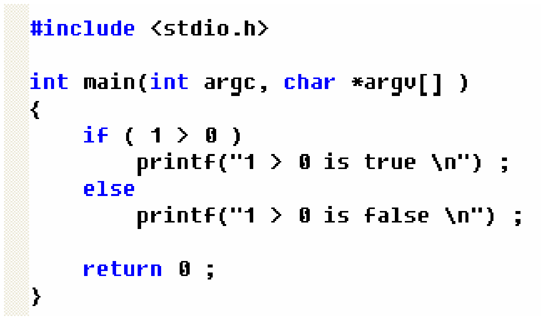
2008 summer course, C-language Homework 3

Exercise 1 (parsing tree): find the parsing tree of

(1) sizeof(int)

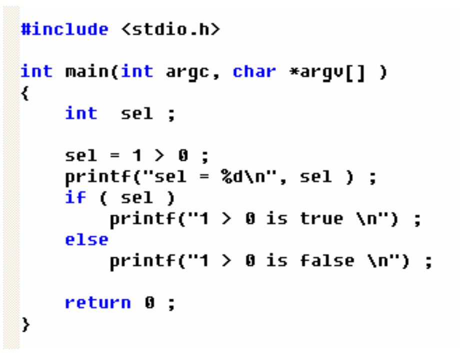
(2)
$$\begin{array}{l} for\left(i=0\ ;\ i<5\ ;\ i++\right)\{ \\ \qquad a[i]=i \\ \} \end{array}$$

Exercise 2 (if-else statement):



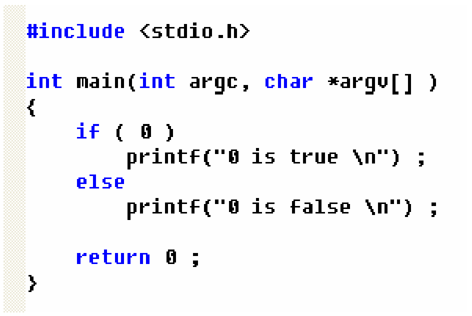
```
#include <stdio.h>

int main(int argc, char *argv[] )
{
    if ( 1 > 0 )
        printf("1 > 0 is true \n") ;
    else
        printf("1 > 0 is false \n") ;

    return 0 ;
}
```

(a)



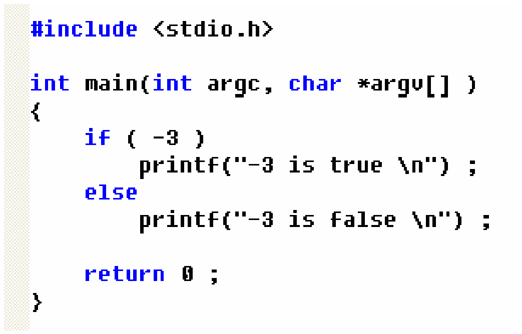
```
#include <stdio.h>

int main(int argc, char *argv[] )
{
    int  sel ;

    sel = 1 > 0 ;
    printf("sel = %d\n", sel ) ;
    if ( sel )
        printf("1 > 0 is true \n") ;
    else
        printf("1 > 0 is false \n") ;

    return 0 ;
}
```

(b)



```
#include <stdio.h>

int main(int argc, char *argv[] )
{
    if ( 0 )
        printf("0 is true \n") ;
    else
        printf("0 is false \n") ;

    return 0 ;
}
```

(c)



