2008 summer course, C-language Homework 2

Exercise 1 (2's complement): we have shown that signed integer of 2's complement under size = 4-bits ranges from -8 to 7, the complete mapping between decimal and 2's complement is

decimal	2's complement (binary)	decimal	2's complement (binary)
0	0 0 0 0	-8	1 0 0 0
1	0 0 0 1	-7	1 0 0 1
2	0 0 1 0	-6	1 0 1 0
3	0 0 1 1	-5	1 0 1 1
4	0 1 0 0	-4	1 1 0 0
5	0 1 0 1	-3	1 1 0 1
6	0 1 1 0	-2	1 1 1 0
7	0 1 1 1	-1	1 1 1 1

Now answer the following questions

(1) 2's complement representation of 0 is unique

(2) for general $n-bits$, signed integer of 2's complement ranges from -2^{n-1} to $2^{n-1}-1$.

(3) Suppose integer is of size 4-bits, a number x has 2's complement representation $x_{2's}=1010$, how can we know its decimal value? (Hint: do 2's complement for $x_{2's}$ again)

Exercise 2: read page 153~155 in textbook to know conversion specification, we usually use

character	Argument type ; printed as
d	int; decimal number
c	int ; single character
s	char * ; print characters from the string until a '\0'
f	double, fixed representation
e,E	double, scientific representation

Exercise 3 (limit of integer type): use Visual Studio to create a project, named as **limit_test**, and write codes to test size and limits of data type in **Table 1**. You can refer codes in Figure 1. Explain how do you why the minimum value you fill-in is actual minimum.

Table 1: fill-in the blanks by writing code in your computer, note that data type is implementation defined, just take data in MSDN library as reference, you must confirm them in your computer.

Case 1: your PC

CPU =

OS (operating system) =

Compiler =

Type	Bytes	Minimum value	Maximum value
(signed) short (int)			

(signed) int			
(signed) long (int)			
unsigned short			
unsigned int			
unsigned long			

```
#include <stdio.h>
#include <limits.h>

int main( int argc, char *argv[] )
{
    short    x_sint ;  // x_sint is signed short integer

    printf("size of short = %d bytes\n", sizeof( short ) );

    x_sint = SHRT_MAX ;  // SHRT_MAX is defined in file limits.h

    printf("maximum of short = %d\n", x_sint );

    x_sint = SHRT_MAX + 1 ;

    printf("maximum(short) + 1 = %d\n", x_sint );

    printf("maximum(short) + 1 = %d\n", SHRT_MAX + 1 );

    return 0 ;
}
```

Figure 1: codes to test size and limits of data type “short”.

Case 2: workstation

CPU =

OS (operating system) =

Compiler =

Type	Bytes	Minimum value	Maximum value
(signed) short (int)			
(signed) int			
(signed) long (int)			
unsigned short			
unsigned int			
unsigned long			

Exercise 4 (escape sequence): read section 2.3 (page 37~38) to know escape sequence and use Visual Studio to create a project, named as **char_test**, and write codes to find out integral value of escape sequence, like Figure 2.

(1) What is execution result, are the results consistent with ASCII table?

(2) If we change NUM_ESCAPE_CHAR = 6 and sweep 12 elements in array “word”, like Figure 3. Compare these results with results of (1), what’s the difference? Can you find out potential bug of this code?

```c
#include <stdio.h>

#define  NUM_ESCAPE_CHAR  12

int main( int argc, char* argv[])
{
    int i ;
    char  word[ NUM_ESCAPE_CHAR ] ;

    word[0] = '\a' ; word[1] = '\b' ; word[2] = '\f' ; word[3] = '\n' ;
    word[4] = '\r' ; word[5] = '\t' ; word[6] = '\v' ; word[7] = '\\' ;
    word[8] = '\?' ; word[9] = '\'' ; word[10] = '\"' ; word[11] = '\0' ;

    for ( i = 0 ; i < NUM_ESCAPE_CHAR ; i++){
        printf("%c = 0x%x\n", word[i] , word[i] );
    }

    return 0 ;
}
```

