```
/* lab2.c
 * Rodrigo Ignacio Rojas Garcia
 * rrojas
 * ECE 222, Fall 2016
 * MP2
 *
 * Purpose: The purpose of the program is to allow the user to encode three letters
 into a hexadecimal number and decode a hexadecimal number into three letters. The
program is divided into two
           sections: Encode and Decode. If the user types the word "enc" along wit
h three letters, the program will jump into function "encode" and will first store
the three letters into an unsigned
           int variable with 24 bits. Then, it stores those 24 bits into an unisgn
ed int variable with 29-bits without bits 0, 1, 3, 7, and 15. After that is accompl
ished, the program will count the
           parity bits of bit 0,1,3,7, and 15 and wil store them in their respecti
ve bits in the 29-bit unsigned int variable. Finally, the program will display the
parity bits along with the encoded
           word in hexadecimal values. If the user types "dec" along with a hexade
cimal number, the program will store that codeword and will store it into an unsign
ed int variable with 29-bits.
           Then, it will check for the parity bits of bits 0,1,3,7, and 15. If the
re is no error, the program will display parity bits 0,1,3,7, and 15 and will displ
ay "No error" along with the
           decoded word.. If the error is between bits 28-0, it will fix the parit
y bit in the right location. Then, it will store the 29-bit unsigned int variable i
nto a 24-bit unsigned int
           variable. After this, the program will store the 24-bits into three dif
ferent letters of data type unsigned char. Finally, it will display the error parit
y bits along with the parity bit
           fixed and the word that was decoded. If the error is greater than 29, t
he program will display "Detection error" and will display the wrong parity bits. T
he user can enter "quit" to quit
           the program as well.

 *
 * Assumptions: User knows to enter either "dec" to decode the word, "enc" to encod
e the word, or "quit" to quit the program. Also, the user knows that he/she must en
ter three letters when encoding
   and has a maximum of 8 letters and numbers to enter when decoding. User knows th
at the program can handle only one error of parity bits.

 *
 *
 * Bugs:
 *
 *
 * To create a nicely formated PDF file for printing install the enscript
 * command.  To create a PDF for "file.c" in landscape with 2 columns do:
 *     enscript file.c -G2rE -o - | ps2pdf - file.pdf
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAXLINE 100

// function prototypes
void encode(unsigned char first_letter, unsigned char second_letter,
        unsigned char third_letter);
void decode(unsigned int codeword);

int main()
{
    char line[MAXLINE];
    char command[MAXLINE];
    char inputcs[MAXLINE];
    int  items;
    int i, invalid;
    unsigned int codeword;

    printf("\nMP2: encoding and decoding (29, 24) Hamming code.\n");
    printf("Commands:\n\tenc 3-letters\n\tdec 8-hex-digits\n\tquit\n");

    // each call to fgets, collects one line of input and stores in line
    while (fgets(line, MAXLINE, stdin) != NULL) {
        items = sscanf(line, "%s%s", command, inputcs);
        if (items == 1 && strcmp(command, "quit") == 0) {
            break;
        } else if (items == 2 && strcmp(command, "enc") == 0) {
            // encoding
            if (strlen(inputcs) != 3 || !isprint(inputcs[0]) ||
                    !isprint(inputcs[1]) || !isprint(inputcs[2])) {
                printf("Invalid input to encoder: %s\n", inputcs);
            } else {
                encode(inputcs[0], inputcs[1], inputcs[2]);
            }
        } else if (items == 2 && strcmp(command, "dec") == 0) {
            // decoding: convert hex digits to integer
            items = sscanf(inputcs, "%x", &codeword);
            if (items != 1 || strlen(inputcs) > 8) {
                printf("Invalid input to decoder: %s\n", inputcs);
            } else {
                // verify all digits are hex characters because
                // scanf does not reject invalid letters
                for (i=0, invalid=0; i < strlen(inputcs) && !invalid; i++) {
                    if (!isxdigit(inputcs[i]))
                        invalid = 1;
                }
                // if 8 digits, leading digit must be 1 or 0
                if (invalid) {
                    printf("Invalid decoder digits: %s\n", inputcs);
                } else if (strlen(inputcs) == 8 && inputcs[0] != '1'
                        && inputcs[0] != '0') {
                    printf("Invalid decoder leading digit: %s\n", inputcs);
                } else {
                    decode(codeword);
                }
            }
        } else {
            printf("# :%s", line);
        }
    }
    printf("Goodbye\n");
    return 0;
}

// Encode Function.
