10 Interesting Programs

"It is better to be humble."

Everybody might have the question: why programmers are prone to C? The answer is very simple: C's structure allows programmers to write a small-tight code for complex programs. In this chapter let's see a few interesting programs that use C's real power.

10.1 Power of 2

How to find whether the given number is a power of 2? i.e., 1, 2, 4, 8, 16, 32.. are powers of 2.

```
#define ISPOWOF2( n )        ( ! ( n & ( n-1 ) )
```

Amazing, isn't it?

10.2 Prime Numbers

Everyone knows that prime number is a number that is not divisible by any other number except by 1 and itself. Hence the prime number series will be: 2, 3, 5, 7, 11, 13, 17, 19…

Generation of prime number seems to be easy. But the efficient implementation is not common. The following program does the efficient implementations and it will help you to increase your programming skill.

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

typedef int BOOLEAN;

BOOLEAN IsPrime( int n )  /* checks for prime */
{
   int i;
   BOOLEAN flag = ( n>1 );
   for( i=2 ; flag && i<=sqrt(n) ; ++i )
            flag = ( n%i );
   return( flag );
} /*--IsPrime( )--------*/

int main( void )
{
   int i;
```

```
   for( i=1 ; i<1000 ; ++i )
      if ( IsPrime(i) )
            printf( "%d " , i );
   return(0);
} /*--main( )-----*/
```

See, the `BOOLEAN` variable `flag` in `IsPrime( )`. It is used to break the *for* loop. As we haven't used any break or jump statement, it is considered as a good programming.

10.3 Roman Letters

The following program will help you to improve your programming skill. The following program converts the Arabic numbers to Roman numbers.

```
void  InRoman( int n )        /* converts arabic to roman */
{
   int i, v[ ] = { 1, 4, 5, 9, 10, 40, 50, 90, 100,
               400, 500, 900, 1000, 9999 };
   char *r[ ] = { "I", "IV", "V", "IX", "X", "XL", "L", "XC", "C",
              "CD", "D", "CM", "M" };
   while ( n )
     {
      for( i=0 ; v[i]<=n ;++i )
            ;
      --i;
      n -= v[i];
      printf( "%s", r[i] );
     }
} /*--InRoman( )----------*/

int main( void )
{
    int n;
    printf( "Enter the Arabic number: " );
    scanf( "%d", &n );
    printf( "In Roman, " );
    InRoman( n );
    return(0);
} /*--main( )---------*/
```

Note

The above program works fine upto 4999, because for 5000 we have $\overline{V}$. In ANSI C, we can't get $\overline{V}$. It can be done with Turbo C(DOS programming) by changing character set with int 10h.

10.4 Day of Week

For a given date (i.e., year, month & day), we may need to know the day of the week (i.e., Sunday or Monday…). We have so many ways to find that. But the code by **Tomohiko Sakamoto** is very interesting as well as mysterious! Here is the code…It works for the years greater than 1752 (Gregorian Calendar).

```
int DayOfWeek( int y, int m,  int d )  /* returns Day of Week:
                                          0 = Sunday, 1= Monday...
                                       */
{
    static int t[] = { 0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4 };
    y -= m < 3;
    return (y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;
} /*--DayOfWeek( )--------*/
```

10.5 Calendar

The following program prints the calendar for a given year like Unix's *cal* utility. However, it won't work exactly like "cal" for year-wise output. For that you need to store the output in an array as a grid.

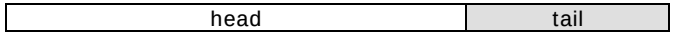
```
#include <stdio.h>
#include <stdlib.h>

int Days_Tbl[2][12] = {
                  { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
                  { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
                  };
char *Month_Tbl[12] =  {
                    "January", "February", "March", "April", "May",
                    "June", "July", "August", "September",
                    "October", "November", "December"
                  };

int FirstDayOfMonth( int m, int y );
void PrintCalendar( int m, int y );

int FirstDayOfMonth( int m, int y )
{
   int i, leap;
   long d;

   if ( y>1752 )  /* for Gregorian Calendar */
      {
      leap = ( y%4==0&&y%100!=0 || y%400==0 );
```

```
      d = 365L*1752 + 1752/4;
      d += 365L*(y-1752-1) + (y-1752-1)/4 - (y-1752-1)/100
                      + (y-1752-1)/400 + 6;
      }
    else   /* for Julian Calendar */
     {
      leap = ( y%4==0 );
      d = 365L*(y-1) + (y-1)/4 + 6;
     }
   for( i=1; i<m; ++i )
            d += Days_Tbl[leap][i-1];
   if ( y>1752 || (y==1752 && m>9) )
            d -= 11;
   return( d % 7 );
} /*--FirstDayOfMonth( )--------*/

void PrintCalendar( int m, int y )
{
   int i, leap, firstdayofmonth;

   firstdayofmonth = FirstDayOfMonth( m, y );
   leap = ( y>1752 ) ? ( y%4==0&&y%100!=0 || y%400==0 ) : ( y%4==0 );

   printf( "%13s - %d\n", Month_Tbl[m-1], y );
   printf( "Sun Mon Tue Wed Thu Fri Sat\n" );
   for ( i=0; i<firstdayofmonth ; ++i )
            printf( "    " );
   for ( i=1 ; i<=Days_Tbl[leap][m-1] ; ++i )
     {
      printf( "%3d ", i );
      if ( (firstdayofmonth + i)%7 == 0 )
                printf("\n");
      if (y==1752 && m==9 && i==2)
          {
            i += 11;
            firstdayofmonth += 3;
          }
     }
     printf( "\n" );
} /*--PrintCalendar( )-----*/

