2008 summer course C-language midterm

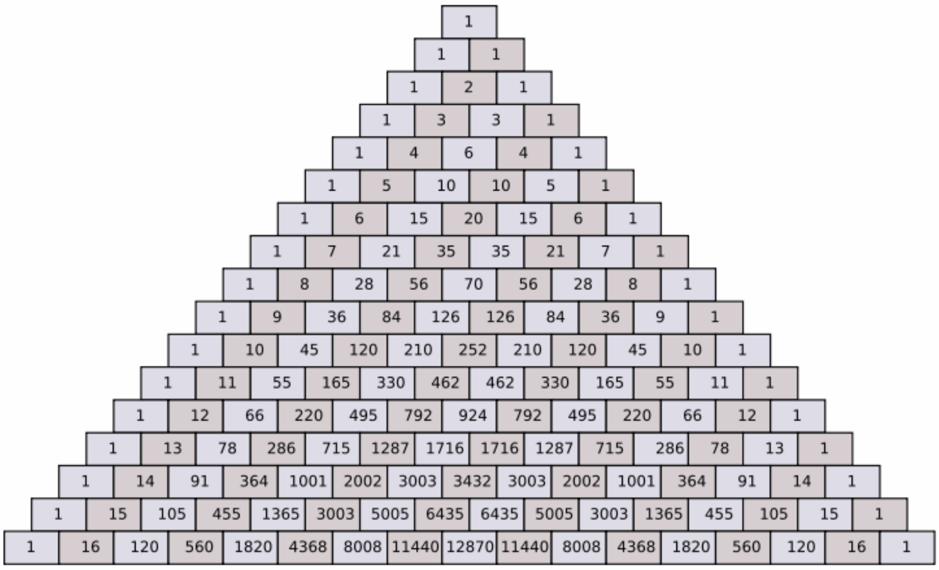
Date : 2008/7/14 ~ 2008/7/27

hand over ***electronic paper*** (don't hand over handwriting paper) on Monday, 28 July.

You can edit your result in this document file directly.

Exercise 1 (Pascal's triangle): In mathematics, Pascal's triangle is a geometric arrangement of the binomial coefficients in a triangle. This construction is related to the binomial coefficients by Pascal's rule, with states if $C_k^n = \dfrac{n!}{k!(n-k)!}$ is the k-th binomial coefficinet in the binomial expansion of $(x+y)^n$, then $C_k^n = C_{k-1}^{n-1} + C_k^{n-1}$ for any $n \ge 0$ and $k = 0,1,2,\cdots,n$.

Write a C-code to produce Pascal's triangle like Figure 1 in a file.



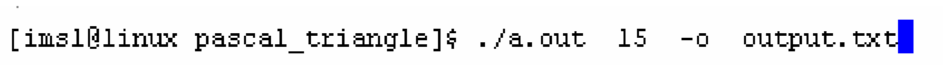
```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
```

Figure 1: Pascal's triangle of $n = 16$

Program requirement:

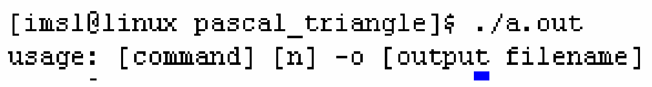
(1) read n and output filename from command

[command] [n] –o [output filename]



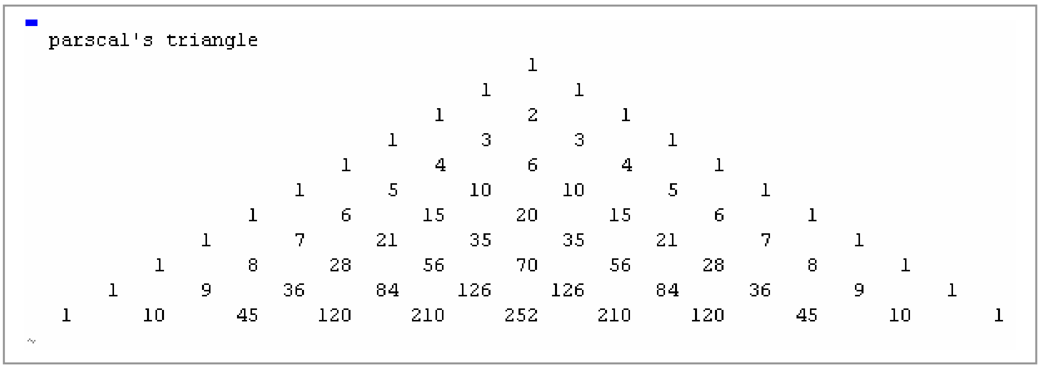
```
[imsl@linux pascal_triangle]$ ./a.out  15  -o  output.txt
```