void encode(unsigned char first_letter, unsigned char second_letter,
        unsigned char third_letter)
{
    printf("%9s%9c%9c%9c\n", "Encoding:", third_letter, second_letter, first_letter
);
    printf(" 0x    00%9x%9x%9x\n", third_letter, second_letter, first_letter);

   // Variable Declaration Section.
   int counter2 = 0;
   int counter3= 0;
   int counter4 = 15;
   int counter5 = 0;
   int counter6 = 23;
   int counter7 = 0;
```

```
    int counter8 = 0;
    int counter9 = 28;
    int counter10 = 0;
    int counter11 = 14;
    int counter12 = 0;
    int counter13 = 6;
    int counter14 = 0;
    int counter15 = 0;
    int counter16 = 0;
    int counter17 = 0;
    int counter18 = 0;
    int counter19 = 0;
    int counter20 = 0;
    int counter21 = 0;
    int counter22 = 0;
    int counter23 = 0;
    int counter24 = 0;
    int count1 = 0;
    int count2 = 0;
    int count3 =0;
    int count4 = 0;
    int count5 = 0;
    int count6 = 0;
    int count7 = 0;
    int count8 = 0;
    int count9 = 0;
    int count10 = 0;
    int count11 = 0;
    int count12 = 0;
    int count13 = 0;
    int p1 = 0;
    int p2 = 0;
    int p4 = 0;
    int p8 = 0;
    int p16 = 0;
    unsigned int word = 0;
    unsigned int codeword = 0;

    // Stores all 24 bits from three different letters into one variable called word
.
    for (counter2 = 7; counter2 >= 0; counter2--)
    {
         // Sets bits 8-0 of variable word to one if bits 8-0 of first_letter is 1.
         if ((first_letter & ( 1 << counter2)) >> counter2 == 1)
         {
         word = word | (1 << counter2);
         }
    }

    for (counter3 = 7; counter3 >= 0; counter3--)
    {
         // Sets bits 16-9 of variable word to one if bits 8-0 of second_letter is 1
.
         if ((second_letter & (1 << counter3)) >> counter3 == 1)
         {
                 word = word | (1 << counter4);
         }
         counter4--;
    }

    for (counter5 = 7; counter5 >= 0; counter5--)
    {
         // Sets bits 24-17 of variable word to one if bits 8-0 of third_letter is 1
.
         if ((third_letter & (1 << counter5)) >> counter5 == 1)
         {
                 word = word | (1 << counter6);
         }
         counter6--;
    }

    // Prints the information word in binary form with spaces
    printf(" -------- ");
    for (counter7 = 23; counter7 >= 0 ; counter7--)
    {
         if (counter7 <= 23  && counter7 >= 16)
         {
                 printf("%d", (word & (1 << counter7)) >> counter7);
                 if (counter7 == 16)
                 {
                         printf(" ");
                 }
         }
         if (counter7 <= 15 && counter7 >= 8)
         {
                 printf("%d", (word & (1 << counter7)) >> counter7);
                 if (counter7 == 8)
                 {
                         printf(" ");
                 }
         }
         if (counter7 <= 7 && counter7 >=0)
         {
                 printf("%d", (word & (1 << counter7)) >> counter7);
         }

    }

    printf("\n");

    // Sets bits 24-0 from variable word to a 29-bit variable codeword.