Figure 2: display character and hexadecimal value of escape sequence.

```c
#include <stdio.h>

#define  NUM_ESCAPE_CHAR  6

int main( int argc, char* argv[])
{
    int i ;
    char  word[ NUM_ESCAPE_CHAR ] ;

    word[0] = '\a' ; word[1] = '\b' ; word[2] = '\f' ; word[3] = '\n' ;
    word[4] = '\r' ; word[5] = '\t' ; word[6] = '\v' ; word[7] = '\\' ;
    word[8] = '\?' ; word[9] = '\'' ; word[10] = '\"' ; word[11] = '\0' ;

    for ( i = 0 ; i < 12 ; i++){
        printf("%c = 0x%x\n", word[i] , word[i] );
    }

    return 0 ;
}
```

Figure 3: array size is declared too small to contains all escape sequences.

Exercise 5 (string constant): use Visual Studio to create a project, named as **string_const**, and write codes to find relationship between string constant and character array, like

```c
#include <stdio.h>

int main( int argc, char* argv[])
{
    char p1[] = "hello, world" ; /* compiler would decide size of p1 */

    printf("%s\n", p1 );  /* %s : print string */

    p1[12] = '!' ; /* remove \0 of p1 */

    printf("%s\n", p1 );

    return 0 ;
}
```

Figure 4: string constant versus character array.

(a) why second “printf” does not print “hello, world!”, could you explain this?

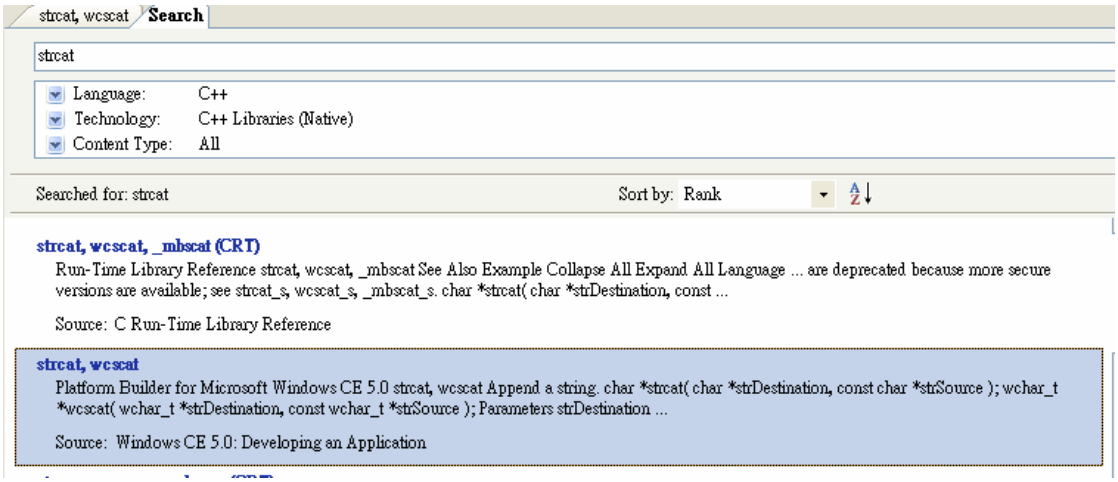
(b) May you modify second “printf” such that you can print “hello, world!”? Hint: you can print character by character if you know size of the string.

(c) Why we need string terminator ‘\0’ ?

(d) Read A2.6 in page 194

Exercise 6 (string concatenation): use Visual Studio to create a project, named as **strcat_test**,

and search for “strcat” in MSDN library, find theme “strcat,wcscat” as following figure, copy the source code in that theme, see Figure 5, verify the result.



