EEE101: C Programming & Software Engineering I

Lecture 2: Fundamentals of C Programming

Dr. Rui Lin/Dr. Mark Leach
Office: EE512/EE510
Email: rui.lin/mark.leach@xjtlu.edu.cn
Dept. of EEE XJTLU

Outline of Today's Lecture (week 2)

- Your first C Program
- Computer Data
- Number Systems
- Fundamentals of the C programming language

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)

The previous 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

The #include Directive

Syntax: **#include** <header file>

Examples:

#include <stdio.h> **#include** <math.h>

Interpretation:

- Tell the preprocessor where to find the definitions of standard identifiers used in the program
- Definitions are collected in files called standard header files
- Examples:
  - stdio.h contains definitions of standard input/output functions such as scanf and printf
  - math.h contains definitions of common math functions such as pow(2,3) and sqrt(4)

The #define Directive

Syntax: **#define** NAME value

Examples:

- **#define** PI 3.141593
- **#define** MAX 100

Interpretation:

- The preprocessor is notified that it is to replace each use of the identifier NAME by value

Remarks:

- C program statements cannot **can not change** the value associated with NAME

The main () Function (1/2)

Syntax:

```
int main (void)
        {
        main function body
        }
```

example:

```
int main (void)
        {
        printf("Programming in C is fun.\n");
        return 0;
        }
```

The main () Function (2/2)

Interpretation:

- C Program execution begins and (normally) ends with the main function.
- Braces {} enclose the main function body which contains declarations and executable statements

Observations:

- **int** main (**void**) indicates that the main function returns an integer value to the operating system when it finishes normal execution (the example returns zero)
- **int** main (**void**) indicates that the main function receives no parameters/functions from the operating system before execution.

Character Set in C

The characters in C are grouped into the following categories:

Letters:

UPPERCASE A…Z, lowercase a…z

Digits:

0…9

Special Characters:

e.g. + = - / ! : ; etc.

White Space:

e.g. blank space, tab, return (enter)

Keywords in C

Some words have fixed meanings

These meanings cannot be changed

Must be written in **lowercase**

Are used as basic building blocks

Around 32 Keywords, some examples:

break	**case**	**char**
const	**do**	**else**
float	**for**	**if**
int	**return**	**sizeof**
static	**struct**	**typedef**
union	**void**	**while**

Standard Identifiers in C

These have special meaning or use in C

They **can** be redefined by the programmer (**NOT recommended!**)

examples:

printf

scanf

names of operations (identifiers) defined in the standard header file stdio.h

User-Defined Identifiers

These refer to names of variables, functions and arrays

They are defined by the user (That's **YOU**)

Names can be chosen following these rules:

- First character **must** be a letter or underscore
- **Must** only contain characters, digits, underscores
- Must **not** be a C keyword
- Must **not** contain whitespace
- Should **not** be longer than 31 characters

User-Defined Identifiers

Q. Are the following examples valid identifiers:

Hello_My_Name_Is_Tom

hello_my_name_is_tom

F12345

1F2345

F12 45

Constants

Refer to fixed values that can **not** change during the execution of a C program

C supports several types of constants:

- Numeric Constants

  Integer: 123, -321, 0, 12345, 100

  Real: 123.0, -321.123, 0.009, 1.12345, +100

- Character Constants

  Single: ‘X’, ‘5’, ‘x’ (within single quotes ‘ ‘)

  String: “Hello!”, “2015”, “5+3”, “I am Dr Lin”

  (within double quotes “ “)

Variables and Variable Declarations

A variable:

- Is a data name that can be used to store a data value
- May take different values at different times during program execution
- Associated with a data type (i.e. character or number)

Variable declaration:

- Tells the compiler the variable name
- Specifies the type of data the variable will hold

Syntax: **data_type** name

Examples: **int** number; **char** name; **double** sum, total;

Computer Data
and
Number Systems

Bits, Bytes…and Nibbles

Bits:

- bit – derived from **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 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 **???**

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**

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:

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

where a are the digits used in the number system of base *b* and *n* can be any natural number between 0 and *b*-1

Examples:

In decimal (*b*=10):

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

In binary (*b*=2)