    for (counter8 = 23; counter8 >=11; counter8--)
    {
         // Sets bits 24-12 of variable word into bits 29-17 of variable codeword.
         if ((word&(1 << counter8)) >> counter8 == 1)
         {
                 codeword = codeword | (1<< counter9);
         }
         counter9--;
    }

    for (counter10 = 10; counter10 >= 4; counter10--)
    {
         // Sets bis 11-5 of variable word into bits 15-9 of variable codeword.
         if ((word&(1 << counter10)) >> counter10 == 1)
         {
                 codeword = codeword | (1 << counter11);
         }
         counter11--;
    }

    for (counter12 = 3; counter12 >= 1; counter12--)
    {
         // Sets bits 4-2 of variable word into bits 8-5 of variable codeword.
         if ((word&(1 << counter12)) >> counter12 == 1)
         {
                 codeword = codeword | (1 << counter13);
         }
         counter13--;
    }

    // Sets bit 1of variable word into bit 3 of variable codeword.
    if ((word&(1 << 0)) >>  0 == 1)
    {
```

```
        codeword = codeword | (1 << 2);
}

// Calculate number of parity bits on P1.
for (counter14 = 0; counter14 < 29; counter14 += 2)
{
        if ((codeword&(1 << counter14)) >> counter14 == 1)
        {
                count1++;
        }
}

// If count1 is an even number, count1 is equal to 0.
if (count1 % 2 == 0)
{
        count1 = 0;
}

// If P1 is even, is set to 0.
if (count1 == 0)
{
        p1 = count1;
}

// If P1 is odd, is set to 1.
else
{
        p1 = 1;
}

// Counts parity bit p2 from bit 2-26 by adding by 4.
for (counter15 = 1; counter15 < 29; counter15 += 4)
{
        if ((codeword&(1<< counter15)) >> counter15 == 1)
        {
                count2++;
        }
}

// Counts parity bit p2 from bit 3-27 by adding by 4.
for (counter16 = 2; counter16 < 29; counter16 += 4)
{
        if ((codeword&(1<< counter16)) >> counter16 ==1)
        {
                count3++;
        }
}

// Adds counts of count2 and count3 and stores it on count4.
count4 = count2 + count3;

// If count4 is even, p2 is 0.
if (count4 % 2 == 0)
{
        p2 =0;
}
// If count4 is odd, p2 is 1.
else
{
        p2 = 1;
}

// Counts parity bits of p4 from bit 4-7 by adding 1.
for (counter17 = 3; counter17 < 7; counter17++)
{
        if ((codeword&(1<< counter17)) >> counter17 == 1)
        {
                count5++;
        }
}

// Counts parity bits p4 from bit 12-15 by adding 1.
for (counter18 = 11; counter18 < 15; counter18++)
{
        if ((codeword&(1<<counter18)) >> counter18 == 1)
        {
                count6++;
        }
}

// Counts parity bits p4 from bit 20-23 by adding 1.
for (counter19 = 19; counter19 < 23; counter19++)
{
        if ((codeword&(1<< counter19)) >> counter19 == 1)
        {
                count7++;
        }
}

// Counts parity bits p4 from bit 28-29 by adding 1.
for (counter20 = 27; counter20 <= 28; counter20++)
{
        if ((codeword&(1<<counter20)) >> counter20 == 1)
        {
                count8++;
        }
}

// count9 is the addition of p4 parity bits.
count9 = count5 + count6 + count7 + count8;

// If count 9 is odd, p4 is 0.
if (count9 % 2 == 0)
{
        p4 = 0;
}

// If count9 is even, p4 is 1.
else
{
        p4 = 1;
}

// Counts parity bits of p8 from 6-14 by adding 1.
for (counter21 = 7; counter21 < 15; counter21++)
{
        if ((codeword&(1<<counter21)) >> counter21 == 1)
        {
                count10++;
        }
}

// Counts parity bits of p8 from 23-28 by adding 1.
