications and computer programs.
 - Object Oriented Programming Languages – C++, JAVA, C#
 - Procedure Oriented Programming – C, BASIC

3.3 A simple program

The first program – print “Hello world” to the standard output

- C - Hello world

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

```
int main(void)
```

```
{
```

```
    printf("Hello world\n");
```

```
    return 0;
```

```
}
```

- C++ - Hello world

```
#include <iostream>
```

```
int main(void)
```

```
{
```

```
    std::cout<<"Hello world"<<std::endl;
```

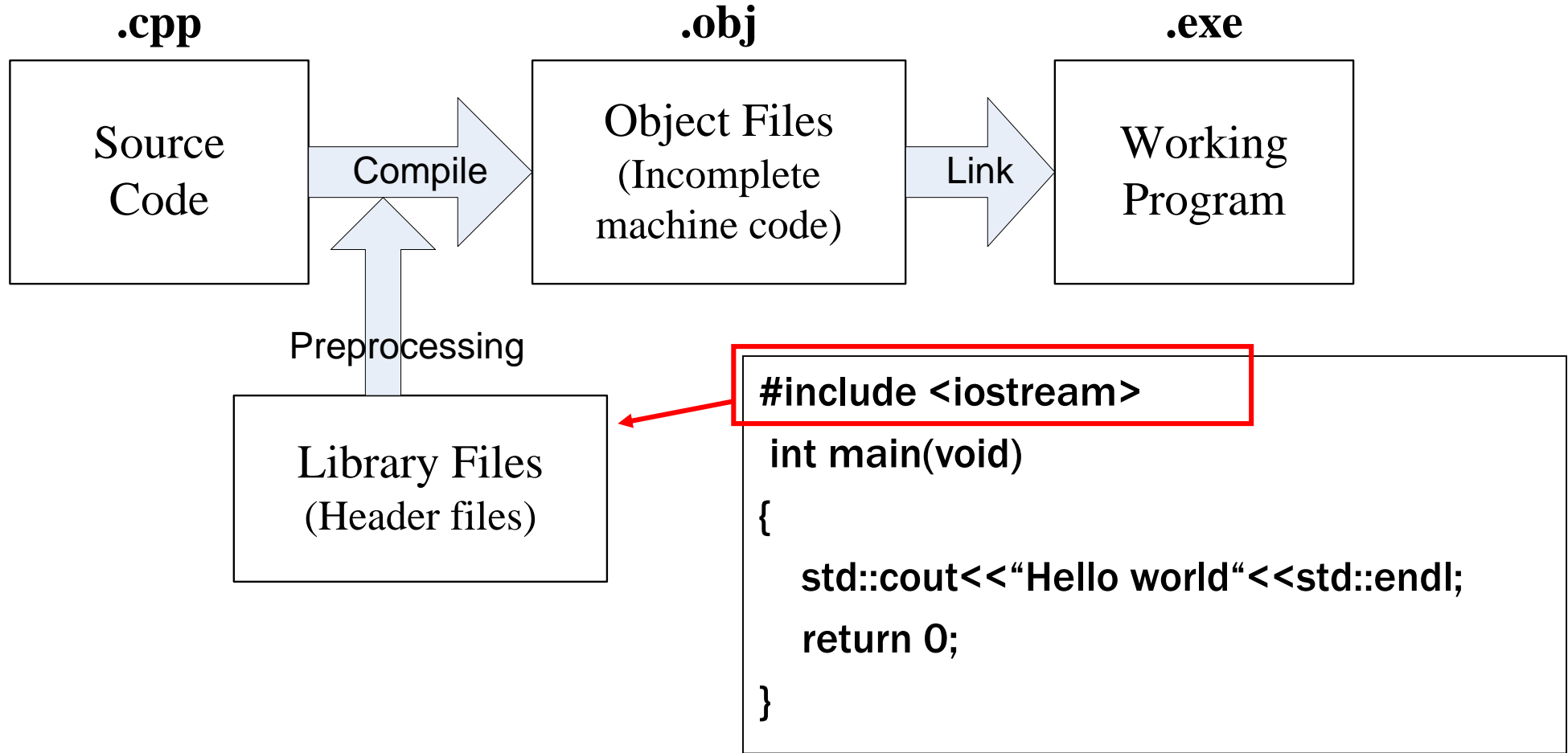
```
    return 0;
```

```
}
```



Compilation process

From C++ Source Code to Working Programme



3.4 Typical Structure of the Source Code of a C++ Main Function

Comment

```
// Comment on a single line
// 2010-Jan-01 by Z.Wang
/* Comment on multiple lines
   A programme to output information to screen*/
```

Preprocessor
directive

```
#include <iostream>
using namespace std;
```

Function
body

```
int main(void)
{
    int i;
    char c;
    i=20;
    c='J';
    cout <<"I am " <<i <<" years old." <<endl;
    cout <<"My initial is " <<c <<endl;
    return 0;
}
```

Statements



Header File

```
#include <iostream>
#include "myheaderfile.h"
```

- Pre-processor lines
- Always start with a #
- No semi-colon (;) at the end of this line
- This line will logically be replaced by the codes contained in the header file when the source code is compiled.
- A lot of pre-defined actions (functions) can be carried out without the user writing the source codes. The user only need to call the name of that particular function.

