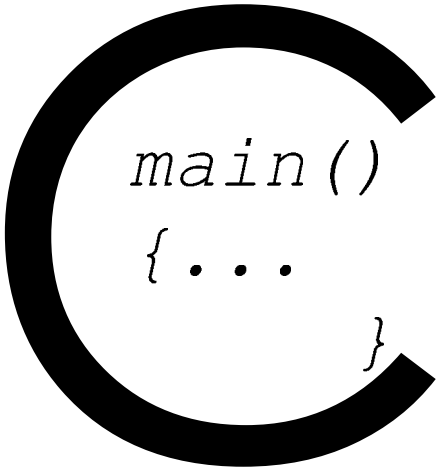
C Programming Practice
No[1]

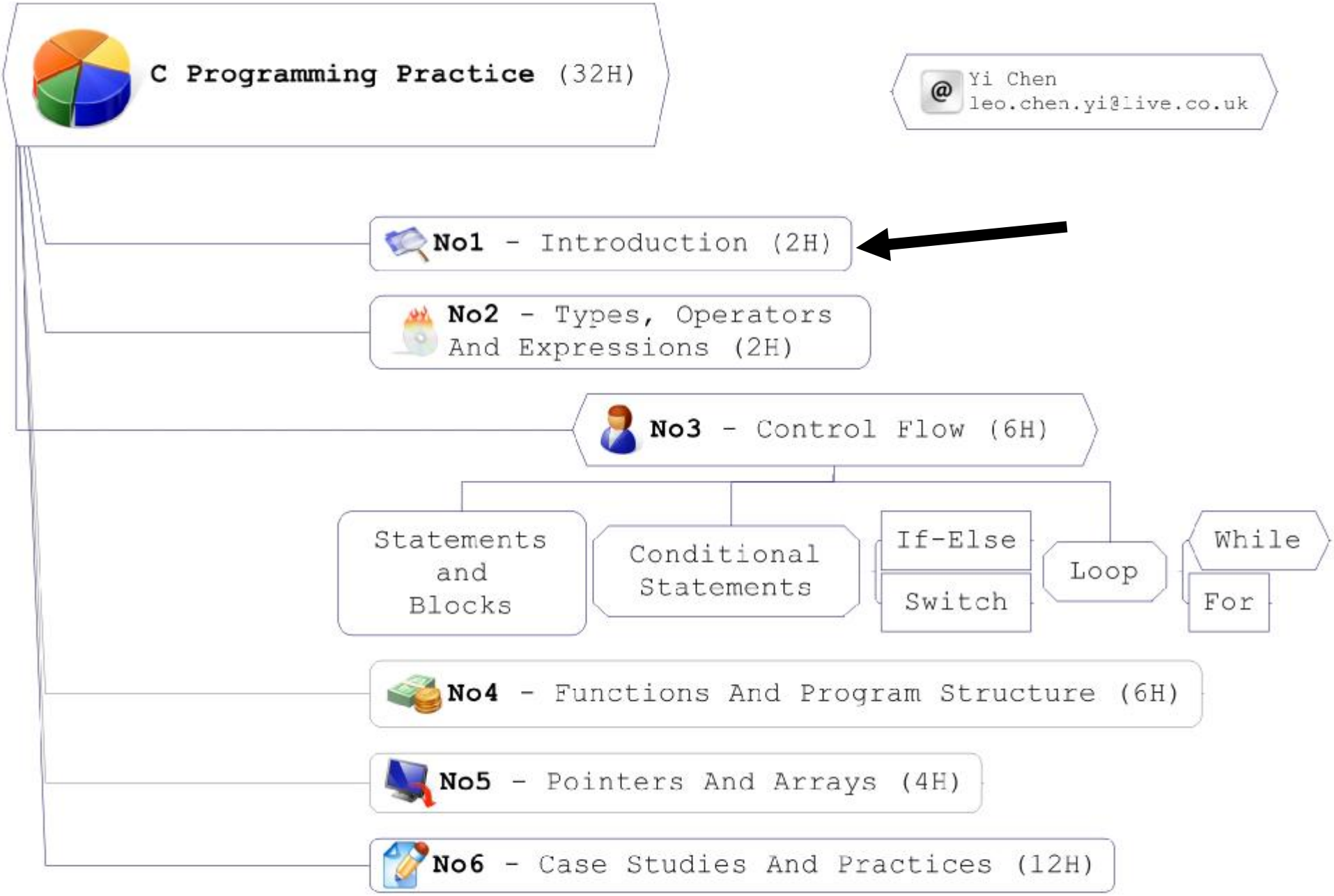
Chen , Yi
leo.chen.yi@live.co.uk
30-Jul-2010

C Programming Practices - from University Students to Engineers (C-S2E)