for (counter22 = 23; counter22 <= 28; counter22++)
{
        if((codeword&(1<<counter22)) >> counter22 == 1)
        {
                count11++;
        }
}

// Adds count10 and count11 to variable count12.
count12 = count10 + count11;
```

```
// If count12 is even, p8 is 0.
if (count12 % 2 == 0)
{
      p8 = 0;
}
// If count12 is odd, p8 is 0.
else
{
      p8 = 1;
}

// Counts parity bits of p16 from 16-29 by adding 1.
for (counter23 = 15; counter23 <= 28; counter23++)
 {
      if ((codeword&(1<<counter23)) >> counter23 == 1)
      {
              count13++;
      }
 }

 // If count13 is even, p16 is 0.
 if (count13 % 2 == 0)
 {
      p16 = 0;
 }
 // If count13 is odd, p16 is 1.
 else
 {
      p16 = 1;
 }

 // Sets bit 1 of variable codeword equal to p1.
 codeword = codeword | (p1 << 0);
 // Sets bit 2 of variable codeword equal to p1.
 codeword = codeword | (p2 << 1);
 // Sets bit 4 of variable codeword equal to p1.
 codeword = codeword | (p4 << 3);
 // Sets bit 8 of variable codeword equal to p1.
 codeword = codeword | (p8 << 7);
 // Sets bit 16 of variable codeword equal to p1.
 codeword = codeword | (p16 << 15);

  //Prints parity bits.
  printf("P1 : %d\n", p1);
  printf("P2 : %d\n", p2);
  printf("P4 : %d\n", p4);
  printf("P8 : %d\n", p8);
  printf("P16: %d\n", p16);

  // Prints the codeword bits in binary form with spaces
  printf(" ---");
 for (counter24 = 28; counter24 >= 0 ; counter24--)
 {
      if (counter24 <= 28  && counter24 >= 24)
      {
              printf("%d", (codeword & (1 << counter24)) >> counter24);
              if (counter24 == 24)
              {
                      printf(" ");
              }
      }
      if (counter24 <= 23 && counter24 >= 16)
      {
              printf("%d", (codeword & (1 << counter24)) >> counter24);
              if (counter24 == 16)
              {
                      printf(" ");
              }
      }
      if (counter24 <= 15 && counter24 >= 8)
      {
              printf("%d", (codeword & (1 << counter24)) >> counter24);
              if (counter24 == 8)
              {
                      printf(" ");
              }
      }
      if (counter24 <= 7 && counter24 >= 0)
      {
              printf("%d", (codeword & (1 << counter24)) >> counter24);
      }

   }

    printf("\n");

    // Prints the codeword in hex format
    printf(" Codeword: 0x%.8X\n", codeword);
    printf("\n");
}

//Decode Function.
void decode(unsigned int codeword)
{
    // Variable Declaration Section.
    unsigned int decodeword = 0;
    unsigned char first_letter = 0;
    unsigned char second_letter = 0;
    unsigned char third_letter = 0;
    unsigned int error = 0;
    int counter1 =0;
    int counter2 = 0;
    int counter3 = 0;
    int counter4= 0;
    int counter5 = 0;
    int counter6 = 0;
    int counter7 = 0;
    int counter8 = 0;
    int counter9 = 0;
    int counter10 = 0;
    int counter13 = 0;
    int counter14 = 23;
    int counter15 = 0;
    int counter16 = 10;
    int counter17 = 0;
    int counter18 = 3;
    int counter20 = 0;
    int counter21 = 0;
    int counter22 = 7;
    int counter23 = 0;
    int counter24 = 7;
    int counter25 = 0;
    int count1 = 0;
    int count2 = 0;
    int count3 = 0;
    int count4 = 0;
    int count5 = 0;
    int count6 = 0;
    int count7 = 0;
    int count8 = 0;
    int count9 = 0;
    int count10 = 0;
    int count11 = 0;
    int count12 = 0;
```

```
int count13 = 0;
int e1 = 0;
int e2 = 0;
int e4 = 0;
int e8 = 0;
int e16 = 0;

printf("Decoding: 0x%.8X\n", codeword);

// Counts parity bits of Bit 1 of variable codeword.
