2008 summer course, C-language homework 7

Exercise 1 (familiar with getchar and putchar): implement code in Figure 1

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

/* convert input to lower case */
int main( int argc, char* argv[])
{
    int c ;

    while( EOF != (c=getchar()) ){
        putchar( tolower(c) ) ;
    }
    return 0 ;
}
```

Figure 1: convert character though input into lower case

(1) execute it in Linux machine and feed the program a file by using redirect operator
(2) modify the code such that one can read files from *argv* and convert the characters to lower case
(3) modify the code in (2), if character is alphabet, then convert upper case to lower case and lower case to upper case, otherwise keep the character.

Exercise 2 (variable-length argument list): if you search "va_list" in MSDN library, you can find example code like Figure 2.

```c
#include <stdio.h>
#include <stdarg.h>
int average( int first, ... );

int main( int argc, char* argv[] )
{
   /* Call with 3 integers (-1 is used as terminator). */
   printf( "Average is: %d\n", average( 2, 3, 4, -1 ) );
   /* Call with 4 integers. */
   printf( "Average is: %d\n", average( 5, 7, 9, 11, -1 ) );
   /* Call with just -1 terminator. */
   printf( "Average is: %d\n", average( -1 ) );
   return 0 ;
}

int average( int first, ... )
{
   int count = 0, sum = 0, i = first;
   va_list marker;

   va_start( marker, first );  // Initialize variable arguments.
   while( i != -1 ){
      sum += i;
      count++;
      i = va_arg( marker, int);
   }
   va_end( marker );     // Reset variable arguments.
   return( sum ? (sum / count) : 0 );
}
```

Figure 2: example of variable-length argument list in MSDN library.

(1) What is purpose of ***-1*** in Figure 2? Can you use other number?
(2) describe difference between function *average* and function *printf*.

(3) modify code in Figure 2 to deal with *double* and verify your answer.

Exercise 3 (minprintf): in the course, we write a minimum printf function to demonstrate usage of variable-length arguments, now we want to add one more option in function minprintf such that we can output *sturct point* (note that *struct point* is not a primitive type, printf does not show its content). First we need to define protocol (協定)

(1) format option for *sturct point* is ***%pt***

(2) pass pointer to *sturct point* into function minprintf

(3) show each field of *sturct point*

example: struct point maxpt = { 20, 30, "Earth" }

minprintf(“%pt\n”, &maxpt)

output :

point.x = 20

point.y = 30

point.name = Earth

```c
#include <stdio.h>
#include <stdarg.h>
// minprintf: miniml printf with variable argument list
void minprintf( char *fmt, ... )
{
    va_list ap ; // points to each unnamed arg in turn
    char *p, *sval ;
    int ival ;
    double dval ;
    va_start(ap, fmt); // make ap point to 1st unamed arg
    for ( p = fmt ; *p ; p++ ){
        if ( '%' != *p ){
            putchar(*p) ;
            continue ;
        }
        switch( *++p ){
        case 'd' :
            ival = va_arg(ap, int) ; printf("%d", ival) ;
            break ;
        case 'f' :
            dval = va_arg(ap, double) ; printf("%f", dval) ;
            break ;
        case 's' :
            for ( sval = va_arg(ap, char*) ; *sval ; sval++){
                putchar( *sval ) ;
            }
            break ;
        default:
            putchar( *p ) ;
            break ;
        }

    }// for each character *p
    va_end( ap ) ; // clean up when done
}
```

```c
struct point {
    int x ;  // x component of a point
    int y ;  // y component of a point
    char name[20] ; // name of the point

} ;
```

Figure 3: add one more option in function minprintf to deal with *struct point*

Exercise 4 (sprintf): sprintf is the same as printf except that output is written to a given string, not to standard output (display). Consider the following code

```c
#include <stdio.h>

int main( void )
{
   char  buffer[200], s[] = "computer", c = 'l';
   int   i = 35, j;
   float fp = 1.7320534f;