if user does not match the format, then output correct usage,



```
[imsl@linux pascal_triangle]$ ./a.out
usage: [command] [n] -o [output filename]
```

(2) output file has the format as Figure 2.



```
parscal's triangle
                                         1
                                     1       1
                                 1       2       1
                             1       3       3       1
                         1       4       6       4       1
                     1       5      10      10       5       1
                 1       6      15      20      15       6       1
             1       7      21      35      35      21       7       1
         1       8      28      56      70      56      28       8       1
     1       9      36      84     126     126      84      36       9       1
 1      10      45     120     210     252     210     120      45      10       1
```

Figure 2: Pascal's triangle when $n = 10$

(3) you need to test $n = 0,1,2,3,4$ at least

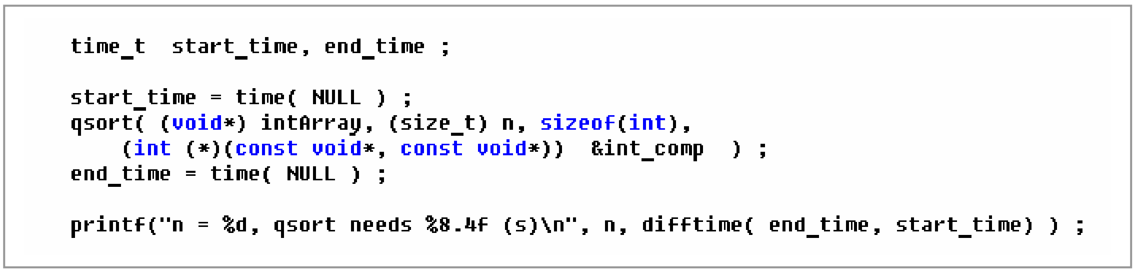
Exercise 2 (asymptotic behavior of sorting): in the course, we introduce quick sort (qsort in stdlib.h) and bubble sort (written by speaker), now we want to evaluate these two sorting algorithms. We need a timer to record cost of sorting, we use two functions, ***time*** and ***difftime*** in ***time.h***, the usage of these two functions are described in Figure 3.

time_t *time* (*time_t* *tp)

time returns the current calendar time or -1 if the time is not available. If tp is not NULL, the return value is also assigned to *tp

double **difftime** (*time_t* time2, *time_t* time1)

difftime returns time2 – time1 expressed in seconds.



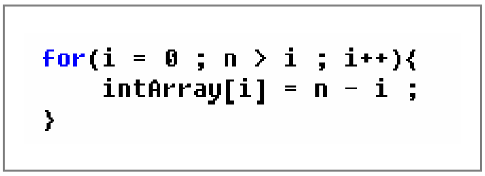
```
time_t  start_time, end_time ;

start_time = time( NULL ) ;
qsort( (void*) intArray, (size_t) n, sizeof(int),
    (int (*)(const void*, const void*))  &int_comp   ) ;
end_time = time( NULL ) ;

printf("n = %d, qsort needs %8.4f (s)\n", n, difftime( end_time, start_time) ) ;
```

Figure 3: usage of timer

In order to evaluate worst case, we create an integer array with n elements and arrange the array into descending order and then quick sort (bubble sort) would sort it into ascending order.



```
for(i = 0 ; n > i ; i++){
    intArray[i] = n - i ;
}
```