Introduction

BETA 1.0.0.1

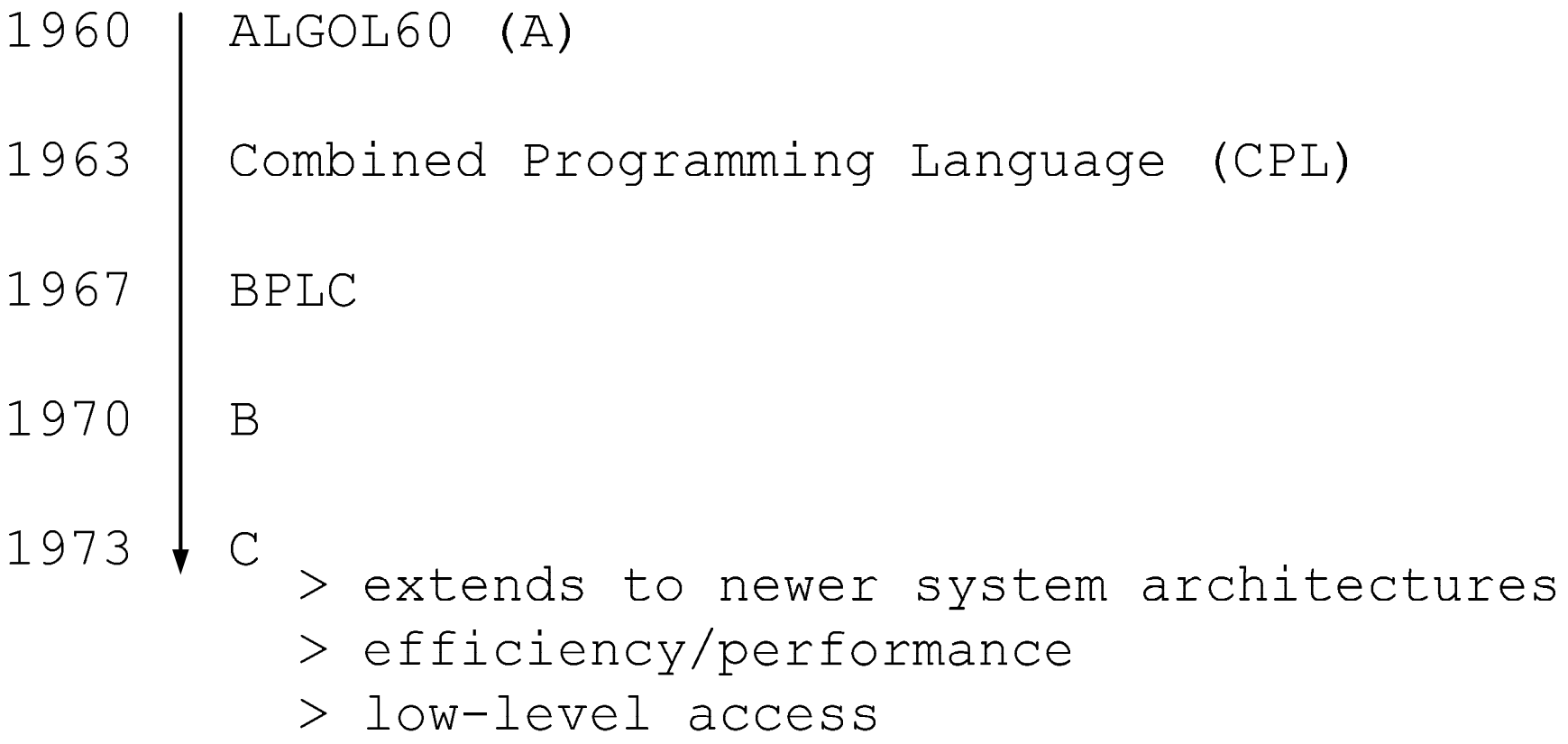

C Programming Practice (32H)
Yi Chen
leo.chen.yi@live.co.uk
No1 - Introduction (2H)
No2 - Types, Operators And Expressions (2H)
No3 - Control Flow (6H)
Statements and Blocks
Conditional Statements
If-Else
Switch
Loop
While
For
No4 - Functions And Program Structure (6H)
No5 - Pointers And Arrays (4H)
No6 - Case Studies And Practices (12H)

Contents

Contents

What is C

1972, by **Dennis Ritchie** at the Bell Telephone Laboratories for use with the Unix operating system.



What can C do

Operating System (OS), E.g. Linux

Microcontrollers: Automobiles and Airplanes

Embedded Processors: Phones, Portable Electronics, Robotic System

DSP Devices: Digital Audio and TV Systems

… any other applications you know?

Features of C

Few keywords and Cross-platform(multi-platform)

Structures, unions – compound data types

Pointers – memory, arrays

External standard library – I/O, etc.

Compiles to native code

Macro preprocessor

Standard ANSI C keywords

auto break case char const continue default do

double else enum extern float for goto if int long

register return short signed sizeof static struct

switch typedef union unsigned void volatile while

In 1983, the **A**merican **N**ational **S**tandards **I**nstitute commissioned a committee, X3J11, to standardize the C language. The Standard has been adopted as an international standard, **ISO/IEC 9899:1990**, although the Rationale is currently not included.

Version of C

1972 – C invented

1978 – The C Programming Language published

1989 – C89 standard (known as ANSI C or Standard C)

1990 – ANSI C adopted by ISO, known as C90

1999 – C99 standard, mostly backward-compatible;

not completely implemented in many compilers

2007 – work on new C standard C1X announced

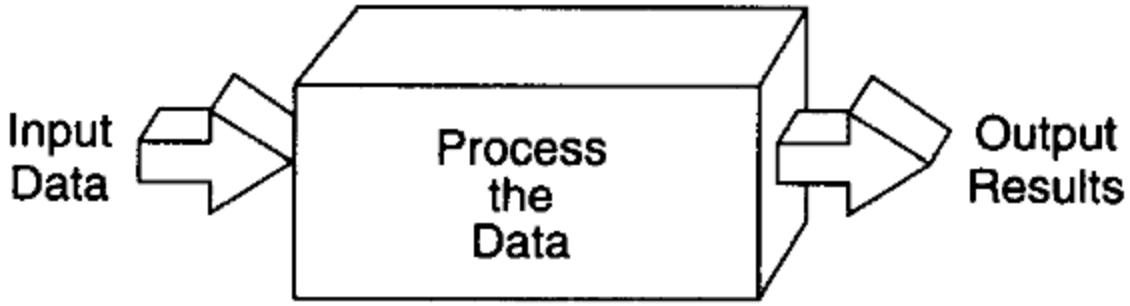
In this course: ANSI/ISO C (C89/C90)

With other languages

Usual file extensions .h .c
.h -- header file
.c -- source file

Influenced by:
CPL , BCPL, ALGOL 68, Assembly, PL/I, FORTRAN

Influenced:
AWK, csh, C++, C-- , C#, Objective-C, BitC,
D, Go, Java, JavaScript, Limbo, LPC, Perl,
PHP, Pike, Processing, Python



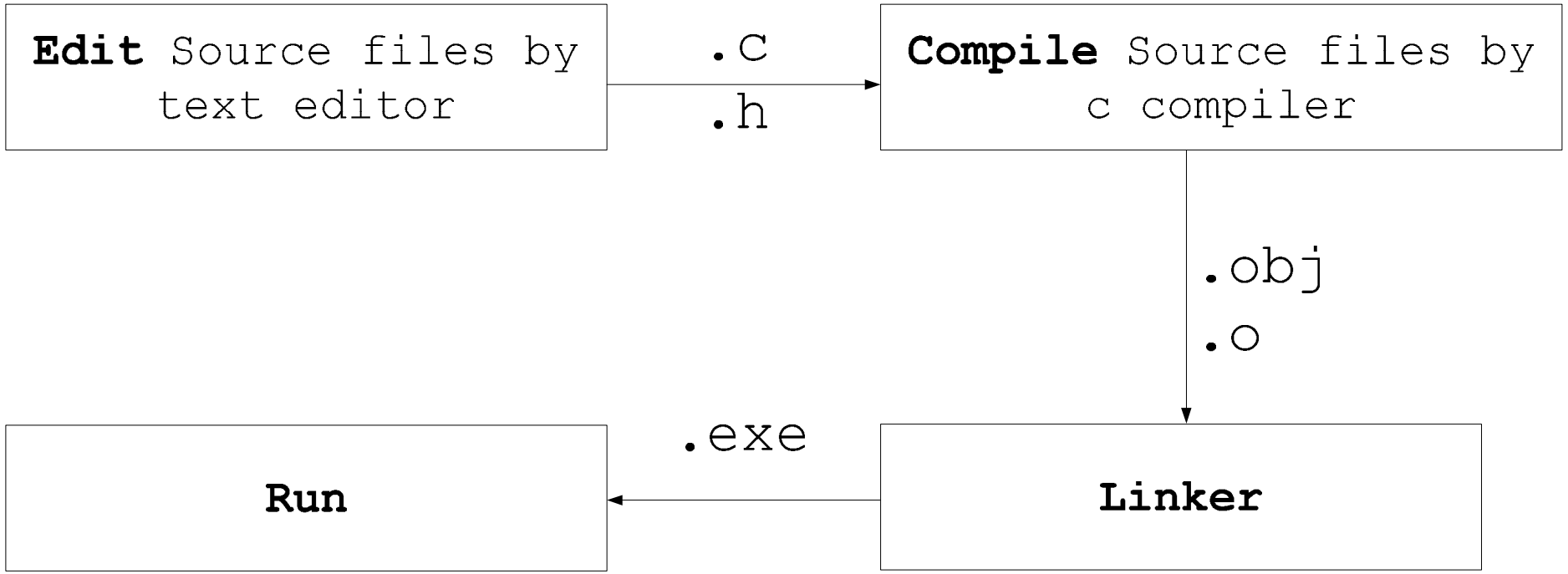
With other languages

Operation	FORTRAN	BASIC	COBOL	Pascal	C
Input (Get the data)	READ	INPUT READ/DATA	READ ACCEPT	READ READLN	getchar() gets() scanf() sscanf() fscanf()
Processing (Use the data)	= IF/ELSE DO	LET IF/ELSE FOR	COMPUTE IF/ELSE PERFORM	:= IF/ELSE FOR WHILE REPEAT	= if/else for while do
Output (Display the data)	WRITE PRINT	PRINT PRINT/ USING	WRITE DISPLAY	WRITE WRITELN	putchar() puts() printf() sprintf() fprintf()

Before the First Case

Major Steps to start the first case

C program source text is free-format,
using the semicolon as a statement terminator.