for (counter1 = 0; counter1 <= 28; counter1 += 2)
{
    if ((codeword & (1 << counter1)) >> counter1 == 1)
    {
            count1++;
    }
}

// If count1 is even, sets e1 to 0.
if (count1 % 2 == 0)
{
    e1 = 0;
}
// If count1 is odd, sets e1 to 0.
else
{
    e1 = 1;
}

// Counts parity bit Bit 2 from variable codeword.
for (counter2 = 1; counter2 <= 26; counter2 += 4)
{
    if ((codeword & (1 << counter2)) >> counter2 == 1)
    {
            count2++;
    }
}
// Counts parity bit Bit 2 from variable codeword.
 for (counter3 = 2; counter3 <= 27; counter3 += 4)
{
    if ((codeword & (1 << counter3)) >> counter3 == 1)
    {
            count3++;
    }
}

// Adds count of parity bits of Bit 2 and sets it to count4.
count4 = count2 + count3;

// If count4 is even, sets e2 to 0.
if (count4 % 2 == 0)
{
    e2 = 0;
}
// If count4 is odd, sets e2 to 0.
else
{
    e2 = 1;
}

// Counts parity bits of Bit 4 from codeword.
for (counter4 = 3; counter4 < 7; counter4++)
{
    if ((codeword&(1 << counter4)) >> counter4 == 1)
    {
            count5++;
    }
}
for (counter5 = 11; counter5 < 15; counter5++)
{
    if ((codeword&(1 << counter5)) >> counter5 == 1)
    {
            count6++;
    }
}
for (counter6 = 19; counter6 < 23; counter6++)
{
    if ((codeword&(1 << counter6)) >> counter6 == 1)
    {
            count7++;
    }
}
for (counter7 = 27; counter7 <= 28; counter7++)
{
    if ((codeword&(1 << counter7)) >> counter7 == 1)
    {
            count8++;
    }
}

// Adds counts of parity bit Bit 4 and stores it in count9.
count9 = count5 + count6 + count7 + count8;

// If count9 is even, e4 is 0.
if (count9 % 2 == 0)
{
    e4 = 0;
}
// If count9 is odd, e4 is 1.
else
{
    e4 = 1;
}

// Counts parity bits of Bit 8 from codeword.
for (counter8 = 7; counter8 < 15; counter8++)
{
    if ((codeword&(1 << counter8)) >> counter8 == 1)
    {
            count10++;
    }
}
for (counter9 = 23; counter9 <= 28; counter9++)
{
    if ((codeword&(1 << counter9)) >> counter9 == 1)
    {
            count11++;
    }
}

// Adds counts10 and count11 and stores result to count12.
count12 = count10 + count11;

// If count12 is even, sets e8 to 0.
if (count12 % 2  == 0)
{
    e8 = 0;
}
// If count12 is odd, sets e8 to 0.
else
{
    e8 = 1;
}
```

```
    // Counts parity bits of Bit 16 of codeword.
    for (counter10 = 15; counter10 <= 28; counter10++)
    {
        if ((codeword&(1 << counter10)) >> counter10 == 1)
        {
                count13++;
        }
    }

    // If count13 is even, sets e16 to 0.
    if (count13 % 2 == 0)
    {
        e16 = 0;
    }
    // If count13 is odd, set to 1.
    else
    {
        e16 = 1;
    }

    // If all parity bits are 0, error is set to 0.
    if (e1 == 0 && e2 == 0 && e4 == 0 && e8 == 0 && e16 == 0)
    {
        error = 0;
    }

    // If there is one error in parity bits, it stores the location of the error in
 variable error.