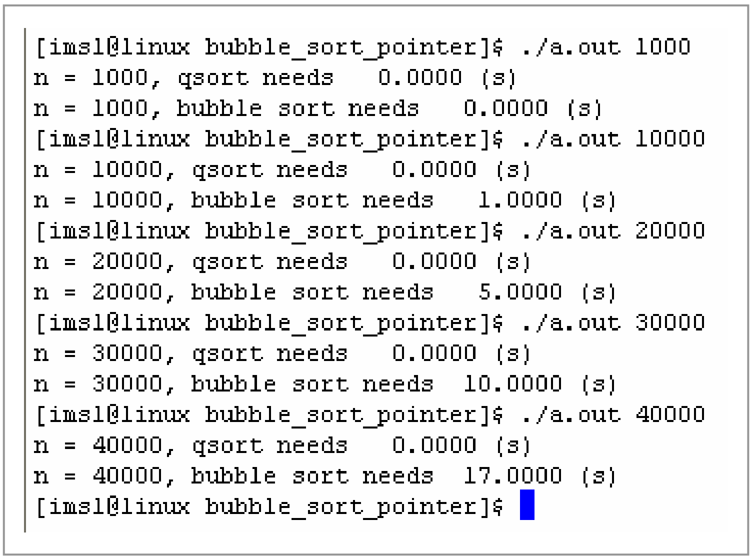
Figure 4: create an integer array with reverse order

Program requirement:

(1) read n from command

[command] [n]

(2) report time of quick sort and bubble sort like

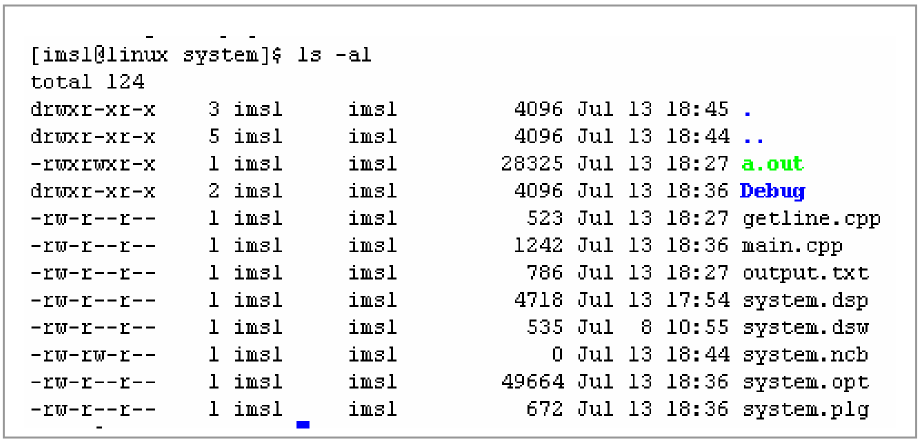


```
[imsl@linux bubble_sort_pointer]$ ./a.out 1000
n = 1000, qsort needs   0.0000 (s)
n = 1000, bubble sort needs   0.0000 (s)
[imsl@linux bubble_sort_pointer]$ ./a.out 10000
n = 10000, qsort needs   0.0000 (s)
n = 10000, bubble sort needs   1.0000 (s)
[imsl@linux bubble_sort_pointer]$ ./a.out 20000
n = 20000, qsort needs   0.0000 (s)
n = 20000, bubble sort needs   5.0000 (s)
[imsl@linux bubble_sort_pointer]$ ./a.out 30000
n = 30000, qsort needs   0.0000 (s)
n = 30000, bubble sort needs  10.0000 (s)
[imsl@linux bubble_sort_pointer]$ ./a.out 40000
n = 40000, qsort needs   0.0000 (s)
n = 40000, bubble sort needs  17.0000 (s)
[imsl@linux bubble_sort_pointer]$ 
```

(3) find the asymptotic behavior between time T and problem size n, for example $T = O\left(n^k\right)$ or $T = O\left(n^k \log n\right)$, compare asymptotic behavior bewteen quick sort and bubble sort.

Exercise 3 (find filename or directory): in the course, we introduce function ***system***, which can execute a command, we use ***system(“ls -al”)*** to list content of current directory. If we want to find out all files and subdirectories, then function ***system*** can help us.