Before the First Case

*An **i**ntegrated **d**evelopment **e**nvironment (**IDE**) is a software application that provides comprehensive facilities to computer programmers for software development.*

An IDE normally consists of:
> a source code editor
> a compiler and/or an interpreter
> build automation tools
> a debugger

Some multiple-language IDEs:
Eclipse,
ActiveState Komodo,
Microsoft Visual Studio,
WinDev, and Xcode.

Before the First Case

Program: self-contained set of instructions used to operate a computer to produce a specific result Also called **software**

Programming: the process of writing a program, or software

Source code: the programs written in a high- or low-level language. Source code must be translated to machine instructions in one of two ways:

Interpreter: each statement is translated individually and executed immediately after translation

Compiler: all statements are translated and stored as an executable program, or object program; execution occurs later

Before the First Case

Computer storage:

Bit: smallest unit of data; value of 0 or 1

Byte: grouping of **8** bits representing a single character

Low-level languages:

languages that use instructions tied directly to one type of computer

Examples: machine language, assembly language

High-level languages:

instructions resemble written languages, such as English,and can be run on a variety of computer types

Examples: Visual Basic, C, C++, Java

Before the First Case

Programs can also be classified by their orientation:

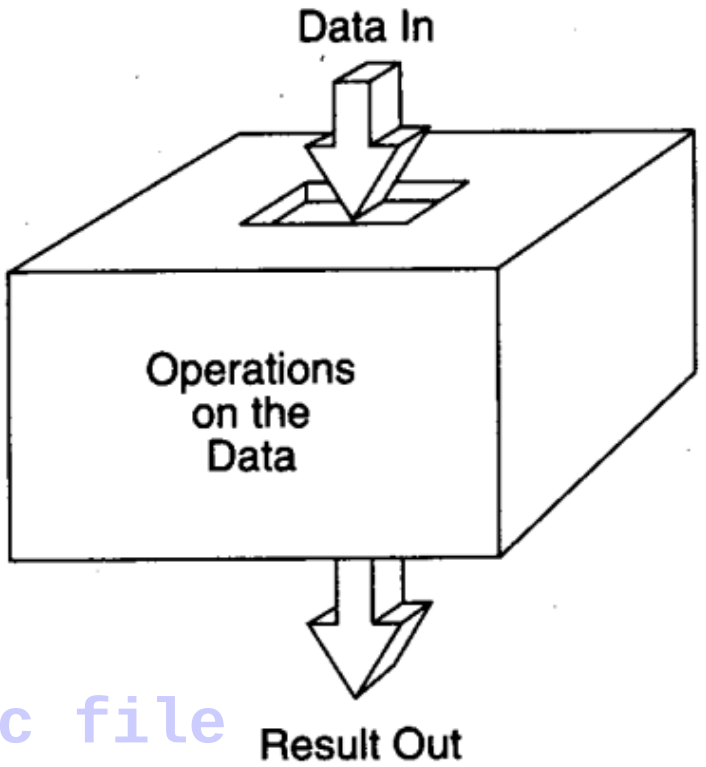
Procedural: available instructions are used to create self-contained units called procedures.

FORTRAN,ALGOL,COBOL,BASIC,Pascal

Object-oriented: reusable objects, containing code and data, are manipulated

C++,C#, VB,VFORTRAN

Contents



First Case

Dev-C++
Microsoft Visual C++
Eclipse CDT

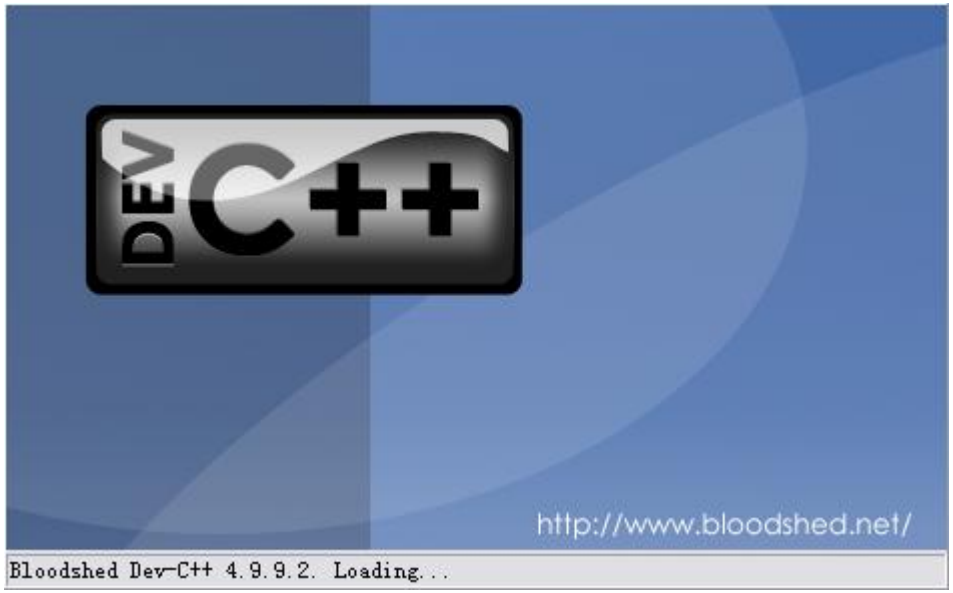
The first program to write is the same for all languages, to print the words ``**hello, world**''

```
#include <stdio.h>

int main(void)
  {
    printf("hello, world\n"); /*print string*/
    getch();                  /*waiting for keyboard*/
    return 0;                 /*return*/
   }
```

