Lecture 2 C Basics

1. C program structure
2. Syntax
3. Data types
4. Variables
5. Constants

1. C program structure

- C source code program is organized as a sequence of functions.
  - A function contains a logic sequence of statements.
  - A statement may call another function, a function has to be declared or defined before it can be called.
  - Must contain *main()* function for an executable program. The execution of a C program starts from the *main()* function.
- Executable program is organized as a sequence of function blocks of machine code.

C program structure model

[preprocessor directives]

[global variables]

[function declarations]

main(arguments) {

[statements]

}

[function definitions]

```
/* C program structure example */
#include<stdio.h>       // preprocessor directive include
int a;                  // global variable declaration
int add(int, int);      // function declaration
int minus(int, int);    // function declaration
// definition of main function, the start function
int main()              // main function header
{                       // start of block function definition / function body
  a=1;                  // assign/set value 1 to global variable a
  int b=2;              // declare local variable b and initialize/set value 2
  printf("a+b=%d\n", add(a, b));   // function calls, output a+b=3
  printf("a-b=%d\n", minus(a, b)); // function calls, output a-b=-1
  return 0;
}                       // end of block function definition
// definition/implementation of function add(int, int)
int add(int x, int y)   // function header
{
   return x+y;          // function body
}
// definition/implementation of function minus(int, int)
int minus(int x, int y)  // function header
{
  return x-y;           // function body
}
```

C program organization

- A large C program is decomposed into
  - function header files
  - header function implementation files
  - a main function file, called driver program file

Refer to Lesson 1.2.4

2. Basic syntax

C has 5 types of elements:
symbol, keyword, expression, statement, function

- Basic symbols

Table 1: Basic symbols

Symbol	Description
//	line comment, compiler will ignore the line
/* ... */	block comment, compiler will ignore what in between
#	prepreprocessor
;	statement terminator
,	list separator
()	parenthesis of function parameter/argument list and algebraic expressions
{}	scope of a program block

- C has 32 reserved words (keywords)

Table 2: Keywords

Category		Keyword
Basic data types	(9)	char, int, float, double, short, long, signed, unsigned, void
Define data types	(4)	typedef, struct, union, enum
Modifiers	(6)	const, auto, static, extern, volatile, register
Flow control	(11)	if, else, switch, case, default, goto, for, while, do, break, continue
Function	(2)	return, sizeof

- Expressions
  - Use infix notation, consisting of constants, variables, operators, parenthesis.

  E.g., (1 + 2) * 3, 1==2, (1==1) && (2!=1)

- Statements
  - A C statement is a command/instruction to C compiler.
  - Statement types: declaration, assignment, condition, function call, flow control
  - Statements are organized to blocks (program **block)**, a sequence of statements scoped by { }.

- Function syntax

1. Function declaration/header syntax:

```
returndatatype function_name(argument type list);
```

2. Function definition/implementation syntax:

```
returndatatype function_name(argument type and name list)
{
  // function block
}
```

3 Function call syntax:

```
function_name(parameter list)
```

3. Data types

- A data type (or simply type) defines
  1. how a **certain type of data values** is **represented in programming.**
  2. **how many bytes and what bit pattern** are used to represent a value **in memory**.
  3. what and how **operations** are applied to the data values in programming and computers.

Brief description: data type defines how a certain type of data values are represented and operated in programming and computers.

Primary and derived data types

- C provides primary data types (primitive, basic data types)
  - Defined by keywords:
    char, int, short, long, float, double, signed, unsigned
  - Arithmetic operations (+, -, *, /) are defined for primary types. Modular operation % is defined for non-floating primary data types.
  - Each of the primary data types has corresponding bit pattern in representation and operations.

- C provides methods to build secondary data types (derived, extended) using primary data types and keywords typedef, struct, union, enum.

- Each data type has a **size**, i.e. **the number of bytes** to store the values of the type in memory.
- Each data type and has a defined valid **value range**.

