```
     /*
0    *    Simple bit manipulation.
     *
     *         Sample C program to illustrate:
     *              - various C control structures and operations
     *              - specification style
     *              - introduce gray codes
     */

1    #include <stdio.h>
2    #define HIGH_BIT 31

     /*
     *    General Assumptions:
     *         The HIGH_BIT+1 bits in an unsigned int are numbered in
     *         big-endian order.  That is, the most significant bits occur
     *         first.  HIGH_BIT, HIGH_BIT-1, ..., 2, 1, 0
     */

3    unsigned int bit_get (unsigned int x, int i) {
     /*
4    *    Pre:
     *         0 <= i <= HIGH_BIT
     *    Post:
     *         bit_get (x,i) is the value of bit i of x
     *    Side effects:
     *         none
     */

5         return ( x >> i) & 1;
     }

     unsigned int bit_put (unsigned int x, int i, unsigned int v) {
     /*
     *    Pre:
     *         0 <= i <= HIGH_BIT
     *    Post:
     *         bit_put (x, i, v) is the same as x except that bit i is
     *         set to the same value as bit 0 of v.
     *    Side effects:
     *         none
     */
6         if ( v & 1 ) {
               return x | (1 << i);
          } else {
               return x & ~(1 << i);
          }
     }
```



```
7    void bit_printf (unsigned int x) {
     /*
     *    Pre:
     *         none
     *    Post:
     *         none
     *    Side Effects:
     *         the bits of x are printed on stdout in big-endian order.
     */
          int i;
          for (i = HIGH_BIT; i >= 0; i-- ) {
8              printf("%d", bit_get(x,i));
          }
     }


9    void main (int argc, char* argv[]) {
     /*
     *    Sample driver program to exercise bit routines.  Some of these
     *    comments are simplistic - intended for first time readers.
     */
10        int i;
          unsigned int x, e, d;

          /* read in an integer */
          printf ("Give me an integer:");
11        scanf ("%d", &x);

          /* print the bits of x */
          printf ("The bits of %d are:", x);
          bit_printf (x);
12        printf ("\n");

          /* complement the bits of x */
          printf ("The comp of %d are:", x);
          i = HIGH_BIT;
          while ( i >= 0 ) {
13             printf ("%d", bit_get(bit_put(x,i,bit_get(x,i)+1), i));
               i--;
          }
          printf ("\n");
```

```c
/*
 *    A gray code is a bit encoding of the natural numbers such
 *    that adjacent numbers differ only in one bit position,
 *    including the wrap around back to 0.
 *
 *              n         binary                   gray
 *              0            000                    000
 *              1            001                    001
 *              2            010                    011
 *              3            011                    010
 *              4            100                    110
 *              5            101                    111
 *              6            110                    101
 *              7            111                    100
 */

/* encode x: compute e, the encoding of x in gray code */
14   {
     int i;
     e = 0;
     for ( i=0; i < HIGH_BIT; i++) {
15        /* Invariant:
          *    bits 0 to i-1 of e are correct gray code bits
          */
16        e = bit_put (e, i, bit_get(x,i) ^ bit_get(x, i+1) );
     }
     e = bit_put(e, HIGH_BIT, bit_get(x, HIGH_BIT) );
}
printf ("Gray encode of  %d:", x);
bit_printf (e);
printf ("\n");
```



```c
/* decode x: compute d, the number whose gray code is x */
     {
          int i, j;
          unsigned int s;

          d = 0;
          for ( i = 0; i <= HIGH_BIT; i++ ) {
               /* Invariant:
               *    bits 0 to i-1 of d are correctly decoded bits
               */
               s = 0;
               for ( j = i; j <= HIGH_BIT; j++ ) {
17                  s ^= bit_get(x, j);
               }
               /* Invariant:
               *    s is the exclusive or of bits i ... HIGH_BIT of x
               */
18             d = bit_put (d, i, s);
          }
     }
     printf ("Gray decode of  %d:", x);
     bit_printf (d);
     printf ("\n");
}
```