cp_case_helloworld.c

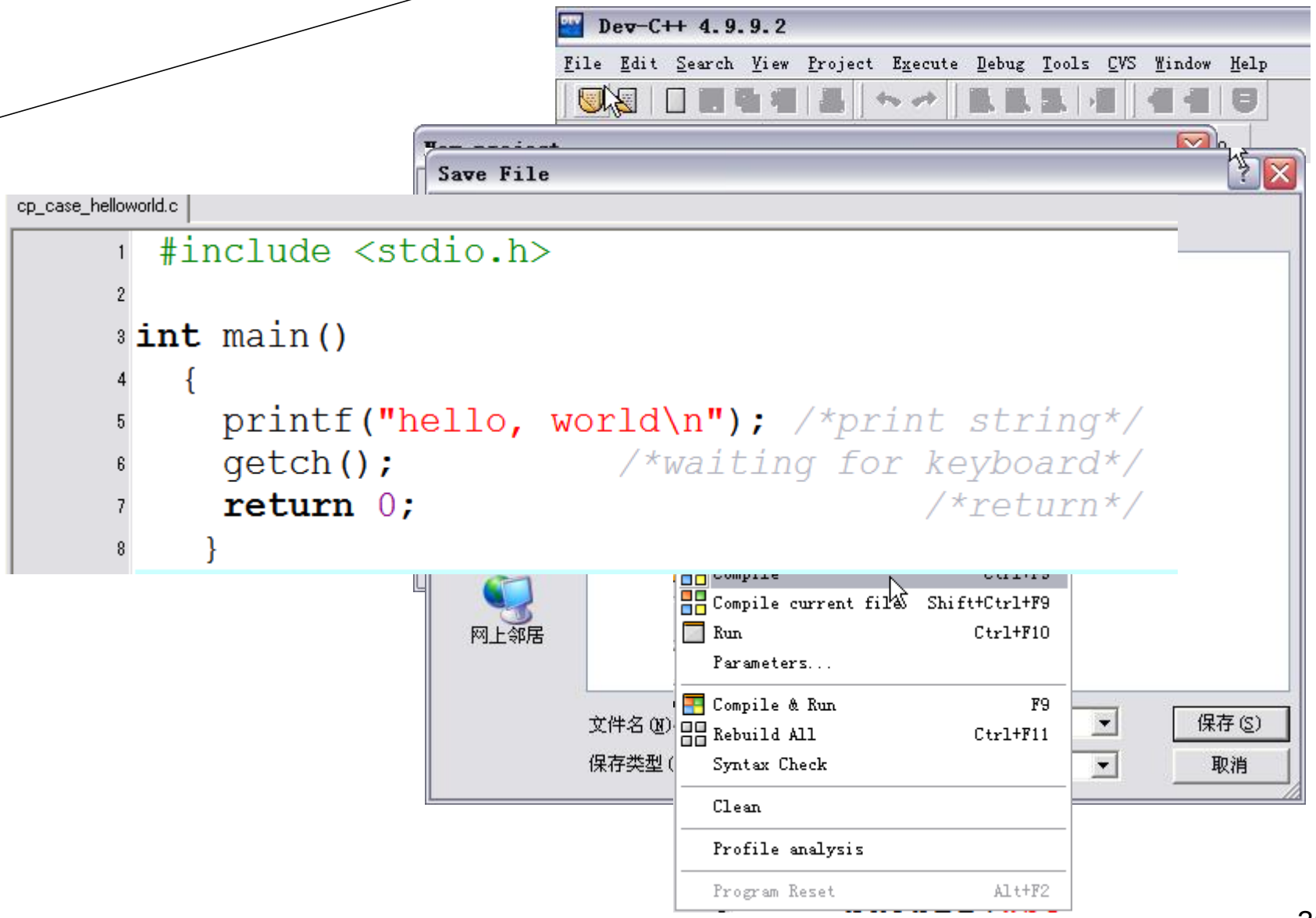
First Case

DEV-C++

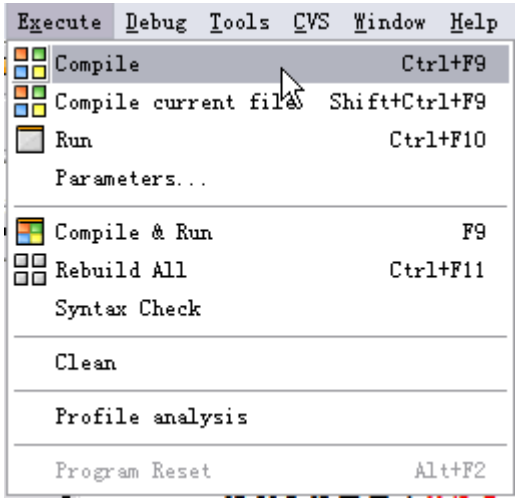


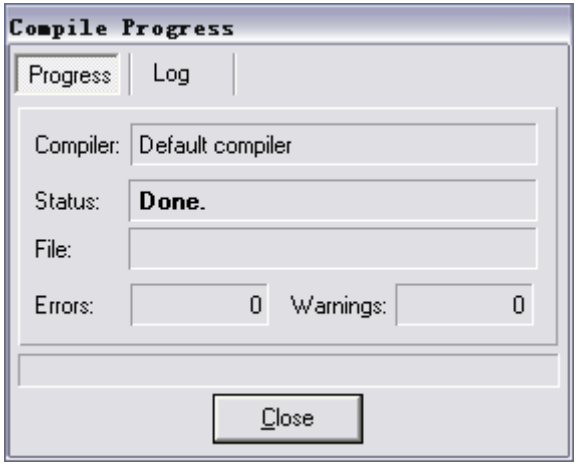
Name	Exte...
bin	
Examples	
Help	
Icons	
include	
Lang	
lib	
libexec	
mingw32	
Packages	
Templates	
copying.txt	txt
devcpp.exe	exe
devcpp.map	map
NEWS.txt	txt
Packman.exe	exe
Packman.map	map
uninstall.exe	exe

First Case


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
Save File
cp_case_helloworld.c
1 #include <stdio.h>
2
3 int main()
4 {
5 printf("hello, world\n"); /*print string*/
6 getch(); /*waiting for keyboard*/
7 return 0; /*return*/
8 }
网上邻居
Compile current file Shift+Ctrl+F9
Run Ctrl+F10
Parameters...
Compile & Run F9
Rebuild All Ctrl+F11
Syntax Check
Clean
Profile analysis
Program Reset Alt+F2
文件名(N)
保存类型(
保存(S)
取消

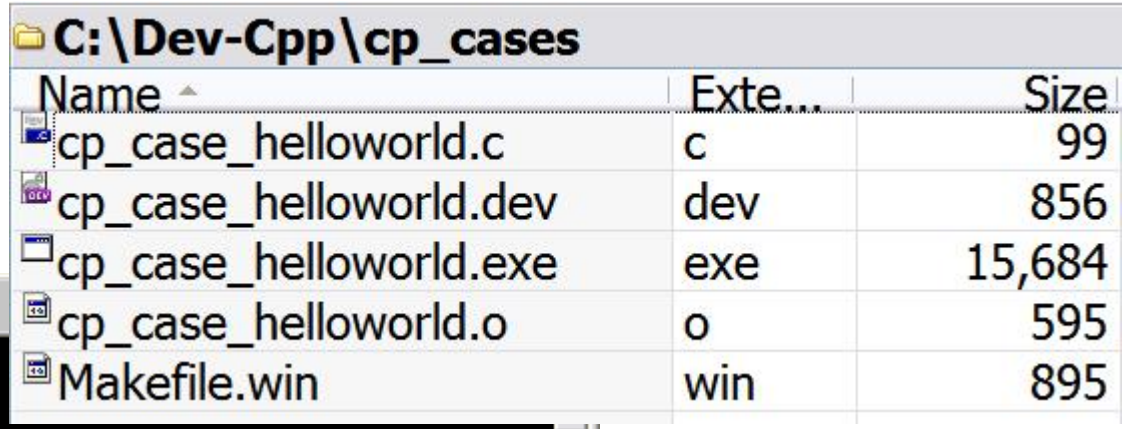
First Case





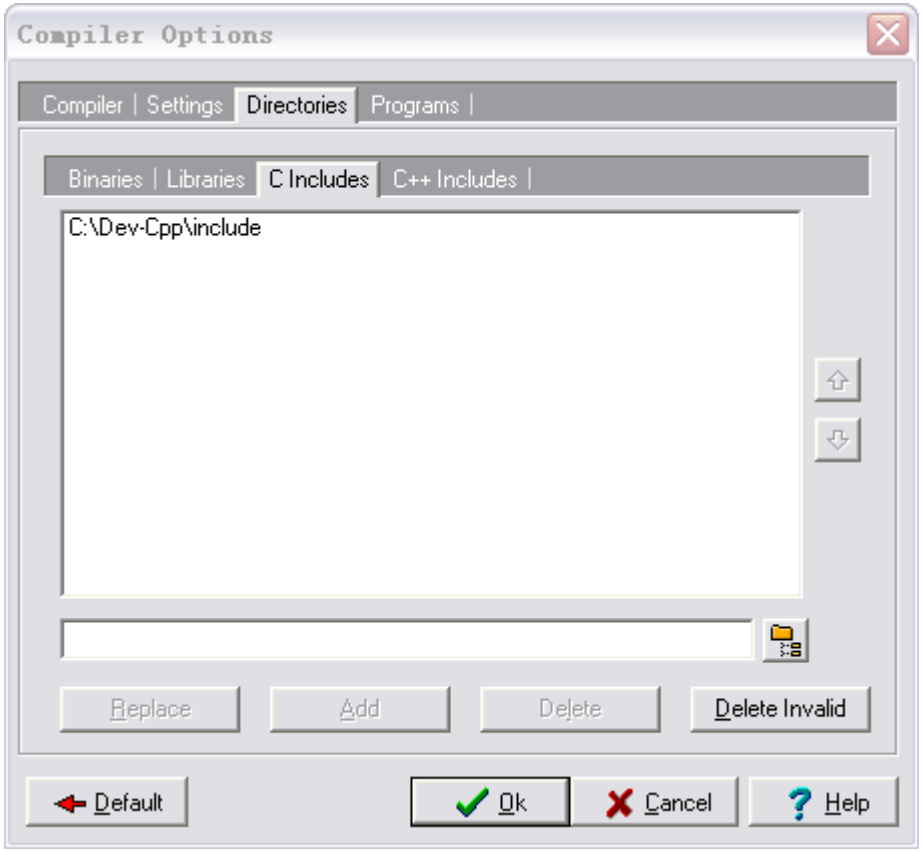

```
Compiler: Default compiler
Building Makefile: "C:\Dev-Cpp\cp_cases\Makefile.win"
Executing  make...
make.exe -f "C:\Dev-Cpp\cp_cases\Makefile.win" all
gcc.exe cp_case_helloworld.o  -o "cp_case_helloworld.exe" -
L"C:/Dev-Cpp/lib"

Execution terminated
Compilation successful
```