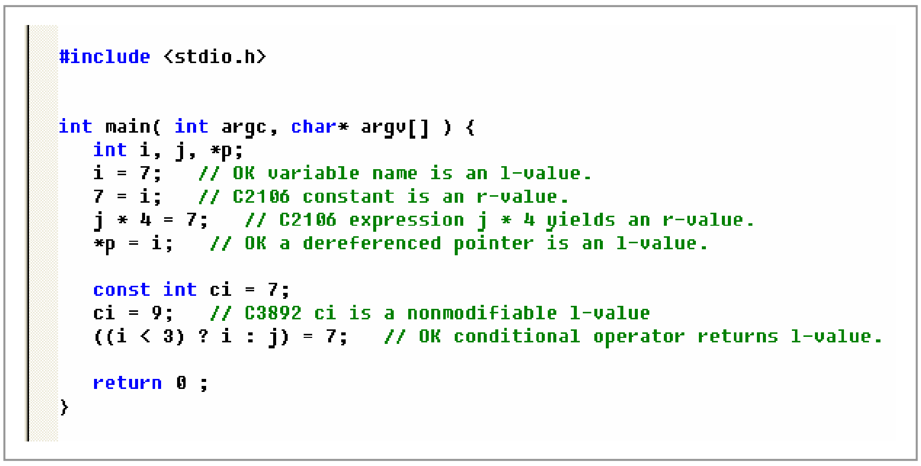
```
#include <stdio.h>

int main(int argc, char *argv[] )
{
    if ( -3 )
        printf("-3 is true \n") ;
    else
        printf("-3 is false \n") ;

    return 0 ;
}
```

(d)

(1) try above 4 kinds of if-else statement

(2) If we replace integer by floating point, then do results change?

Exercise 3 (L-value): create a project named **Lvalue_test** and copy source code in theme "L-values and R-Values" of MSDN Library into your project, see Figure 1.

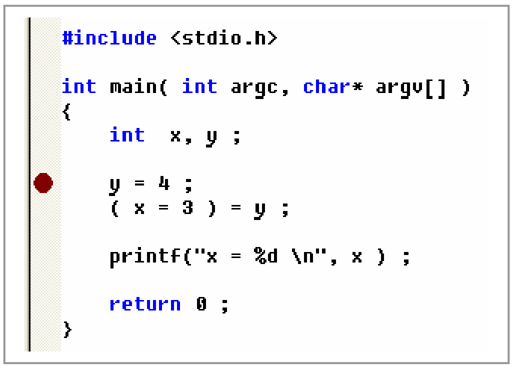


```c
#include <stdio.h>

int main( int argc, char* argv[] ) {
   int i, j, *p;
   i = 7;   // OK variable name is an l-value.
   7 = i;   // C2106 constant is an r-value.
   j * 4 = 7;   // C2106 expression j * 4 yields an r-value.
   *p = i;   // OK a dereferenced pointer is an l-value.

   const int ci = 7;
   ci = 9;   // C3892 ci is a nonmodifiable l-value
   ((i < 3) ? i : j) = 7;   // OK conditional operator returns l-value.

   return 0 ;
}
```

Figure 1: example code in MSDN Library with theme “L-values and R-Values”

(1) Compile it, then check error messages.

Exercise 4 (L-value test): Implement source code of example 6 in power point file, see Figure 2.



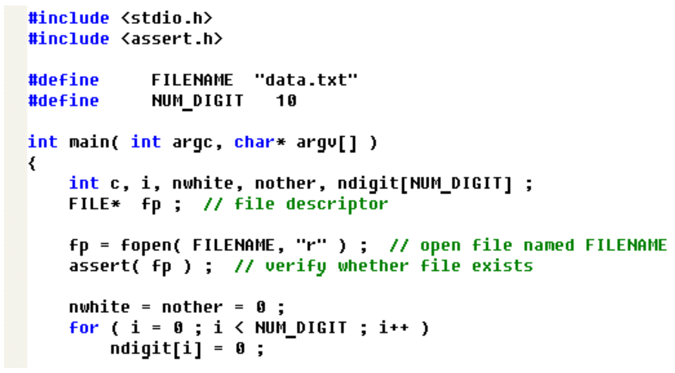
```c
#include <stdio.h>

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

    y = 4 ;
    ( x = 3 ) = y ;

    printf("x = %d \n", x ) ;

    return 0 ;
}
```

Figure 2: L-value issue of assignment expression, set break point at statement “y = 4 ;”

Follow steps in power point file, use debugger to trace source codes in assembly format.