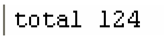
Step 1: use ***system(“ls –al > output.txt”)*** to store information of **ls** into file ***output.txt***. We take Figure 5 as an example in the later discussion.



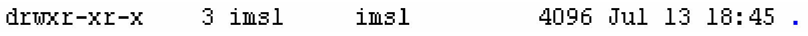
```
[imsl@linux system]$ ls -al
total 124
drwxr-xr-x    3 imsl     imsl         4096 Jul 13 18:45 .
drwxr-xr-x    5 imsl     imsl         4096 Jul 13 18:44 ..
-rwxrwxr-x    1 imsl     imsl        28325 Jul 13 18:27 a.out
drwxr-xr-x    2 imsl     imsl         4096 Jul 13 18:36 Debug
-rw-r--r--    1 imsl     imsl          523 Jul 13 18:27 getline.cpp
-rw-r--r--    1 imsl     imsl         1242 Jul 13 18:36 main.cpp
-rw-r--r--    1 imsl     imsl          786 Jul 13 18:27 output.txt
-rw-r--r--    1 imsl     imsl         4718 Jul 13 17:54 system.dsp
-rw-r--r--    1 imsl     imsl          535 Jul  8 10:55 system.dsw
-rw-rw-r--    1 imsl     imsl            0 Jul 13 18:44 system.ncb
-rw-r--r--    1 imsl     imsl        49664 Jul 13 18:36 system.opt
-rw-r--r--    1 imsl     imsl          672 Jul 13 18:36 system.plg
```

Figure 5: content of directory ***system***

Step 2: open file output.txt and read each line by function ***getline*** in page 69 of textbook (note that function ***getline*** in textbook read data from standard input, you need to rewrite it such that reading data from a file handler). Fro example, in Figure 5 we have

Line 1: 

```
total 124
```

Line 2: 

```
drwxr-xr-x    3 imsl     imsl         4096 Jul 13 18:45 .
```

Line 3:

```
drwxr-xr-x    5 imsl     imsl         4096 Jul 13 18:44 ..
```

Line 4:

```
-rwxrwxr-x    1 imsl     imsl        28325 Jul 13 18:27 a.out
```

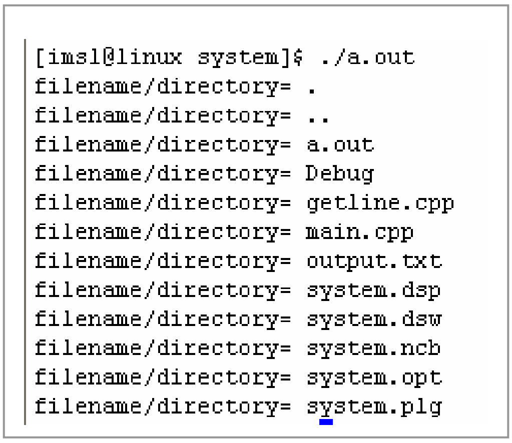
Step 3: for each line we can use space character (空白字元) as delimiter to find a token, for example. Line 1 has two tokens, "***total***" and "***124***". Line 2 has 9 tokens, token 1 is "***drwxr-xr-x***", token 2 is "***3***", token 3 is "***imsl***", token 4 is "***imsl***", token 5 is "***4096***", token 6 is "***Jul***", token 7 is "***13***", token 8 is "***18:45***" and token 9 is "**.**".

Remark 1: "**.**" is current directory and "**..**" means parent directory (上一層目錄).