First Case


C:\Dev-Cpp\cp_cases
Name
Exte...
Size
cp_case_helloworld.c
c
99
cp_case_helloworld.dev
dev
856
cp_case_helloworld.exe
exe
15,684
cp_case_helloworld.o
o
595
Makefile.win
win
895


C:\Dev-Cpp\cp_cases\cp_case_helloworld.exe
hello, world

Compiler path



Contents

Structure of a .c file

```
/* Begin with comments about file contents */

#include statements and preprocessor definitions

Function prototypes and variable declarations

Define  main()  function
{
        Function body by sub-functions
}

Define sub-functions
{
        Function body
}
.
.
.
```

Comments

> **Comments**: /* this is a simple comment */

> Can span **multiple lines**

```
/*  This  comment
      spans
     multiple   lines   */
```

> Completely **ignored** by compiler

> Can appear almost **anywhere**

```
/* cp_case_helloworld.c  - our first C programme
    Created by Y. Chen 30-Jul-2010 */
```

The **#include** macro

Header files: declarations of the constants, functions and other definitions

#include <stdio.h> – read the contents of the header file stdio.h

stdio.h: standard I/O functions for console, it is a part of the C Standard Library, other important header files:
ctype.h, math.h,
stdlib.h, string.h, time.h

Included fies must be on *include* path,
e.g. *C:\Dev-Cpp\include*

```
#include <stdio.h>  /* standard I/O facilities */
#include "stdio.h"  /* searches ./ for stdio.h first*/
```

Contents

Variables Declaration

> Must declare variables before use

> e.g.

```
int    loop_number;
int    ind;
int    jnd;


float  phi;
float  distance;


/*
int   - integer data type
float - floating-point data type
Many other types see next chapter */
```

Variables Initialisation

> If uninitialised, the variables will be assumed a random address by compiler;

> It is a good practice to do the initialisation at declaration or soon after it.

> Do initialisation at declaration:

```
float  pi   = 3.14;

int    int  = 0;
int    jdx,kdx = 0;
```

> Do initialisation at declaration after declaration:

```
float  pi;   /*Semicolon(;) ends statement */

pi = 3.14;
```

Contents

What is a function

A function definition has this form:

```
return-type function-name(parameter declarations, if
any)
            {
                        declarations statements

                                                    }
```

C is a language of functions.

In C, a **function** is equivalent to a subroutine or function in Fortran, or a procedure or function in Pascal.

A function provides a convenient way to encapsulate some computation, which can then be used without worrying about its implementation.

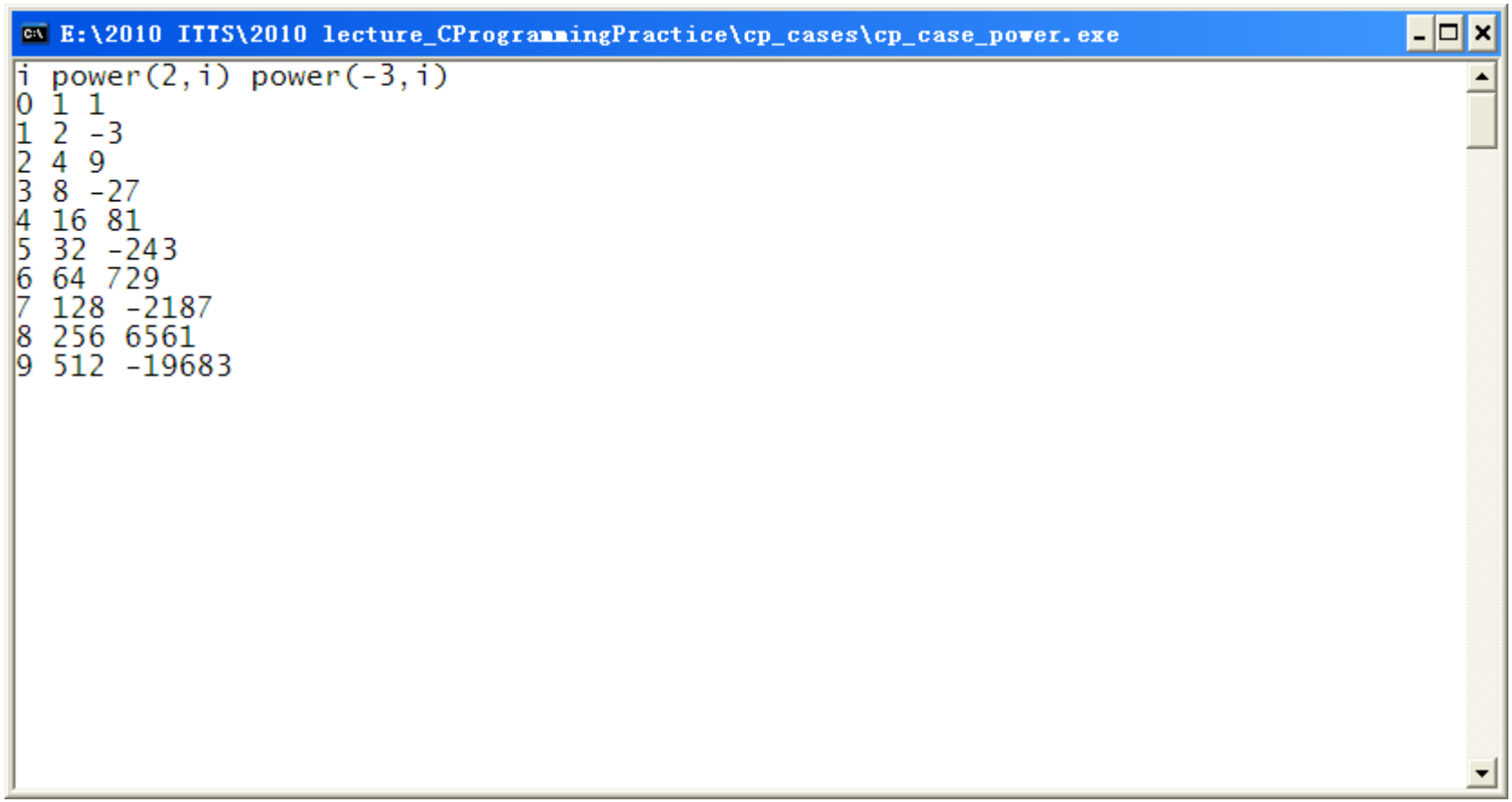
What is a function

Case study:

a function **power(x,y)** that computes x^y

power(2,5) is $2^5 = 32$

cp_case_power.c



Function prototypes

> Functions also must be declared **before** use, General form:
return_type function_name(arg1, arg2, ...);

arguments: local variables, values passed from/to caller
return value: single value returned to caller when function exits
void: no return value/arguments, e.g. int rand(void);

> Declaration called **function prototype**

> **Function prototypes**, e.g. int factorial(int input);

> Prototypes for many common functions in header files for **C Standard Library**

More files in C:\Dev-Cpp**\include**

Function definition

Function declaration
{
* declare variables;*
* program statements;*
}

> Must **match** its prototype (if there is one)

> The variable names **don't** have to match

> **No** semicolon at end

> Curly braces**{}** define a block – region of code, the variables declared in a block exist **only** in that block

> Variable declarations **before** any other statements

The main()function

> **main()**: entry point for C program, every project has a main() only;

> Simplest version: no inputs, outputs 0 when successful, and nonzero to signal some error:

```
int  main(void);
```

> Two-argument form of main(): access command-line arguments:

```
int  main( int argc,  char **argv );
```

*char **argv* will be discussed later.