Example: the **char** type has size 1, e.g., one byte (8 bits), value range `-128 to 127`

- The size of some data types is platform dependent.

Example: the int type has 2 bytes in old 16 system,
but 4 bytes in 32 and 64 bit system.

We use 4 bytes as default size for the int type.

Size and range of primary data types

DATA TYPE / Keyword	SIZE IN BYTES	RANGE
char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	4 (2)	$-2^{31}+1$ to $+2^{31}-1$ (-32768 to 32767)
unsigned int	4 (2)	0 to $2^{32}-1$ (0 to 65535)
signed short int	4 (2)	$-2^{31}+1$ to $+2^{31}-1$ (-32768 to 32767)
signed int	4 (2)	$-2^{31}+1$ to $+2^{31}-1$ (-32768 to 32767)
short int	2 (4)	-32768 to 32767 ($-2^{31}+1$ to $+2^{31}-1$)
unsigned short int	4 (2)	0 to $2^{32}-1$ (0 to 65535)
long int	4 (8)	$-2^{31}+1$ to $+2^{31}-1$ ($-2^{63}+1$ to $+2^{63}-1$)
unsigned long int	8 (4)	0 to $2^{64}-1$ (0 to 4294967295)
signed long int	8 (4)	$-2^{63}+1$ to $+2^{63}$ (-2147483648 to 2147483647)
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double (C99)	10	3.4E-4932 to 1.1E+4932

char type

- The char type is to present characters by an integer defined by ASCII (American Standard Code for Information Interchange). ASCII covers 128 characters, each is represented (encoded) by an integer between 0 and 127 in a **well-organized way.** Example: 0 is encoded by 48, A by 65, a by 97

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

How is char type represented/stored in memory?

- The binary presentation of ASCII code has at most 7 bits. In computer, each **addressable memory cell** holds 8 bits (1 byte) . The ASCII code of characters can be stored in one addressable memory cell.

Example: The ASCII code of 0 is 48

$48_{10} = 110000_2$ stored in memory cell as 00110000

The ASCII code of character A is 65.

$65_{10} = 100\ 0001_2$ stored as **0**100 0001

ASCII code of character a is 97

$97 = 65+32 = 100\ 0001_2 + 100000_2 = 1\mathbf{1}0\ 0001_2$ stored as 0**1**10 0001

How to do *conversions of number representation in different bases? Refer to Lesson 1.3.2*

- Unicode encoding standards:
  UTF-8 (Unicode Transmission Format 8-bit), UTF-16, UTF-32

int and unsigned int types

- int
  - Value range: from -2147483647= $-2^{31}+1$ to 2147483647= $2^{31}-1$
  - Bit pattern: 4 bytes or 32 bits, left most bit represent sign, 0 for positive, 1 for negative, the rest 31 bits represent the absolute value in base 2 (binary format).

Example

Int values	Binary
1	0000 0000 0000 0001
-1	1000 0000 0000 0001
-2147483647	1111 1111 1111 1111

- unsigned int
  - Value range: from 0 to 4294967295 = $2^{32}-1$
  - Bit pattern: 4 bytes or 32 bits, 32 bits represent the value in base 2.

Int values	Binary
1	0000 0000 0000 0001
4294967295	1111 1111 1111 1111

How int type is stored in memory?

- When a data type size is bigger than 1, it needs a contiguous memory cells (called memory block) to store the value of the type.
  - **Big-endian:** store the most significant byte in the lowest address cell
  - **Little-endian**: store the least significant byte in the lowest address cell.

  little-endian is commonly used.
- int type size is 4, needs 4 memory cells.