Example

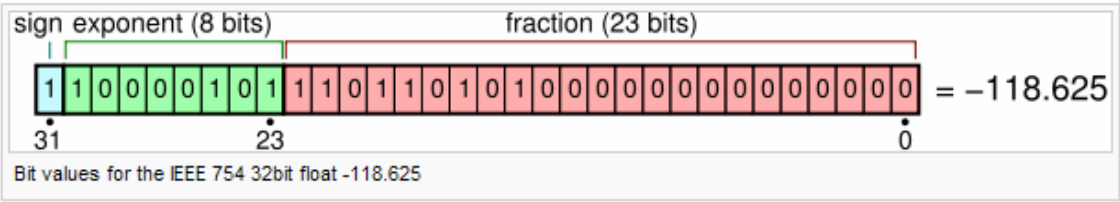
```
/* STRCPY.C: This program uses strcpy
 * and strcat to build a phrase.
 */

#include <string.h>
#include <stdio.h>

void main( void )
{
   char string[80];
   strcpy( string, "Hello world from " );
   strcat( string, "strcpy " );
   strcat( string, "and " );
   strcat( string, "strcat!" );
   printf( "String = %s\n", string );
}
```

Figure 5: example code in theme “strcat,wcscat”

How about if we modify statement “char string[80];” to “char string[**10**];”? Can you explain the execution result?

Exercise 7 (convert single precision binary format to decimal value)

convert following binary representation into normalized decimal value $v = s \times 2^E \times m$



Exercise 8 (transformation between string and integral/floating)

In the course, we draw a picture to show transformation between string and integral/floating as

Figure 6, we know that convert string to integral/floating is easy since standard C support library function atoi, atof, atol to do this.

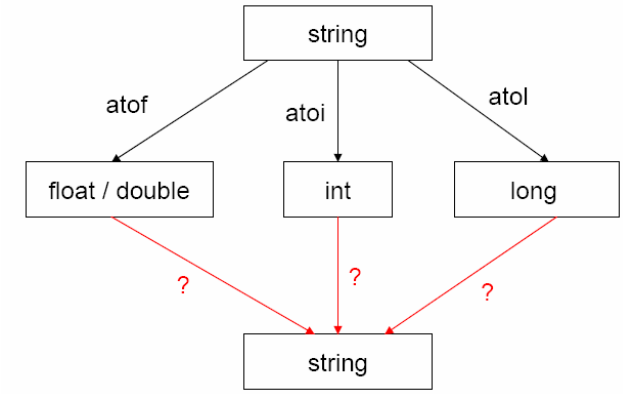


Figure 6: convert string to integral/floating is easy by standard library.

(1) use MSDN library to search for atoi, atof and atol, and test example shown in MSDN library, what do you learn in these examples.
(2) Can you implement the converse way, that is, convert integral/floating to a string?

Exercise 9 (potential bug of equality operator): Test the code in Figure 7, C-language accept such coding style, try to invert "x = 1" to "1 = x", what is compilation error in g++ and icpc?

```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int  x = 5 ;

    if ( x = 1 ){
        printf("x (=%d) is equal to 1\n", x );
    }else{
        printf("x (=%d) is NOT equal to 1\n", x );
    }

    return 0 ;
}
```

Figure 7: x == 1 or 1 == x, which one is better

Exercise 10 (bitwise operator): consider 8-bit operation,
(1) $a = 'a'$ and $b = -1$ what is $a \& b$, $a \mid b$ and $a \wedge b$? write a program to demonstrate this and interpret the result.
(2) Can you use AND operation to implement modulo operation, for example a % 4?

Exercise 11 (shift operator): in the course, we define type of ***a_left_shift_1*** (which is a <<1) as **int**, why? Can we define it as **char** as you see in Figure 8, what's the difference?

```c
#include <stdio.h>

int main( int argc, char *argv[] )
{
    char a = 'a' ;
    char a_right_shift_1 = a >> 1 ;
    char a_left_shift_1  = a << 1 ;

    printf("      a = %p\n", a ) ;
    printf(" a >> 1 = %p\n", a_right_shift_1 ) ;
    printf(" a << 1 = %p\n", a_left_shift_1 ) ;

    return 0 ;
}
```

Figure 8: define ***a_left_shift_1*** as **char.**

Exercise 12 (shift operator versus multiplication): Can you use shift operator to implement multiplication or division on an integer number? Write program to test your idea.

Exercise 13 (type conversion): try all combination of rules of type conversion in textbook, do you thank that explicitly casting done by program himself is a good habit?