Console I/O

#include <stdio.h>

> **stdout**: console output stream, e.g. to **screen**

> **stdin**: console input stream, e.g. from **keyboard**

> **puts(string)**: print string to stdout

> **putchar(char)**, **printf**(): print character to **stdout**

> char = getchar(), **getch**(): return character from **stdin**

> string = gets(string): read line from **stdin** into string

> Strings stored as character array, last character in the array is '**\0**' null

Contents

Macros Definition

#include
#define

> Preprocessor macros begin with # character, e.g. , you can define your own macro in a h file or the front part of a c file.

> String and number macros

```
#define   msg                              "hello, world"
#define   SMAT_SPACE_CHAR                  ' '
#define   SMAT_SPECIAL_CHAR_LEFT_BRACE     '{'
#define   SMAT_SPECIAL_CHAR_RIGHT_BRACE    '}'

#define   SMAT_STATUS_OK                    0
#define   SMAT_STATUS_FAIL                 -1
```

> Defining expression macros, parentheses() ensure order of operations, and compiler performs inline replacement, it is not suitable for recursion;

#define add3(x,y,z) ((x)+(y)+(z))

Macros Definition

> ***#if, #ifdef, #ifndef, #else, #elif , #endif***

conditional preprocessor macros, can control which lines are compiled.

> **#pragma**

preprocessor directive

> **#error, #warning**

trigger a custom compiler error/warning

> **#undef msg**

remove the definition of **msg** at compile time

```
#define FLAG

int main()
{
  #ifdef FLAG
    Lines - 1
  #else
    Lines - 2
  #endif
return 0;
}
```

Format Specifiers

```
printf("1905+31214 is %d", 1905+31214);

printf("pi =  %f", 3.14);

printf("my first message is  %s", "Hello_world.c");
```

%i - int (same as **%d**)
%f - float
%lf - double
%s - string
%x - hexadecimal

Exercise-1

cp_ex01_F2C.c
cp.h

How to convert the Celsius temperature(**Cº**) to Fahrenheit temperature(**Fº**), use formula

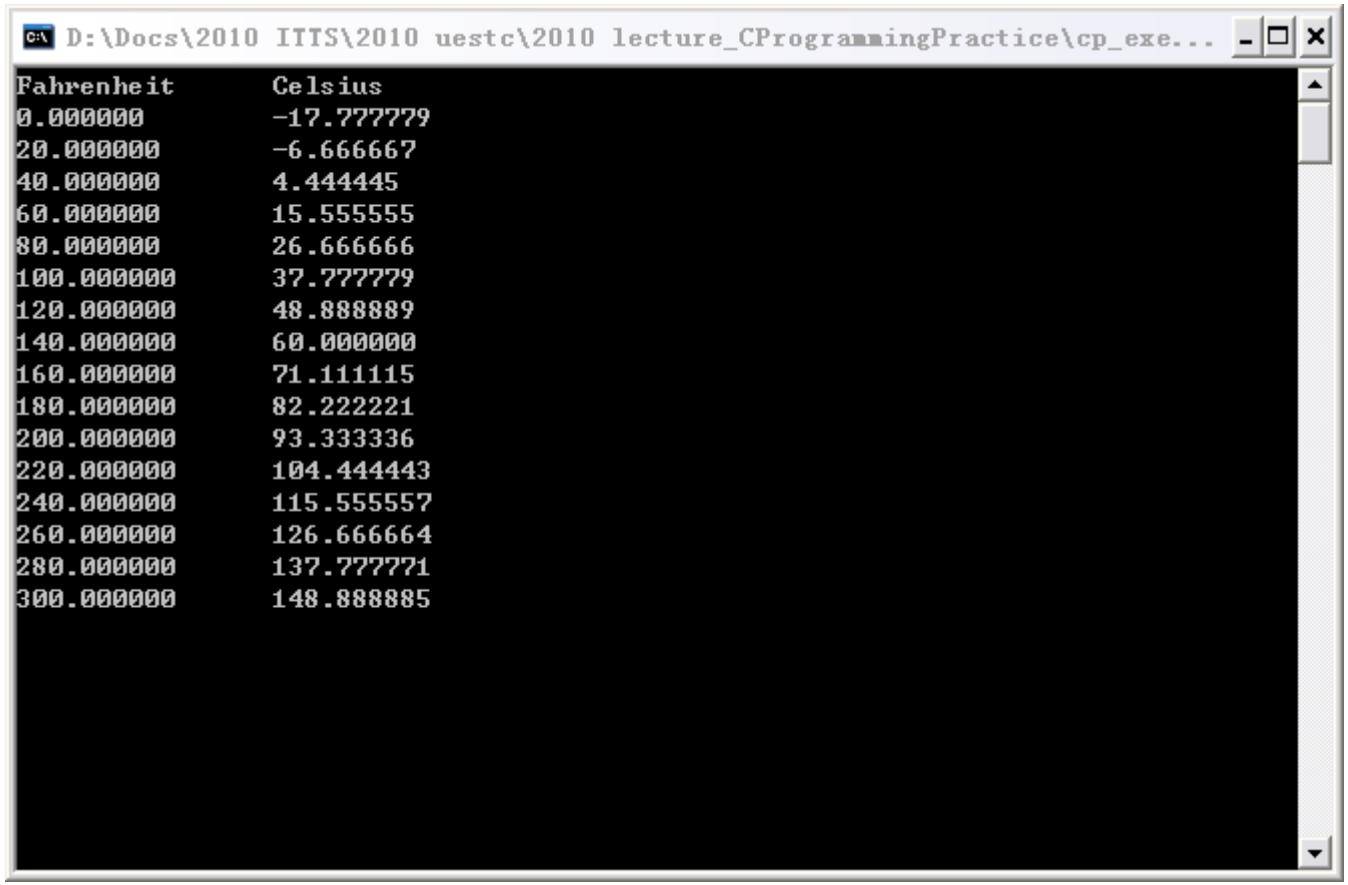
C = (5/9)(F-32)

This function need to print the following table of Fahrenheit temperatures(F)and their centigrade or Celsius equivalents(C).