Simple observation: for each line. 9-th token is file or sub-directory, hence we propose pseudo-code as following

```
system( "ls –al > output.txt" )
open file output.txt
for each line in file output.txt
    read each token of the line and report 9-th token.
endfor
close file output.txt
```

You need to implement tow functions, one is ***getline***, the other is to extract token from each line. In this example (Figure 5), the result is in Figure 6



```
[imsl@linux system]$ ./a.out
filename/directory= .
filename/directory= ..
filename/directory= a.out
filename/directory= Debug
filename/directory= getline.cpp
filename/directory= main.cpp
filename/directory= output.txt
filename/directory= system.dsp
filename/directory= system.dsw
filename/directory= system.ncb
filename/directory= system.opt
filename/directory= system.plg
```

Figure 6: report all files and directories of Figure 5

Exercise 4 (sorting on linked list): in the course, we use linked list to represent keyword of C-language, see Figure 7. Now we write { keyword, count } in file ***data.txt*** (see Figure 8) and read it to form a linked list, also perform a ***bubble sort*** on this liked list. You need

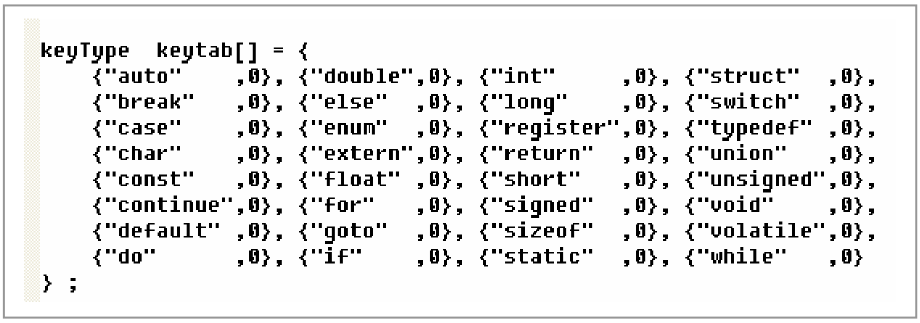
(1) use function ***fscanf*** to read keyword and count from *data.txt*

(2) rewrite bubble sort for this linked list, note that original version is only valid for continuous

array, not for discontinuous linked list.

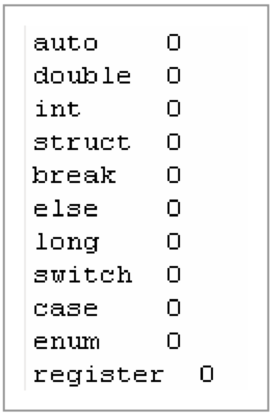
(3) Can you derive framework of sorting on linked list? Is quick sort possible?

(4) Can you do binary search in a sorted linked list?



```
keyType  keytab[] = {
    {"auto"     ,0}, {"double",0}, {"int"      ,0}, {"struct"  ,0},
    {"break"    ,0}, {"else"  ,0}, {"long"     ,0}, {"switch"  ,0},
    {"case"     ,0}, {"enum"  ,0}, {"register",0}, {"typedef" ,0},
    {"char"     ,0}, {"extern",0}, {"return"   ,0}, {"union"   ,0},
    {"const"    ,0}, {"float" ,0}, {"short"    ,0}, {"unsigned",0},
    {"continue",0}, {"for"   ,0}, {"signed"   ,0}, {"void"    ,0},
    {"default" ,0}, {"goto"  ,0}, {"sizeof"   ,0}, {"volatile",0},
    {"do"       ,0}, {"if"    ,0}, {"static"   ,0}, {"while"   ,0}
} ;
```

Figure 7: keyword of C

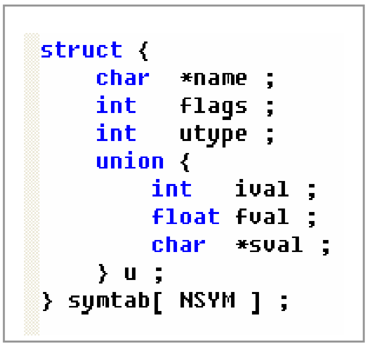


```
auto      0
double    0
int       0
struct    0
break     0
else      0
long      0
switch    0
case      0
enum      0
register    0
```

Figure 8: write {keyword, count} into file ***data.txt***, each pair occupies a line.

Exercise 5 (multi-dimensional array): in the course, we don’t discuss multi-dimensional array but focus on pointer array. However these two objects are similar, read section 5.7 in page 110 of textbook and write a driver to test codes in page 111. What’s the relationship between multi-dimensional array and pointer array? In other words, can you use pointer array to implement 2-dimensional double array?