   // Format and print various data:
   j  = sprintf( buffer,     "   String:    %s\n", s ); // C4996
   j += sprintf( buffer + j, "   Character: %c\n", c ); // C4996
   j += sprintf( buffer + j, "   Integer:   %d\n", i ); // C4996
   j += sprintf( buffer + j, "   Real:      %f\n", fp );// C4996

   printf( "Output:\n%s\ncharacter count = %d\n", buffer, j );

   return 0;
}
```

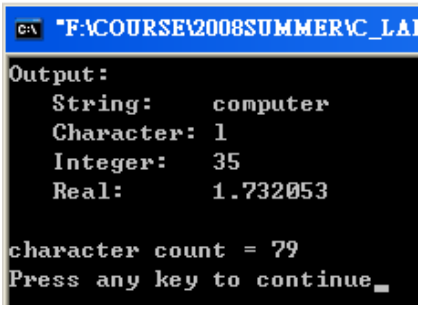

Figure 4: write a sequence of different type into a buffer through function *sprintf*, result is shown in right panel.

(1) what is the purpose of index j

(2) why is parameter $buffer + j$ valid in first argument of function *sprintf* ?

(3) explain the output result and "number of character count is 79"

(4) what happen if we declare size of *buffer* as 20?

Exercise 5 (scanf): in the course, we introduce format specification of function *scanf*,

```
%[*] [width] [{h | l | ll | I64 | L}]type
```

We restrict size of converted data as 2 in Figure 5, execute this program and input integer with different size, what's the result? Can you explain it?

```c
#include <stdio.h>

int main( int argc, char* argv[] )
{
    double  sum, v ;

    sum = 0.0 ;
    while( 1 == scanf("%2lf", &v) ){
        printf("\t read v = %.2f, sum = ",v ) ;
        printf("\t%.2f\n", sum += v ) ;
    }
    return 0 ;
}
```

Figure 5: restrict size of converted data as 2 in function *scanf*.

second, use *fscanf* to rewrite code in Figure 5 and check the result.

Exercise 6 (error of printf): in the course, we say variable-length argument lists has potential bug when format string *fmt* does not match number of parameters. Fro example in Figure 6, we lack a parameter corresponding to %s in function printf

(1) write the program, run it on visual C and Linux machine

(2) what is warning message of icpc and g++?

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("x = %d, word = %s\n", 3 );

    return 0 ;
}
```

Figure 6: we lack a parameter in printf

Exercise 7 (file access): implement code example in the course to familiar with File access routines. In Figure 7, we concatenate all input files and show the result into the screen, modify the code such that we can store the result into another file, you can specify output filename yourself.

```c
#include <stdio.h>

void filecopy( FILE *ifp, FILE *ofp ) ;

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

    if ( 1 == argc ){ // no args: copy standard input
        filecopy( stdin, stdout) ;
    }else{
        while( --argc >0 ){
            fp = fopen( *++argv, "r" ) ;
            if ( NULL == fp ){
                printf("cat: can't open %s\n", *argv);
            }else{
                filecopy( fp, stdout ) ;
                fclose( fp ) ;
            }
        }// for each argument
    }
    return 0 ;
}
```

```c
#include <stdio.h>

void filecopy( FILE *ifp, FILE *ofp )
{
    int c ;

    while ( EOF != (c = fgetc(ifp)) ){
        fputc(c, ofp ) ;
    }
}
```

Figure 7: left panel is *main.cpp* and right panel is *filecopy.cpp*

Exercise 8 (command execution): we use function *system* in stdlib.h to execute command in host machine, in Figure 8, we show content of a directory by issuing command “dir” in windows or “ls” in Linux, here we use compilation directive to help us find the correct version.

```c
#include <stdlib.h>

int main( void )
{

#ifdef _WIN32
    system( "dir" );
#else
    system( "ls -al" ) ;
#endif

   return 0 ;
}
```

Figure 8: use function *system* to invoke command in host machine, since each host machine may has different command name to do the same work, for example, in order to show content of directory, windows uses “dir” and linux uses ”ls”. Hence we use compilation directive to help us find the correct version.

(1) Can you call this command again by function *system*, we called this process as recursive call itself. Write the code and test it in Linux machine, what’s the result

(2) Modify code in (1) such that you can call yourself 5 times only.

Exercise 9 (preprocessor): write a preprocessor (just focus on macro substitution), you need

(1) open a file (source code)
(2) read a character one-by-one till matching some macro
(3) replace the macro