C Programming
Lecture 4 : Variables , Data Types

Lecture notes : courtesy of Ohio Supercomputing Center, science and technolgy support

First Program

```
#include <stdio.h>
int main()
{
        /* My first program */
        printf("Hello World! \n");

        return 0;
}
```

```
Output :

 Hello World!
```

- C is case sensitive.
- End of each statement must be marked with a semicolon (;).
- Multiple statements can be on the same line.
- ***White space*** *(e.g. space, tab, enter, …)* is ignored.

First Program

```
#include <stdio.h>
int main()
{
        /* My first program */
        printf("Hello World! \n");

        return 0;
}
```

```
Output :

 Hello World!
```

- The C program starting point : **`main().`**
- **`main() {}`** indicates where the program actually starts and ends.
- In general, braces {} are used throughout C to enclose a block of statements to be treated as a unit.
- ***COMMON ERROR: unbalanced number of open and close curly brackets!***

First Program

```
#include <stdio.h>
int main()
{
       /* My first program */
       printf("Hello World! \n");

       return 0;
}
```

```
Output :

 Hello World!
```

- **#include <stdio.h>**
    - Including a header file `stdio.h`
    - Allows the use of `printf` function
    - For each function built into the language, an associated ***header file must be included.***

- **printf() is actually a function (procedure) in C that is used for printing variables and text**

First Program

```
#include <stdio.h>
int main()
{
       /* My first program */
       printf("Hello World! \n");

       return 0;
}
```

```
Output :

 Hello World!
```

- Comments
  - /* My first program */
  - Comments are inserted between “/*” and “*/”
  - Or, you can use “//”
  - Primarily they serve as *internal documentation for program structure and function*.

Why use comments?

- Documentation of variables, functions and algorithms
- Ex) for each function, explain input and output of the function, and **what** the function does.
- Describes the program, author, date, modification changes, revisions,…

Header Files

- Header files contain definitions of functions and variables
- Preprocessor `#include` insert the codes of a header file into the source code.
- Standard header files are provided with each compiler
- To use any of the standard functions, the appropriate header file should be included.
  - Ex) to use `printf()` function , insert `#include <stdio.h>`
- In UNIX, standard header files are generally located in the /usr/include subdirectory

Header Files

```
#include <string.h>
#include <math.h>
#include “mylib.h”
```

- The use of brackets <> informs the compiler to search **the compiler’s include directories** for the specified file.

- The use of the double quotes “” around the filename informs the compiler to **start the search in the current directory** for the specified file.

Second Program

```
#include <stdio.h>
#define TAXRATE 0.10
int main () {
       float balance;
       float tax=0.0;    /* declaration + initialization */
       char rate=‘A’;
       int credit_no=1;
       balance = 72.10;
       tax = balance * TAXRATE;
       printf("The tax on %.2f is %.2f\n",balance, tax);
       printf(“CREDIT RATE : %d/%c\n”, credit_no, rate);

       return 0;
}
```

```
Output :
The tax on 72.10 is 7.21
CREDIT RATE : 1/A
```

Names in C

- Identifiers (variable name)
  - Must begin with a character or underscore(_)
  - May be followed by any combination of characters, underscores, or digits(0-9)
  - Case sensitive
  - Ex) `summary, exit_flag, i, _id, jerry7`
- Keywords
  - Reserved identifiers that have predefined meaning to the C compiler. C only has 29 keywords.
  - Ex) `if , else, char, int, while`

Symbolic Constants

- Names given to values that cannot be changed.
- Use preprocessor directive #define

```
#define N 3000
#define FALSE 0
#define PI 3.14159
#define FIGURE "triangle"
```

- Symbols which occur in the C program are replaced by their value before actual compilation

Declaring Variables

- Variable
  - Named memory location where data value is stored
  - Each variable has a certain type (e.g. `int, char, float,` …)
  - Contents of a variable can change
  - Variables must be declared before use in a program
  - Declaration of variables should be done at the opening brace of a function in C. (it is more flexible in C++)

- Basic declaration format
  - *`data_type var1, var2, …;`*
  - *`Examples)`*

```
int i,j,k;
float length, height;
```

Data Types

- `char` : 1 byte, capable of holding one character (ascii code)
- `int  :` 4 byte (on 32bit computer) integer
- `float` : single-precision floating point
- `double` : double-precision floating point

type	size	min value	max value
char	1byte	$-2^{7} = -128$	$2^{7}-1 = 127$
short	2byte	$-2^{15} = -32,768$	$2^{15}-1 = 32,767$
int	4byte	$-2^{31} = -2,147,483,648$	$2^{31}-1 = 2,147,483,647$
long	4byte	$-2^{31} = -2,147,483,648$	$2^{31}-1 = 2,147,483,647$

- Min/Max values are defined in `<limit.h>` header file

unsigned type

- Use when representing only positive numbers

Data type	size	min	max
unsigned char	1byte	0	$2^{8}-1$ = 255
unsigned short	2 byte	0	$2^{16}-1$ = 65,535
unsigned int	4byte	0	$2^{32}-1$ = 4,294,967,295