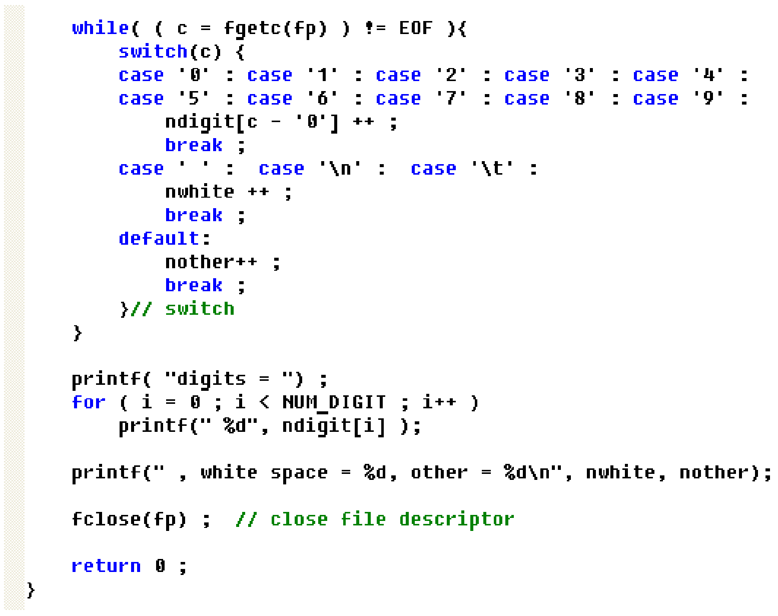
Remark 1: Although Visual Studio can show you assembly code of your C source code, as you see, assembly code is more atomic, cumbersome and the code size is larger than C source code.

Exercise 5 : create a project named **switch_test,** write codes in page 59 of textbook as Figure 3, and create a text file **data.txt** (you can edit it by 記事本) which has content **0123456789\nabcdefg\n,** put data.txt in directory **switch_test.**

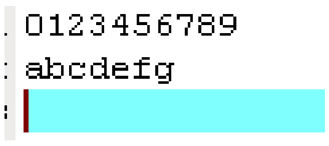


```c
#include <stdio.h>
#include <assert.h>

#define     FILENAME  "data.txt"
#define     NUM_DIGIT   10

int main( int argc, char* argv[] )
{
    int c, i, nwhite, nother, ndigit[NUM_DIGIT] ;
    FILE*  fp ;  // file descriptor

    fp = fopen( FILENAME, "r" ) ;  // open file named FILENAME
    assert( fp ) ;  // verify whether file exists

    nwhite = nother = 0 ;
    for ( i = 0 ; i < NUM_DIGIT ; i++ )
        ndigit[i] = 0 ;
```

Figure 3: example code in page 59 of textbook, this example shows while-loop, for-loop and switch-case statement.



```c
    while( ( c = fgetc(fp) ) != EOF ){
        switch(c) {
        case '0' : case '1' : case '2' : case '3' : case '4' :
        case '5' : case '6' : case '7' : case '8' : case '9' :
            ndigit[c - '0'] ++ ;
            break ;
        case ' ' :  case '\n' :  case '\t' :
            nwhite ++ ;
            break ;
        default:
            nother++ ;
            break ;
        }// switch
    }

    printf( "digits = ") ;
    for ( i = 0 ; i < NUM_DIGIT ; i++ )
        printf(" %d", ndigit[i] );

    printf(" , white space = %d, other = %d\n", nwhite, nother);

    fclose(fp) ;  // close file descriptor

    return 0 ;
}
```