For example,

2427130573_{10}

$= 1001\ 0000\ 1010\ 1011\ 0001\ 0010\ 1100\ 1101_{2}$

$= \quad 9 \quad 0 \quad A \quad B \quad 1 \quad 2 \quad C \quad D_{16}$

Big-endian

Address	Value
1003	CD
1002	12
1001	AB
1000	90

Little-endian

Address	Value
1003	90
1002	AB
1001	12
1000	CD

float and double types

- float type uses 4 bytes for single precision floating point numbers; bit pattern and operations are specified by IEEE 754 standard.

https://en.wikipedia.org/wiki/Single-precision_floating-point_format

- double type uses 8 bytes for double precision floating point numbers, specified by IEEE 754 standard.

https://en.wikipedia.org/wiki/Double-precision_floating-point_format

4. Variables

- **Concepts of variables**

1. A variable is a name identifier used in source code program to represent a data value of a certain type.
2. A variable is assigned a memory block with relative address by compiler, as well as instructions to set and get the values to the memory block.
3. A variable is instanced at runtime with absolute address of memory block.

Brief description: a variable is an identifier of a data value in a program, it gets relative memory allocation at the compile time, and actual memory block at runtime.

C variables

- A variable must be declared with a type and name in a scope, and then used within the scope.
  1. The variable declaration tells compiler to assign memory block with relative address.
  2. A variable assignment statement tells compilers to generate instructions for writing values to the memory block.
  3. Using the variable in an expression tells compilers to generate instructions to read values from the memory block.
- **A variable should be initiated (assigned a value) before it is used in expressions.**
- C variable names must start with a letter, followed by letters, underscores and numbers, and case sensitive.
  - C name convention: underscore_style, camelCaseStyle

Variable and scope

1. A variable has to be declared before it can be used. A variable has a scope, within which the variable is declared and used.

2. Scope can be nested, i.e. one scope is inside another scope. A variable declared before an inner scope can be used in the inner scope. Same variable name can be used to declare and use as a new variable in an inner scope.

3. Global variables are variable declared outside any scope, so can be used anywhere.

4. Local variables are variables declared in a scope block embraced by {}. e.g. in a function, so can only be used in the scope block.

Literals

- Literals refer those constant values assigned to variables in programming.
- Compiler recognizes the data types of a literal and convents to its bit pattern representations, being used in generated instructions.
- Preprocessor #define is used to define a literal string as macro, then use the macro in programming. During the pre-preprocessing step, the macro will be replaced by its corresponding string.

Example

```
#define PI 3.1415926
float r = 4;
float area = PI*r*r;
float cf = 2*PI*r;
float f = 2.4e-5;    // 2.4e-5 = 0.000024
#define MAX(a,b)  ((a)>(b)? (a) : (b))   // function macro with parameters
```

Examples of variable declaration and initialization

Declaration

char c; // let compiler assign 1 byte memory space for char variable c
int a; // let compiler assign 4 bytes memory space for int variable a
float f; // let compiler assign 4 bytes memory space for float variable f

Assign values to variables

a = 2; // compiler generates instructions that store 2 to memory of variable a at runtime

c = 'a'; // compiler generates instructions that store 97_{10} = 0111001 to memory of c

f = 1.41; //compiler convert 1.41 to 32 bits single precision number, and generates instructions to store the number at memory of f.

Declaration & initialization

int a = 12; // or int a = 014; for Oct number 14, or int a = 0xC; for Hex number C

char c = 'a';

float f = 1.41;

int result, x = 9, y = 3; // a list separated by comma if variables are of the same type.

sizeof

- sizeof is a predefined keyword, a function macro, used to get the sizes of data types or variables, applying to all data types.

Example

sizeof(char) will be replaced by 1 , the size of a character data type

int a = 10;

sizeof(a) will be replaced by 4 during the preprocessing

5. Constants

- Constants are fixed data values in a program.
- In C, constant variable (or read-only variable) is used to represent constants. Constant variables are declared and initialized by keyword const. Compiler does not allow to assign values to a constant variable after it is declared and initialized .

Example

```
const float pi = 3.1415926;   // pi is a read only variable
float r = 4;
float area = pi*r*r;
float cf = 2*pi*r;
pi = 3.14;   // this is not allowed by compiler
```

- Symbolic constants are constant values defined by marco preprocessor. Example

  ```
  #define PI 3.1415926     // PI is a symbolic constant
  ```

-