Negative integer representation

- signed
- first bit represents the sign of a number
- Rest of bits represent the value of a number
- Negative integer number
  - Represented as 2’s complement

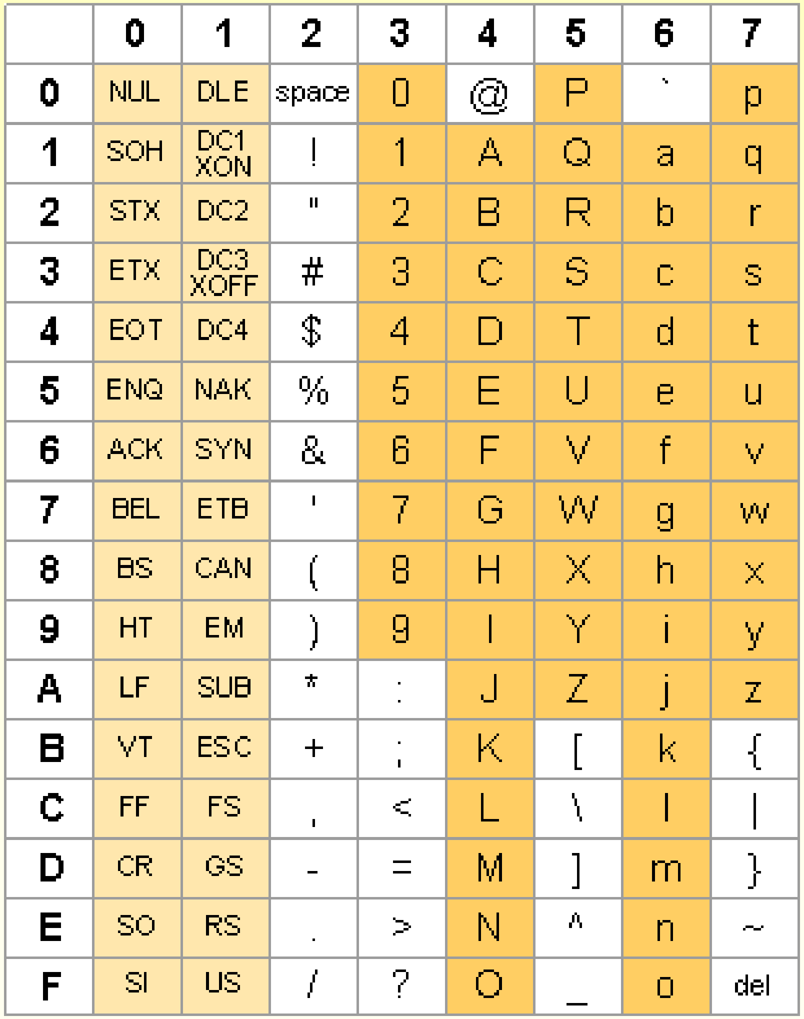
number	**Bit representation**
+5	00000101
1’s complement of 5	11111010
2’s complement of 5	11111011
-5	11111011

floating point

- real number : significant number + position of decimal point
- Decimal point(.) can be placed anywhere relative to the significant digits of the number
- This position is indicated separately in the internal representation
- Advantage of floating point representation
  - Support much wider range of values
  - Representing 314159265358979.3 vs 3.141592653589793

type	size	min	max
float	4 byte	(7 significant numbers) -1.0E+38	(7 significant numbers) 1.0E+38
double	8 byte	(15 significant numbers) -1.0E+308	(15 significant numbers) 1.0E+308