011 (3 in decimal) $= (0*2^2)+(1*2^1)+(1*2^0)$

Fundamental Data Types & ASCII

Data Type	Description	Byte size	Range
int	Integer Numbers	2	-32,768 – 32,767
float	Real Numbers	4	10^{-38} – 10^{+38} (approx.)
char	Characters	1	ASCII

ASCII

The **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (ASCII) is a standard **7**-bit code proposed by ANSI (**A**merican **N**ational **S**tandards **I**nstitute) in1963, finalised in 1968

Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as ‘a’ or ‘@’ which have codes 141 and 40 (see over)

ASCII

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

ASCII

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0	96	60	1100000	140	`
49	31	110001	61	1	97	61	1100001	141	a
50	32	110010	62	2	98	62	1100010	142	b
51	33	110011	63	3	99	63	1100011	143	c
52	34	110100	64	4	100	64	1100100	144	d
53	35	110101	65	5	101	65	1100101	145	e
54	36	110110	66	6	102	66	1100110	146	f
55	37	110111	67	7	103	67	1100111	147	g
56	38	111000	70	8	104	68	1101000	150	h

Fundamentals of the C Programming Language

Comment Lines

Comment lines are for you to give descriptive information about your code.

The C compiler ignores comment lines.

```
/* This is a C style comment */
/*********************
  This is also acceptable
**********************/

// This is C++ style DO NOT USE THIS
/* /* Nested comments cannot be used */ */
```

Variable Initialisation

```
#include <stdio.h>
int main ()
{
        int a = 3;      /*declare and initialise variables*/
        float x = 3.2;
        char name = 'M';
        return 0;
}
```

Statements

```
#include <stdio.h>
int main (){
        printf("Hello.\n");                /*statement 1*/
        printf("How are you.\n");          /*statement 2*/
        printf("I am fine.\n");            /*statement 3*/
        return 0;                          /*statement 4*/
}
```

The body of main is a set of statements

Statements are instructions to the computer

The end of a statement has a semicolon ;

Blank space is ignored

Statements are executed in sequence from top to bottom

Assignment Statements

```
int main (){
        int a;          /*declarations*/
        float x;
        a=3;            /*assignments: a is assigned value 3*/
        x=a+3.2;                /*what is the value of x??*/
        return 0;
}
```

Syntax: variable = expression

The equal sign (=) is called the assignment operator

Expression is evaluated first then assigned to variable

The Function printf() (1/2)

It is used to output (usually to the screen)

Can be passed with or without arguments (an expression to evaluate)

Without argument

printf(“hello”); just prints letters hello

With arguments

%d prints integer

%f prints real number (float)

%c prints a character

%s prints a string

The Function printf() (2/2)

```
#include <stdio.h>
int main () {
        int x = 72; char b = 'Z'; float c = 3.141;
        printf("x equals %d\n",x);
                /*On the screen: x equals 72*/
        printf("b equals %c\n",b);
                /*On the screen: b equals Z*/
        printf("%d multiplied by %f equals %f", x, c, x*c);
/*On the screen: 72 multiplied by 3.141 equals 226.152*/
        return 0;
}
```

Bindings/Precedence

```
#include <stdio.h>
int main (){
        int x=3, y=5, w, z;
        w=x+y*5;   z=(x+y)*5;
        printf("x equals %d\n z equals %d", w, z);
        return 0;
}
```

w and z have different values

As in mathematics, operations have different binding strengths e.g. multiply (*) stronger than add (+)

ALWAYS use parenthesis ( ) to ensure desired result

Common Errors

Syntax errors:

Means you have typed something wrong

Run-time errors:

Happens when the program tries to perform an illegal operation e.g. divide by 0, or input the wrong data type from the keyboard

Logic errors

Due to a faulty algorithm e.g. an incorrect calculation or out of sequence statements.

Laboratory 1

In this weeks laboratory class you will be learning about how to use a compiler to write, compile and run some simple programs.

Note: We will be using Visual Studio 2013 as a compiler in the laboratory. You may use any compiler you like at home, but when submitting coursework it **must** run on Visual Studio 2013.

Week 3

Next week we will be looking again at data types, in particular characters, arrays and strings

We will also consider some mathematical operations

Finally, some input and output functions.

Thank you for your attention ☺

See you in the laboratory…