CPT109: C Programming & Software Engineering I

Lecture 2: Fundamentals of C Programming

Dr. Xiaohui Zhu
Office: SD535
Email: Xiaohui.zhu@xjtlu.edu.cn

Outline of Today's Lecture (week 2)

- First program review…
- C-Language Data
- Variables and Constants
- Computational Problems
- Characters, Arrays and Strings
- C Fundamentals
- Basic Input/Output Functions

That first program again…

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 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 **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.

C-Language Data

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**
include	**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

Variables and Constants

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;

Variables

- A variable is a *named memory location* that can hold a value (in binary)
- In C (and most languages) a variable must be declared before use.
- Declaration does 2 things:
  - Determines the type of variable
  - Reserves specific memory space for that variable

Fundamental Data Types & ASCII

Type	Description	Bytes	Range
int	Integer Numbers	4	-2,147,483,648 to 2,147,483,647
float	Real Numbers	4	10^{-38} – 10^{+38} (approx.)-6 digit precision
char	Characters	1	ASCII (-128 – 127)

characters

- **ASCII - A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- **7**-bit code **- ANSI** (**A**merican **N**ational **S**tandards **I**nstitute) 1968

Computers only understand numbers. ASCII is the global standard numerical representation for each character. Example:

‘a’ = 97

‘@’ = 64

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

Data Types and Ranges

Data Type	Description	Bytes	Range
short	short integer	2	-32,768 to 32,767
unsigned short	positive short integer	2	0 to 65,535
int	integer	2 or 4	see short or long
unsigned int	positive integer	2 or 4	see unsigned short or long
long	long integer	4	-2,147,483,648 to 2,147,483,647
unsigned long	positive long integer	4	0 to 4,294,967,295
float	single precision real number	4	1.2×10^{-38} to 3.4×10^{38} (6 digits of precision)
double	double precision real number	8	1.7×10^{-308} to 1.7×10^{308} (10 digits of precision)
long double	double precision real number	12	3.4×10^{-4931} to 3.4×10^{4931} (10 digits of precision)
char	character	1	-128 to 127
unsigned char	positive character	1	0 to 255

Variable Initialisation

```
#include <stdio.h>
int main ()
{
        int a = 3, b;   /*declare and initialise variables*/
        float x = 3.2;
        char name = 'M';
        a=4;                /*update variable value*/
        return 0;
}
```

Constants

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

- Using the keyword **const** before the variable declaration prevents its value from being changed.

e.g. **const int** a=7;

const float pi=3.141; or

const char msg[]="warning";

Computational Problems

Basic Operations

- C Provides operators for all basic mathematical functions using the standard symbols:
  - Multiply (*)
  - Divide (/)
  - Subtract (-)
  - Addition (+)
- Constants and Variables can be operated on
- There are many additional operators that we will look at next week

Computational Problems

- Real and integer numbers are stored differently
  - Integer – direct binary conversion
  - Real – float/double – use IEEE-754 format (power series)
- How does the computer deal with this e.g.

```
float answer;
int x=24, y=10;
answer=x/y;
```

What value is stored in answer?

Computational Problems

- Real and integer numbers are stored differently
  - Integer – direct binary conversion
  - Real – float/double – use IEEE-754 format (power series)
- How does the computer deal with this e.g.

```
float answer;
int x=24, y=10;
answer=x/y;
```

What value is stored in answer? 2.0

How can the correct value be obtained?

Type Conversions

- This refers to the changing of one data type to another
- Two type conversions: *implicit* and *explicit*
- **Why not just store everything in the same format?**

Type Conversions

- This refers to the changing of one data type to another
- Two type conversions *implicit* and *explicit*
- **Why not just store everything in the same format?**

  Data can be stored in most compact form and converted when needed
- Disadvantage

  Type mismatches can be missed by the compiler

Implicit Conversions

- When types are mixed then type conversions that can be, are performed automatically.
- Remember that a **char** is just a small integer value and can be used in mathematical operations.
- There are some rules for these conversions:

Implicit Conversions

- **int** k=5, m=4, n; **float** x=1.5, y=2.1, z;

Context	Example	Explanation
Expression with binary operator and operands of different numeric types	k+x; value 6.5	Value of **int** variable k is converted to type **float** before operation
Assignment statement with **float** target and **int** expression	z=k/m; expression value is 1 value of z is 1.0	Expression is evaluated first. Result is converted to double for assignment
Assignment statement with **int** target and **float** expression	n=x*y; expression value is 3.15 value of n is 3	Expression is evaluated first. Result is converted to **int** for assignment. Fraction is lost.