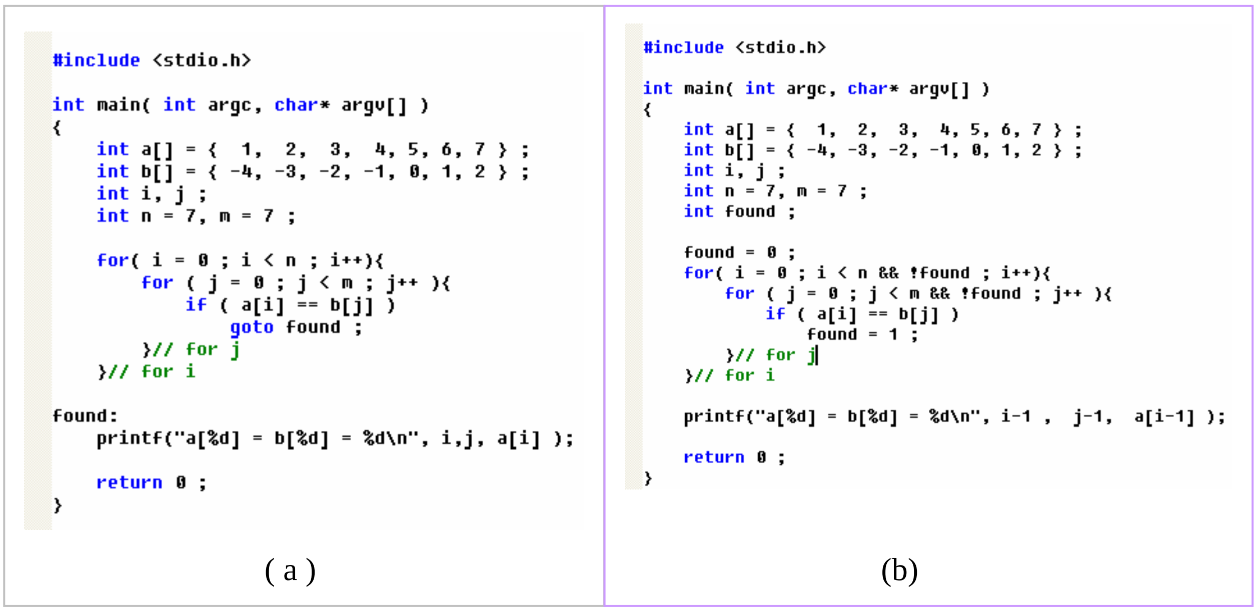
```
0123456789
abcdefg
```

data.txt

(1) modify the switch-case statement to if-then-else statement as you see in page 22 of textbook, and verify if results are the same. Which coding style is better?

(2) Add some codes to show numerical value of macro **EOF**

Exercise 6 (goto): Write codes (a) and (b) in Figure 4 respectively.



```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int a[] = {  1,  2,  3,  4, 5, 6, 7 } ;
    int b[] = { -4, -3, -2, -1, 0, 1, 2 } ;
    int i, j ;
    int n = 7, m = 7 ;

    for( i = 0 ; i < n ; i++){
        for ( j = 0 ; j < m ; j++ ){
            if ( a[i] == b[j] )
                goto found ;
        }// for j
    }// for i

found:
    printf("a[%d] = b[%d] = %d\n", i,j, a[i] );

    return 0 ;
}
```

(a)

```
#include <stdio.h>

int main( int argc, char* argv[] )
{
    int a[] = {  1,  2,  3,  4, 5, 6, 7 } ;
    int b[] = { -4, -3, -2, -1, 0, 1, 2 } ;
    int i, j ;
    int n = 7, m = 7 ;
    int found ;

    found = 0 ;
    for( i = 0 ; i < n && !found ; i++){
        for ( j = 0 ; j < m && !found ; j++ ){
            if ( a[i] == b[j] )
                found = 1 ;
        }// for j
    }// for i

    printf("a[%d] = b[%d] = %d\n", i-1 ,  j-1,  a[i-1] );

    return 0 ;
}
```

(b)

Figure 4: (a) use goto and label **found**, whereas (b) use **found** as flag

(1) check the results are the same

(2) why in (b), we say $a[i-1]=b[j-1]$, not $a[i]=b[j]$ as in (a), you can use debugger to find the reason.