Exercise 6 (union): read section 6.8 in page 147 of textbook, this section introduces another useful technique, ***union***, which we don’t discuss in the course. Write driver to implement a union-like data structure (see page 148).



```
struct {
    char  *name ;
    int   flags ;
    int   utype ;
    union {
        int   ival ;
        float fval ;
        char  *sval ;
    } u ;
} symtab[ NSYM ] ;
```

Figure 9: declare symbol table as a structure with union technique.

Try to show memory information of a union by using debugger.

Exercise 7 (lexical analyzer): given a document (text file), to find its lexical word is very important. Recall that compiler read a source file and recognize C-keyword, identifier, integer constant, floating constant and string constant. In page 97 of textbook, the author writes a piece of code to obtain an integer from standard input (you can also see Figure 10)

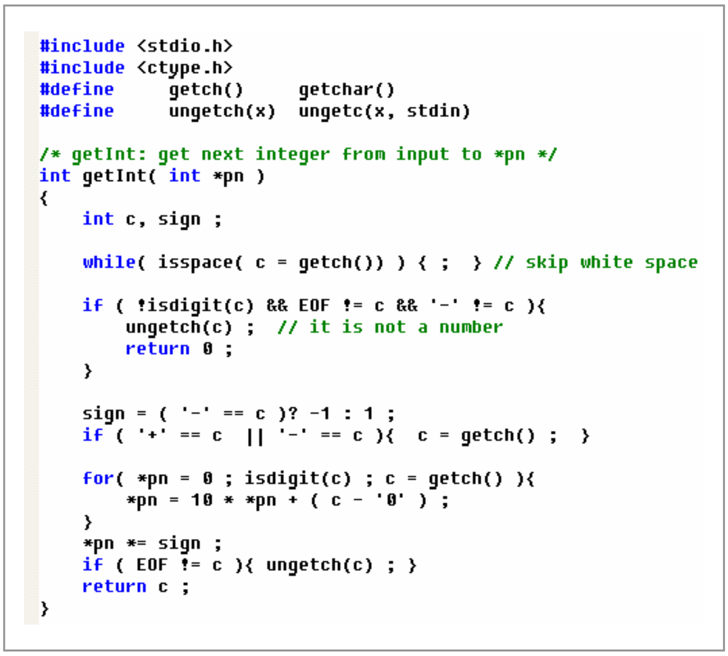
(1) write a driver to test function ***getInt*** in Figure 10, find all possible form of integer that it can recognize.

(2) Do exercise 5-1 in page 97 of textbook.

(3) Modify the code such that ***getInt*** reads an integer from a character string.

(4) Modify the code such that ***getInt*** reads an integer from a file.

(5) Read description **A2.3** of identifier in page 192 of textbook and write a function ***getId*** to recognize identifier from either character array or file .



```
#include <stdio.h>
#include <ctype.h>
#define     getch()     getchar()
#define     ungetch(x)  ungetc(x, stdin)

/* getInt: get next integer from input to *pn */
int getInt( int *pn )
{
    int c, sign ;

    while( isspace( c = getch()) ) { ;  } // skip white space

    if ( !isdigit(c) && EOF != c && '-' != c ){
        ungetch(c) ;  // it is not a number
        return 0 ;
    }

    sign = ( '-' == c )? -1 : 1 ;
    if ( '+' == c  || '-' == c ){  c = getch() ;  }

    for( *pn = 0 ; isdigit(c) ; c = getch() ){
        *pn = 10 * *pn + ( c - '0' ) ;
    }
    *pn *= sign ;
    if ( EOF != c ){ ungetch(c) ; }
    return c ;
}
```

Figure 10: get integer from standard input, see page 97 of textbook

(6) in C-language, comment is delimited by a pair of /* and */, in C++, comment starts from //, as you see in Figure 10, write a program to remove all comments of a given file.

Exercise 8 (static variables): read section 4.6 in page 83 of textbook, write a driver to test static variable for

Case 1: global static variable

Case 2: local static variable

What is life time and scope of static variable?