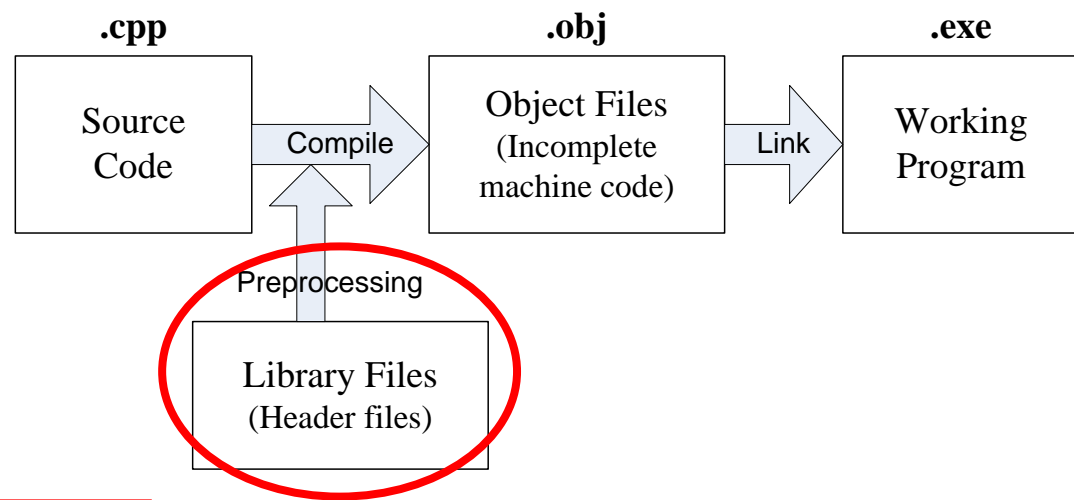
```
// Comment on a single line
// 2010-Jan-01 by Z.Wang
/* Comment on multiple lines
   A programme to output information to screen*/

#include <iostream>
using namespace std;

int main(void)
{
    int i;
    char c;
    i=20;
    c='J';
    cout <<"I am " <<i <<" years old." <<endl;
    cout <<"My initial is " <<c <<endl;
    return 0;
}
```



Pre-processing



Source file

```
// File hello.cpp
#include "pre_io.h"
void main()
{
    cout << "Hello!";
}
```

Header file

```
// File pre_io.h
#include <iostream>
using namespace std;
```

```
// File hello.cpp
#include <iostream>
using namespace std;

void main()
{
    cout << "Hello!";
}
```



Main function

```
int main(void)
{
    ..... // code here
    return 0;
}
```

```
// Comment on a single line
// 2010-Jan-01 by Z.Wang
/* Comment on multiple lines
   A programme to output information to screen*/

#include <iostream>
using namespace std;

int main(void)
{
    int i;
    char c;
    i=20;
    c='J';
    cout <<"I am " <<i <<" years old." <<endl;
    cout <<"My initial is " <<c <<endl;
    return 0;
}
```

- Every C++ source code must have and can only have one main function.
- The **int** is a return type, the **void** means no input parameters passed into the main function, **return 0** means successfully finished the program.
- **main** is a keyword in C++ which you cannot use for other purposes, such as a variable name.



Statements

One statement each line

```
int n;  
n=5;
```

Two statements on one line

```
int n ; n=5;
```

```
// Comment on a single line  
// 2010-Jan-01 by Z.Wang  
/* Comment on multiple lines  
   A programme to output information to screen*/  
  
#include <iostream>  
using namespace std;  
  
int main(void)  
{  
    int i;  
    char c;  
    i=20;  
    c='J';  
    cout <<"I am " <<i <<" years old." <<endl;  
    cout <<"My initial is " <<c <<endl;  
    return 0;  
}
```

One statement on more than one lines

```
cout<<"This statement to output information on the  
screen is too long to be placed on a single  
line";
```



Labs and assessments

- Labs
- Assessments
 - 4 assessments, takes 15%, 15%, 35%, 35% respectively in final marks
 - Submitted to ICE online (soft copy only!).