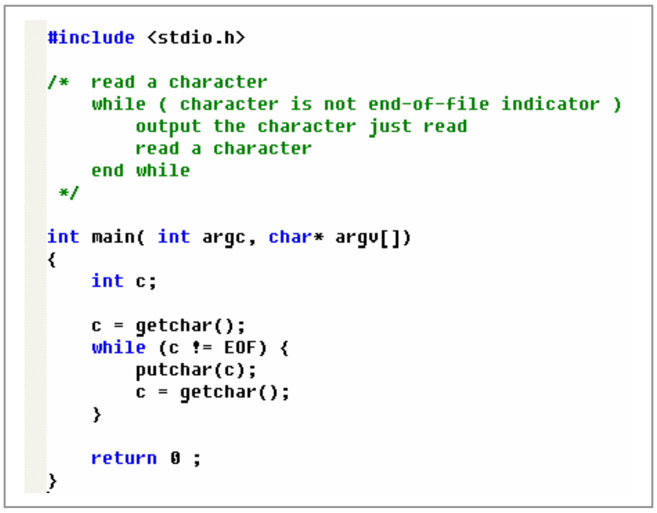
Exercise 7 (EOF versus stdin): in page 16, the authors provide a program for file copying, its pseudo-code is

```
read a character
while ( character is not end-of-file indicator)
    output the character just read
    read a character
end while
```

Moreover the author provide source code like Figure 5.
Now create a project named **filecopy**, and write source code in **main.cpp**,

(1) execute it in visual studio IDE, why enter infinite loop? Hint: you can use debugger to see what happens.

(2) Upload file **filecopy** to workstation and compile the source code (to **a.out**), then use command
[imsl@linux filecopy]$ ./a.out < main.cpp
what is the result?



```
#include <stdio.h>

/*  read a character
    while ( character is not end-of-file indicator )
        output the character just read
        read a character
    end while
 */

int main( int argc, char* argv[])
{
    int c;

    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }

    return 0 ;
}
```

Figure 5: source code of file copying

(3) issue command

[imsl@linux filecopy]$ ./a.out < main.cpp   > output.txt

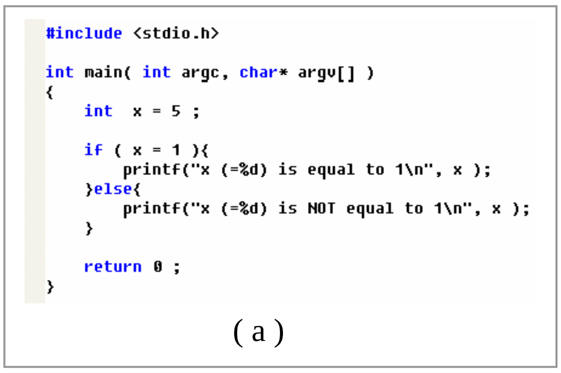
[imsl@linux filecopy]$ cat output.txt

what is the result?

Remark 2: **“./a.out < main.cpp   > output.txt”** means feed file main.cpp into a.out and transfer output of a.out to file output.txt

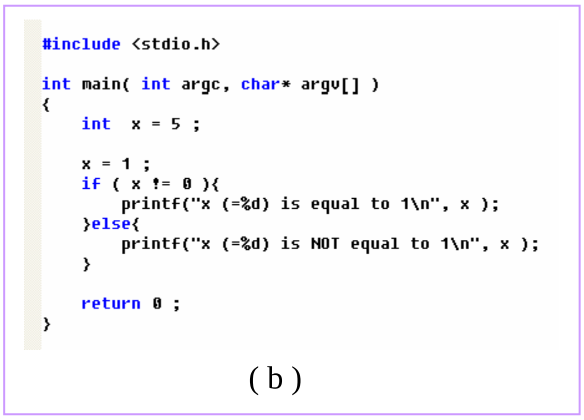
(4) If you want to input EOF in standard input (由鍵盤輸入), not from a file, then in Windows下為組合鍵Ctrl+Z, Linux下為組合鍵Ctrl+D. see http://g.51cto.com/procedures/837 and test these two special characters, Ctrl+Z and Ctrl+D.

Exercise 8 (potential bug of equality operator): in chapter 2, we show that “1 = = x” is better than “x = = 1”, since if typing error occurs, then compiler can help us to find out error if we code it as “1 = = x”. Also we argue that code (a) is the same as code (b), could you use debugger and choose assembly format to confirm that “code (a) is the same as code (b)”.



```
#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 ;
}
```

(a)



```
#include <stdio.h>

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

    x = 1 ;
    if ( x != 0 ){
        printf("x (=%d) is equal to 1\n", x );
    }else{
        printf("x (=%d) is NOT equal to 1\n", x );
    }

    return 0 ;
}
```

(b)