    else
    {
        error = error | (e1 << 0);
        error = error | (e2 << 1);
        error = error | (e4 << 2);
        error = error | (e8 << 3);
        error = error | (e16 << 4);

    }

    // If there is an error, it will fix the error by making the bit a 0 or a 1 dep
ending on what the bit is.
    if (error > 0 && error <= 29)
    {
        if ((codeword&(1 << (error-1))) >> (error-1) == 1)
        {
                codeword = codeword & (~(1<<(error - 1)));
        }
        else if ((codeword&(1 << (error-1))) >> (error-1) == 0)
        {
                codeword = codeword | (1<< (error - 1));
        }
    }

    // Stores bits 29-0 with exception of parity bits to 24-bit decodeword.
    for (counter13 = 28; counter13 >= 16; counter13--)
    {
        if ((codeword&(1<<counter13)) >> counter13 == 1)
        {
                decodeword = decodeword | (1 << counter14);
        }
        counter14--;
    }
    for (counter15 = 14; counter15 >= 8; counter15--)
    {
        if ((codeword&(1 << counter15)) >> counter15 == 1)
        {
                decodeword = decodeword | (1 << counter16);
        }
        counter16--;
    }
    for (counter17 = 6; counter17 >= 4; counter17--)
    {
        if ((codeword&(1 << counter17)) >> counter17 == 1)
        {
                decodeword = decodeword | (1 << counter18);
        }
        counter18--;
    }
    if ((codeword&(1 << 2)) >> 2 == 1)
    {
        decodeword = decodeword | (1 << 0);
    }

    // Stores variable decodeword into three variables to decode codeword into thre
e different words: first_word, second_word, third_word.
    for (counter20 = 7; counter20 >= 0; counter20--)
    {
        if ((decodeword&(1<<counter20)) >> counter20 == 1)
        {
                first_letter = first_letter | (1 << counter20);
        }
    }

    for (counter21 = 15; counter21 >= 8; counter21--)
    {
        if ((decodeword&(1 << counter21)) >> counter21 == 1)
        {
                second_letter = second_letter | (1 << counter22);
        }
        counter22--;
    }

    for (counter23 = 23; counter23 >= 16; counter23--)
    {
        if ((decodeword&(1 << counter23)) >> counter23 == 1)
        {
                third_letter = third_letter | (1 << counter24);
        }
        counter24--;
    }

    // Prints the error of parity bits.
    printf("E1 : %d\n", e1);
    printf("E2 : %d\n", e2);
    printf("E4 : %d\n", e4);
    printf("E8 : %d\n", e8);
    printf("E16: %d\n", e16);

    // If there is no parity bits error, this will be displayed.
    if (error == 0)
    {
        printf(" No error\n");
    }
    // If there is an error between values 1 and 29, message will be displayed.
    else if ((error > 0 && error <= 29))
    {
        printf(" Corrected bit: %d\n", error);
    }
    // If error is greater than 29, there will be a decoding error.
    else
    {
        printf(" Decoding failure: %d\n", error);
    }

    // Prints decodeword in binary.
```

```
    printf(" -------- ");
    for (counter25 = 23; counter25 >= 0; counter25--)
    {
        printf("%d", (decodeword&(1<< counter25)) >> counter25);
        if (counter25 == 16)
        {
                printf(" ");
        }
        if (counter25 == 8)
        {
                printf(" ");
        }
    }

   printf("\n");

    // Prints information in hex.
    printf(" Information Word: 0x%.6X", decodeword);

    // Displays the codeword entered into three different letters.
    if ((first_letter & 0x80) == 0 && isprint(first_letter))
        printf(" (%c", first_letter);
    else
        printf(" ( ");
    if ((second_letter & 0x80) == 0 && isprint(second_letter))
        printf("%c", second_letter);
    else
        printf(" ");
    if ((third_letter & 0x80) == 0 && isprint(third_letter))
        printf("%c)\n", third_letter);
    else
        printf(" )\n");
    printf("\n");
}
```