Explicit Conversions (**Casting**)

- You can force the use of a particular type and the implicit conversion can be ignored
- Use a **cast** operator to convert the type
- Note the **cast** does not change a value stored in a variable, it just changes the type for the calculation.

```
int age;
age=1.2+5.978; age is 7
age=(int)1.2+(int)5.978; casts convert values first
age=?
```

Explicit Conversions (**Casting**)

- You can force the use of a particular type and the implicit conversion can be ignored
- Use a **cast** operator to convert the type
- Note the **cast** does not change a value stored in variable, it just changes the type for the calculation.

```
int age;
```

age=1.2+5.978; (equals 7.178) age is 7

age=(int)1.2+(int)5.978; **casts** convert values first

age=6

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;
}
```

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

Characters, Arrays and Strings

Characters and Arrays

- Characters

  Consist of any printable or non-printable character in the computers character set these include:

  - lowercase/uppercase letters
  - decimal digits
  - special characters
  - escape sequences.

  Generally stored in a single byte (8-bits)

  How to store words?

Characters and Arrays

- Characters

  Consist of any printable or non-printable character in the computers character set these include:

  - lowercase/uppercase letters
  - decimal digits
  - special characters
  - escape sequences.

  Generally stored in a single byte (8-bits)

  How to store words?

- Arrays

  Ordered sequences of same type data elements

  Simply put, several memory cells in a row given the same name.

Defining an Array

- How to declare an array:

```
char name[20];
```

- This declares an array called name with 20 elements (i.e. name can store 20 characters)
- Brackets **[]** indicate an array
- An array **MUST** have a dimension (number of elements)
- Other examples:

```
int attendance[15];
float  daily_temperature[30];
```

What do these mean??

Accessing Array Elements

- How can I use each element of my array
- The first element is always **0**
- **DO NOT** exceed array bounds (no one will check!)
- Consider:

```
float price[20]; /*declare an array 20 elements*/
price[0]=12.12; /*assign value to first element*/
price[1]=13.13; /*assign value to second element*/
…
price[20]=12.34; /*assign value to element 20?*/
```

What will happen here?

String Fundamentals

- A string is an array of characters ending with the NULL character or '**\0**'
- Can be written with double quotes "I am a string"
- Can be assigned when array is declared:
  - **char** firstname[] = "Walter";
  - **char** lastname[] = "White";
  - **char** fullname[] = "Walter White";
- **#define** can be used to create a string
  - **#define** TV "Breaking Bad"

String Fundamentals

- A string can also be defined by specifying individual characters:
  - **char** colour[] = {‘g’,’r’,’e’,’e’,’n’,’\0’};
  - This shows that each character is stored in its own element.

what would this print?

printf(“the fourth character is %c”,colour[4]);

String Library

- There is a library of string functions *string.h*
- Some examples:
  - strlen() – finds the length of a string
  - strcmp() – compares two strings character by character
  - strcpy() – copies a string from one array to another

```
#include<string.h>
#include<stdio.h>
#define NAME “Mark Leach”

main(){
printf(“My name has %d characters”,strlen(NAME));
}
```

C Fundamentals &
Basic Input/Output Functions

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 */ */
```

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 on the right hand side is evaluated first then assigned to a variable on the left hand side

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

Basic Input and Output

- Standard input functions (in stdio.h)
  - getchar() reads a character
  - scanf() input type must be specified
  - gets() reads strings
- Standard output functions (in stdio.h)
  - putchar() outputs a character
  - printf() output type must be specified
  - puts() outputs a string

The Function printf() (1/2)

It is used to output (usually to the screen)

Without argument

printf(“hello”); just prints letters hello

With arguments (expressions to evaluate)

The following common format specifiers are used

Type	Specifier
int	%d
float	%f
double	%lf
char	%c
string	%s

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;
}
```

The scanf() function (1/2)

- reads data from the standard input device *stdin* (usually the keyboard) and stores it in a variable

- General syntax:

  scanf("format specifier", &variable);

- The ampersand (**&**)
  - Specifies the memory address of the variable
- example:

```
int age;
printf("enter your age:");
scanf("%d",&age);
```

The scanf() function (2/2)

- The same common format specifiers used in printf() are used in scanf() functions

Type	Specifier
int	%d
float	%f
double	%lf
char	%c
string	%s

- Add more specifiers to enter more than one argument:

```
float height, weight;
scanf("%f%f",&height,&weight);
```

getchar() and putchar()

- Single character reading and writing, examples:

```
#include<stdio.h>
main(){
        char my_char;
        printf("please type a character: ");
        my_char=getchar();
        printf("\n you typed the character: ");
        putchar(my_char);
}
```

gets() and puts()

- Multiple character reading and writing
- scanf() reads strings using the specifier %s , however it cannot accept the space
- If the string to be read contains spaces, either multiple reads must be made in scanf() or the gets() function can be used

Example (1/2)

```
#include<stdio.h>
main()
{
char string1[50], string2[50];
printf("Enter a string less than 50 characters with spaces: \n");
gets(string1);
printf("you entered:");
puts(string1);
printf("Enter the same string\n");
scanf("%s",string2);
printf("you entered:");
puts(string2);
}
```

Example (2/2)

Sample output:

```
Enter a string less than 50 characters with spaces:
hello class
You entered: hello class
Enter the same string:
hello class
You entered: hello
```

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.

Next time

In lecture 3 next week we will be looking at:

- mathematical operations
- flow control

Thank you for your attention ☺

See you in the laboratory…