int main( int argc, char *argv[ ] )
{
   int m, y;
   switch( argc )
     {
      case 1:
```

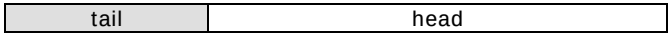
```
            printf( "Syntax: cal [month] year \n" );
            break;
      case 2:
            y = atoi( argv[1] );
            for ( m=1 ; m<=12 ; ++m )
               {
                  PrintCalendar( m, y );
                  printf( "Press <ENTER>....\n" );
                  getchar( );
               }
            break;
      case 3:
            m = atoi( argv[1] );
            y = atoi( argv[2] );
            PrintCalendar( m, y );
     }
   return(0);
} /*--main( )----*/
```

10.6 Memory-Swap

Normal Memory



Swapped Memory



Consider the situation in which you want to swap the contents of memory without using much external storage space and one portion is larger than the other. In our example, the *head* portion is larger than *tail*. It is really a tough job. The code by **Ray Gardner** efficiently solves this problem.

```
/* memrev: reverse "count" bytes starting at "buf" */
void memrev( char  *buf, size_t count )
{
      char *r;

      for ( r = buf + count - 1; buf < r; buf++, r-- )
      {
          *buf ^= *r;
          *r   ^= *buf;
          *buf ^= *r;
      }
}
```

```
/* aswap: swap "head" bytes with "tail" bytes at "buf" */
void aswap( char  *buf, size_t head, size_t tail )
{
      memrev( buf, head );
      memrev( buf + head, tail );
      memrev( buf, head + tail );
}
```

10.7 Block Structure

When we want to declare a variable in the middle of the program, we use block structure as:

```
int main( void )
{
      int a;
      a = 5;
        :
        :
      {
      int b;   /* declaration requires block structure. Value of
                  'b' is available only to this block   */
      b = 6;
        :
       }
        :
}
```

10.7.1 Swap macro using Block Structure

When we need a swap macro that works for any data types, we must use block structure.

```
#define SWAP(datatype, a, b)  {                        \
                                    datatype a##b = a;\
                                    a = b;             \
                                    b = a##b;          \
                              }
```

In order to swap the values of two variables we need a temporary variable and it needs a name. In fact the name may be *temp*. But if someone passes a variable that has a name *temp*, like `SWAP( int, a, temp)`, everything will collapse! So, we use the preprocessor argument concatenation operator ## to create the name (here we get `ab`) from the actual variable names in the call. This guarantees that the result won't be either of the actual arguments.

Using XOR(^) operator also we can write the above `SWAP` macro. Here is the code…

```
#define SWAP(datatype, a, b)                    \
    (unsigned char *)x=(unsigned char *)(&(a)); \
```

```
        (unsigned char *)y=(unsigned char *)(&(b)); \
        size_t size = sizeof(datatype);             \
        while (size--) {                            \
            *x ^= *y;                               \
            *y ^= *x;                               \
            *x ^= *y;                               \
            x++;                                    \
            y++;                                    \
        }
```

10.8 Printf with %b

Using the conversion characters %X and %0 we can directly print any decimal number as hexadecimal and octal. But to print binary value, we don't have any conversion characters. The following program introduces '%b' as a conversion character for binary.

```
#include <stdarg.h>

void MyPrintf( char *fmt, ... )
{
   va_list aptr;   /* Points to each unscanned arg in turn */
   char *p, *sval, str[17];
   int ival;
   double dval;
   va_start( aptr, fmt ); /* Initialize the argument pointer. */

   /* Retrieve each argument in the variable list... */
   for( p=fmt; *p ; ++p )
      if( *p=='%' )
       switch( * ++p )
       {
          case 'd':
                  ival = va_arg( aptr, int );
                  printf( "%d", ival );
                  break;
          case 'f':
                  dval = va_arg( aptr, double );
                  printf( "%f", dval );
                  break;
          case 's':
                  for( sval=va_arg(aptr, char*); *sval; ++sval )
                        putchar( *sval );
                  break;
          case 'b':     /* for binary */
                  ival = va_arg(aptr, int);  /* Get it as integer */
                  /* radix should be 2 for binary in itoa... */
                  itoa( ival, str, 2 );
```

```
                    for( sval=str; *sval; ++sval )
                          putchar(*sval);
                       break;
               default:
                       putchar(*p);
            }
         else
          putchar( *p );
      va_end( aptr );          /* Clean up when done */
} /*--MyPrintf( )----------*/

int main( void )
{
    MyPrintf( "7 in binary is %b \n", 7 );
    return(0);
} /*--main( )-----*/
```

This is not a complete implementation of `printf( )`. In fact `MyPrintf( )` don't work for %1d, %u, and other format strings. The complete implementation is left to the reader as an exercise.

Exercises

1. Write a program that use only bitwise operators to multiply any number by 2.
2. Find out the difference between Unix's text file and DOS's text file. Write a program that converts Unix based text file into DOS based text file, and vice-versa.
3. Implement your own data type for very very long integer (i.e., it should accept any number of digits say, 899999998998998998998998989989). Use that data type to find out factorial for any number.

Suggested Projects

1. Write source code colorizer software. Source code colorizer formats the given C file into HTML file with necessary syntax highlighting. (Hint: You may need to know the syntaxes of HTML)
2. Write a utility that indents the given C file. That is it should align the C code properly for better clarity.
3. Solve all the questions in K&R. It's really a tough project as no one achieved it successfully!