Ascii Code



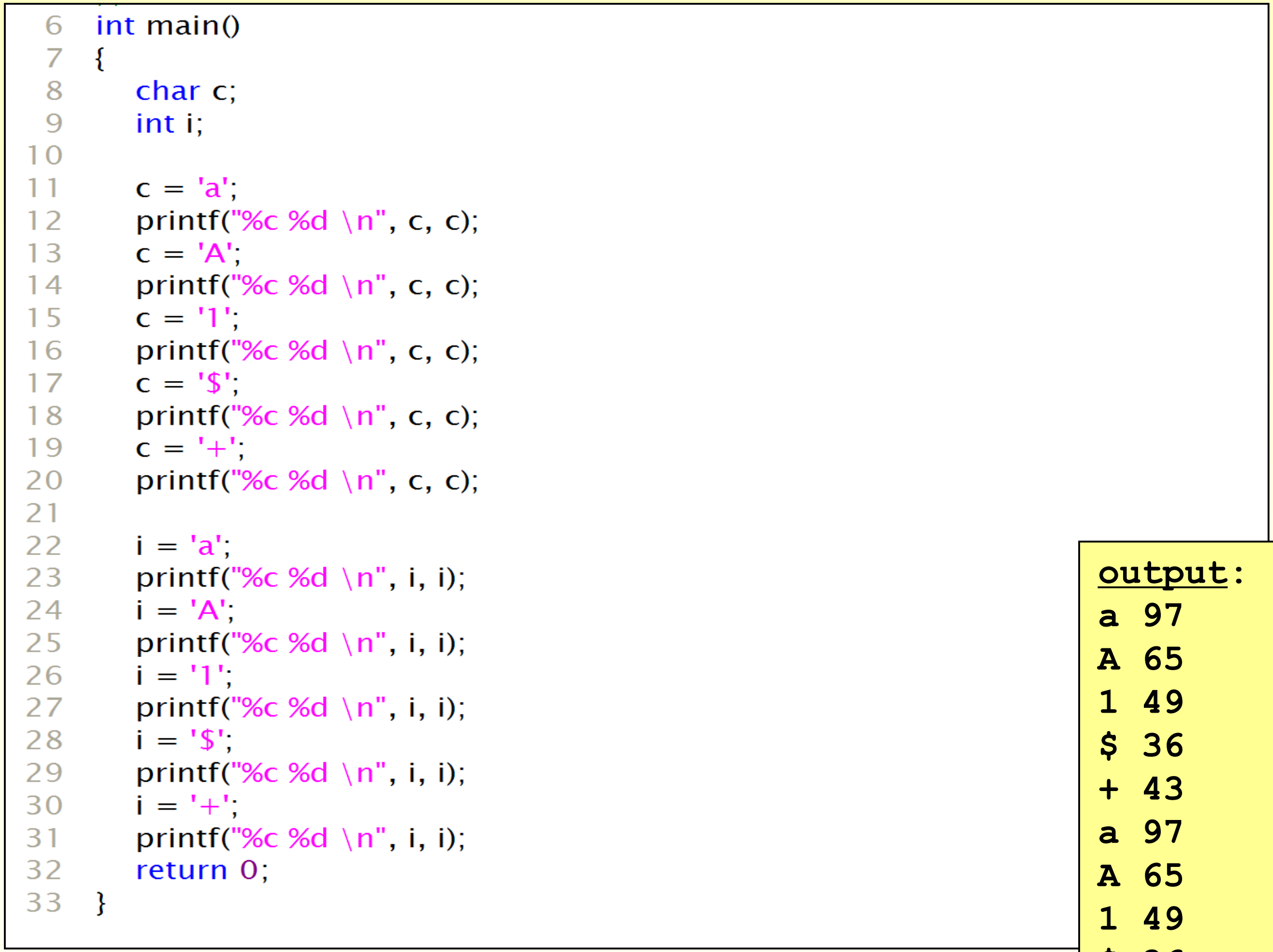
	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Escape character

- Starts with backslash(\)
- Indicate special meaning and interpretation

Escape character	meaning
\b	backspace
\t	tab
\n	newline
\r	formfeed
\"	double quote
\'	single quote
\\	back slash

code.c



```c
int main()
{
    char c;
    int i;

    c = 'a';
    printf("%c %d \n", c, c);
    c = 'A';
    printf("%c %d \n", c, c);
    c = '1';
    printf("%c %d \n", c, c);
    c = '$';
    printf("%c %d \n", c, c);
    c = '+';
    printf("%c %d \n", c, c);

    i = 'a';
    printf("%c %d \n", i, i);
    i = 'A';
    printf("%c %d \n", i, i);
    i = '1';
    printf("%c %d \n", i, i);
    i = '$';
    printf("%c %d \n", i, i);
    i = '+';
    printf("%c %d \n", i, i);
    return 0;
}
```

```
output:
a 97
A 65
1 49
$ 36
+ 43
a 97
A 65
1 49
$ 36
+ 43
```

getchar() , putchar()

- **int getchar()**
  - Defined in <stdio.h>,
  - Get one character input from keyboard and return the ascii value
- **int putchar(int c)**
  - Defined in <stdio.h>
  - prints one character provided as a parameter

```
#include <stdio.h>

int main()
{
          int c;

          printf(“keyboard input (one character?)”);

          c=getchar();

          printf(“character input : %c\n”,c);
          printf(“ascii code : %d\n”, c);

          return 0;
}
```

Output :

```
character input : A
ascii code : 65
```

korea.c

```
#include <stdio.h>

int main()
{
   short no_univ = 276;
   int population = 48295000;
   long budget = 2370000000000000L;

   printf(“korea info\n”);
   printf(“univ no : %d\n”, no_univ);
   printf(“population : %d\n”, population);
   printf(“budget : %d\n”, budget);

   return 0;
}
```

Output :

```
korea info
univ no : 276
putpulation:   48295000
budget:   -590360576
```

Overflow?

- (integer type) overflow
  - occurs when storing a value that is bigger than what can be stored.

  - Ex) 2,147,483,647 (= 2^{31}-1) + 1 = ?

```
      01111111 11111111 11111111 11111111
    + 00000000 00000000 00000000 00000001
  -------------------------------------------
      10000000 00000000 00000000 00000000
```

```
#include <stdio.h>

int main()
{
        int a=2147483647;

        printf("%d,%d\n",a,a+1);
        return 0;
}
```