Use **float**

F	C
1	-17
20	-6
32	0
40	4
60	15
80	26
100	37
120	48
140	60
160	71
180	82
200	93
220	104
240	115
260	126
280	137
300	148

Exercise-1


D:\Docs\2010 ITTS\2010 uestc\2010 lecture_CProgrammingPractice\cp_exe...
Fahrenheit Celsius
0.000000 -17.777779
20.000000 -6.666667
40.000000 4.444445
60.000000 15.555555
80.000000 26.666666
100.000000 37.777779
120.000000 48.888889
140.000000 60.000000
160.000000 71.111115
180.000000 82.222221
200.000000 93.333336
220.000000 104.444443
240.000000 115.555557
260.000000 126.666664
280.000000 137.777771
300.000000 148.888885

Exercise-2

The following lines of code, when arranged in the proper sequence, output the simple message “All your base are belong to us.”

```
1. return 0;

2. const char msg[] = MSG1;

3. }

4. #define MSG1 "All  your  base  are  belong  to  us!"

5. int  main(void) {

6. #include <stdio.h>

7. puts(msg);
```

Exercise-2

```
#include <stdio.h>

#define MSG1 "All  your  base  are  belong  to  us!"

int  main(void) {

 const char msg[] = MSG1;

 puts(msg);

 return 0;

}
```

Exercise-3

For each of the following statements, explain **why** it is not correct, and fix it.

(a)
```
#include  <stdio.h>;
```

(b)
```
int function(void  arg1)
{
   return  arg1-1;
                    }
```

(c)
```
#define  MESSAGE  =  "Happy  new  year!"
puts(MESSAGE);
```

Exercise-3

Solution:

(a)
```
#include  <stdio.h>;
```

(b)
```
int function(void  arg1)
{
        return arg1-1;
}
```

(c)
```
#define  MESSAGE  =  "Happy  new  year!"
        puts(MESSAGE);
```

Problem Solution

Software development procedure:

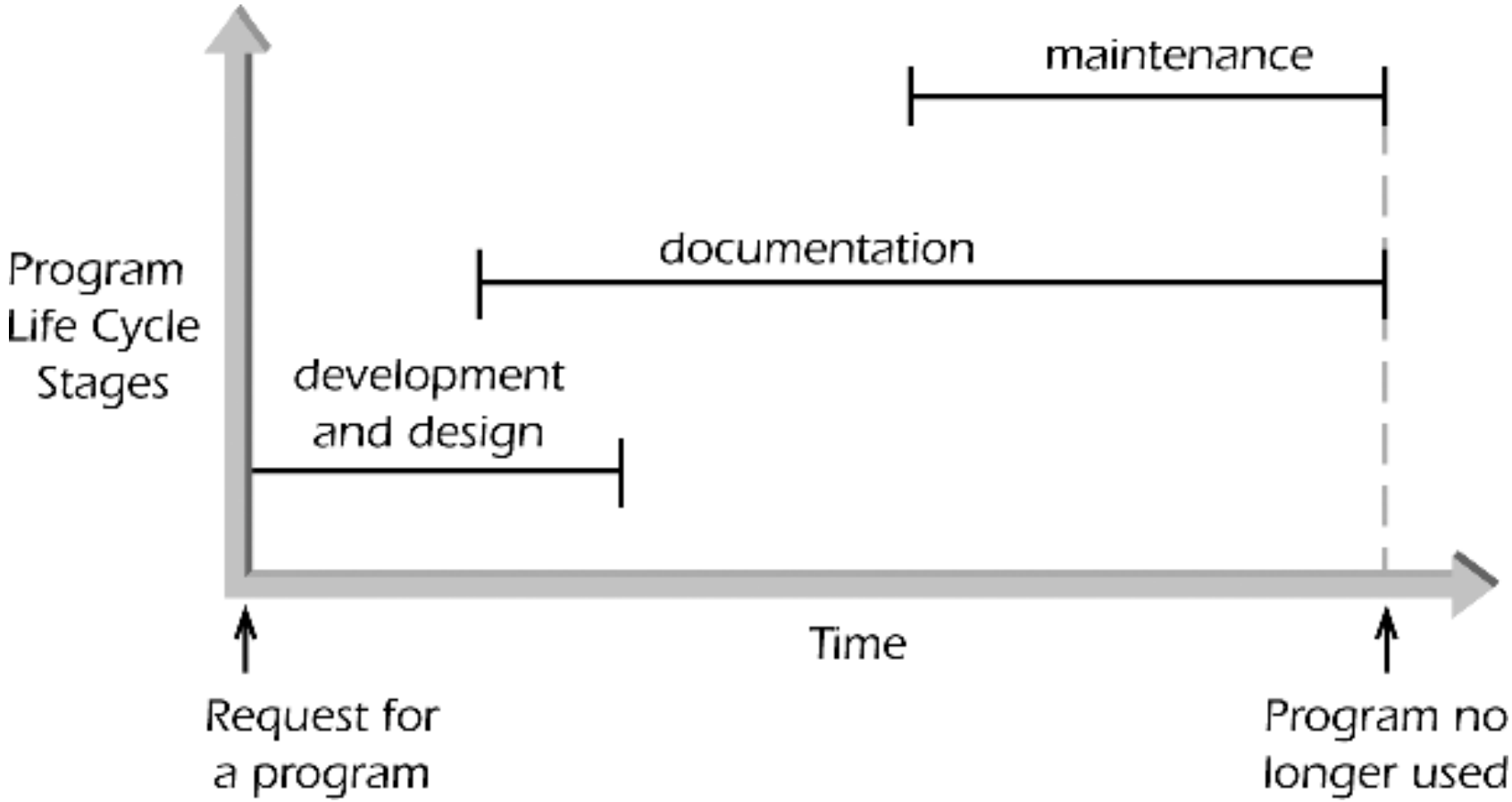
method for solving problems and creating software solutions, consists of three phases:

Phase I: Development and Design
Phase II: Documentation
Phase III: Maintenance

Software engineering:

discipline concerned with creating efficient, reliable, maintainable software

Problem Solution



Problem Solution

Phase I: Development and Design

Program requirement:

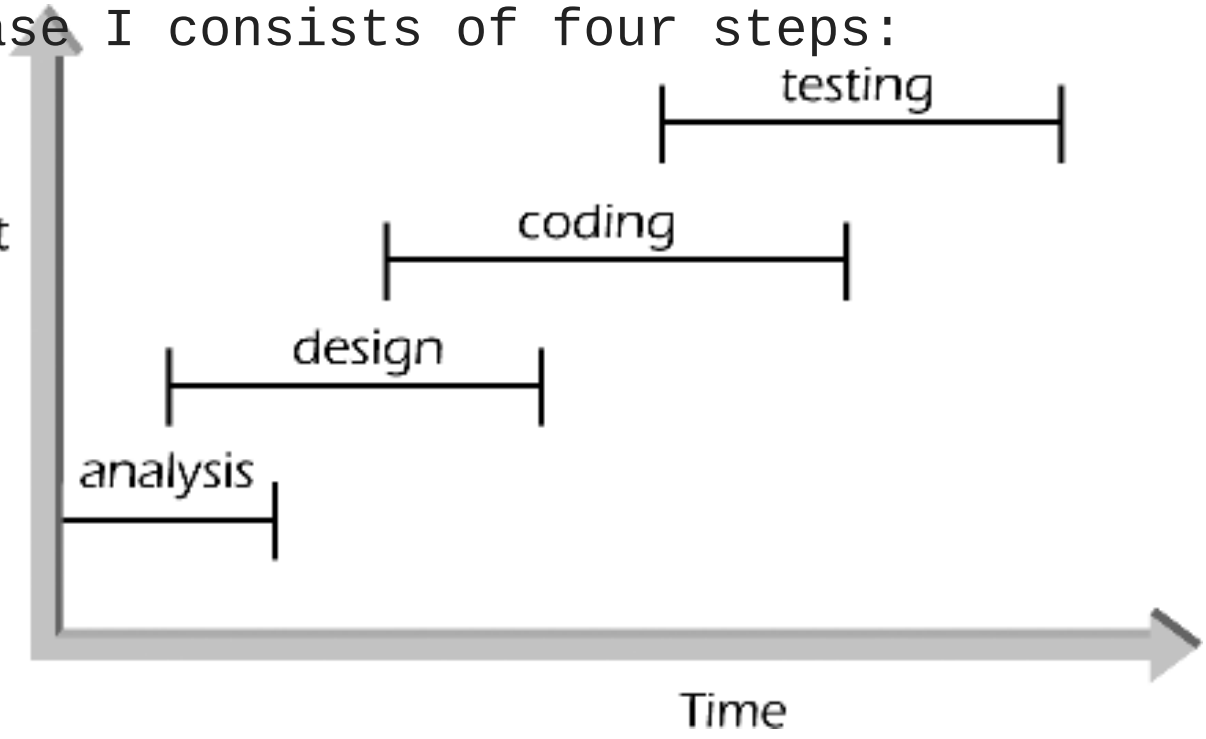
request for a program or a statement of a problem
After a program requirement is received,
Phase I begins, Phase I consists of four steps:

10% Analysis

20% Design

20% Coding

50% Testing



Problem Solution

Phase I: Development and Design

Step 1: Analyze the Problem

-Analyst must understand

-What outputs are required

-What inputs will be needed

-How the inputs can be used to produce the desired outputs

-Failure to analyze and understand the requirements leads to a failed solution

Step 2: Develop a Solution

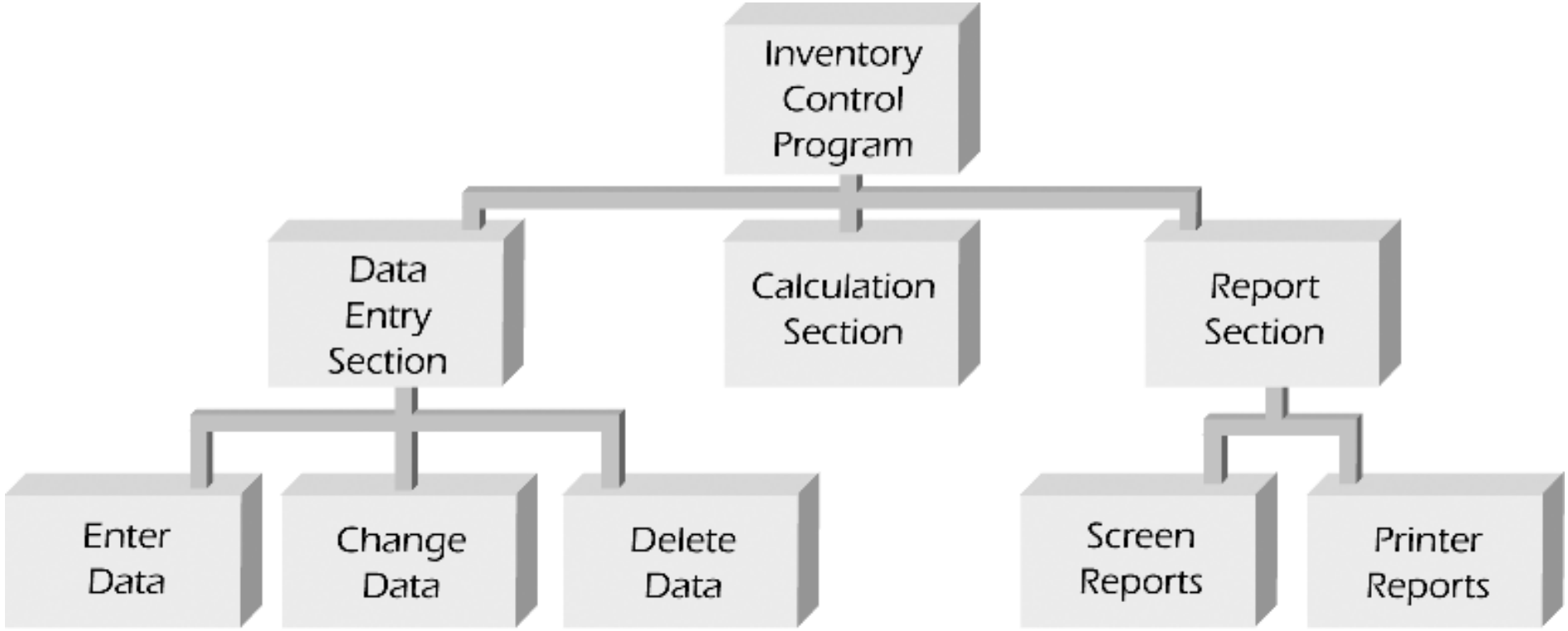
-Algorithm: the exact steps used to solve a problem

-Algorithm is usually defined at high level, then refined to detailed lower levels

-Structure level diagrams may be used to represent the levels of analysis

Problem Solution

Phase I: Development and Design



Step 3: Code the Solution

-Also called writing the program, and implementing the solution

-Program should contain well-defined patterns or structures of the following types:

Sequence: defines the order in which instructions are executed
Selection: allows a choice between different operations, based on some condition
Iteration: allows the same operation to be repeated based on some condition; also called *looping* or *repetition*
Invocation: involves invoking a set of statements when needed

Step 4: Test and Correct the Program

Testing: method to verify correctness and that requirements are met

Bug: a program error

Debugging: the process of locating an error, and correcting and verifying the correction

Testing may reveal errors, but does not guarantee the absence of errors

Phase II: Documentation

Many documents may be required, including:

- **Program description**
- **Algorithm development and changes**
- **Well-commented program listing**
- **Sample test runs**
- **Users' manual**

Problem Solution

Phase III: Maintenance

Maintenance includes:

- -Ongoing correction of newly discovered bugs
- -Revisions to meet changing user needs
- -Addition of new features

Maintenance usually the longest phase, and may be the primary source of revenue

Good documentation vital for effective maintenance

Problem Solution

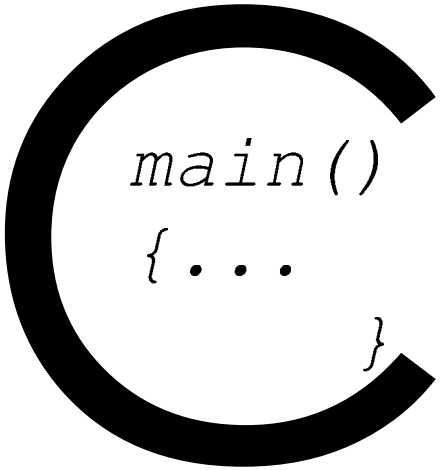
Backup:
process of making **copies** of program code
and documentation **on a regular basis**

Backup copies = insurance against loss or damage
- Consider using off-site storage for additional protection

Links

[1] ANSI C for Programmers on UNIX Systems
http://www.eng.cam.ac.uk/help/tpl/languages/C/teaching_C/teaching_C.html

Chen , Yi
leo.chen.yi@live.co.uk
30-Jul-2010



C Programming Practices - from University Students to Engineers (C-S2E)

01 - Introduction

End

BETA 1.0.0.1

Dennis Ritchie dies aged 70

Dr Ritchie was one of the creators of the hugely influential Unix operating system and the equally pioneering C programming language.

Along with Ken Thompson, Brian Kernighan, Douglas McIlroy, and Joe Ossanna, Dr Ritchie was one of the key creators of the Unix operating system at Bell Labs during the 1960s and 70s.

http://www.bbc.co.uk/news/technology-15287391


[江城子.中秋.Scotland-Highland]
琴声细雨浸云舫，熏衣草，百里香，
千里高地风笛邀裙裳。
羽扇纶巾谈笑间，论天地，海内乃。
夜来幽梦已还乡，孩童装，竹径旁，
往事花开，絮叙尽欢畅。
松风潺潺鹊鸣处，圆月夜，江亭上。