Chapter 10: Structures, Unions and Bit Manipulations

Introduction
-------------
* structures are collections of related variables under one name.
* structures may contain variables of many different data types – in contrast to arrays that contain only elements of the same data type.
* similar to records to be stored in files.
* pointers and structures facilitate the formation of more complex data structures such as linked lists, queues, stacks, and trees.

Definitions
------------
* e.g.
  ```c
  struct card
  {
    char *face;
    char *suit;
  };
  ```
* "struct" – structure definition
* "card" – structure tag, not structure type
* the structure type is "struct card"
* "face" and "suit" – structure members: can be variable of basic data types, arrays, pointers and other structures
* ";" is important to end the definition of structure
* a structure cannot contain an instance of itself, but a pointer can be included
* structure variables:
  ```c
  struct card a, deck[52], *cPtr;
  ```
* or incorporated into the "struct card":
  ```c
  struct card
  {
    char *face;
    char *suit;
  } a, deck[52], *cPtr;
  ```

Initializing Structures
-----------------------
* using initializer lists as with arrays
* e.g.
  ```c
  struct card a = {"Three", "Hearts"};
  ```
* member "face" to "Three", member "suit" to "Hearts"

Accessing Members
------------------
* structure member operator: "." – access via structure variable name
* e.g.
  ```c
  printf("%s", a.suit);
  ```
* structure pointer operator: "->" – access via structure pointer
* e.g.
  ```c
  printf("%s", cPtr->suit);
  ```
  ```c
  ```
  ```c
  ```
  equivalent to
  ```c
  printf("%s", (*cPtr).suit);
  ```
  ```c
  ```
  ```c
  ```
  e.g.
  ```c
  ```
```c

#include <stdio.h>

```c
```
typedef struct card Card;
* "Card" is alias for type "struct card", so it is structure type, not structure tag
* e.g.

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

struct card
{
    char *face;
    char *suit;
};

typedef struct card Card;

void fillDeck(Card *, char *[], char *[]);
void shuffle(Card *);
void deal(Card *);

main()
{
    struct card a;
    struct card *aPtr;

    a.face = "Ace";
    a.suit = "Spades";
    aPtr = &a;
    printf("%s %s %s
", a.face, " of ", a.suit,
            aPtr->face, " of ", aPtr->suit,
            (*aPtr).face, " of ", (*aPtr).suit);
}
```
srand(time(NULL));
fillDeck(deck, face, suit);
shuffle(deck);
deal(deck);
}

void fillDeck(Card *wDeck, char *wFace[], char *wSuit[])
{
    int i;
    for (i=0;i<52;i++)
    {
        wDeck[i].face = wFace[i%13];
        wDeck[i].suit = wSuit[i/13];
    }
}

void shuffle(Card *wDeck)
{
    int i,j;
    Card temp;
    for (i=0;i<52;i++)
    {
        j = rand() % 52;
        temp = wDeck[i];
        wDeck[i] = wDeck[j];
        wDeck[j] = temp;
    }
}

void deal(Card *wDeck)
{
    int i;
    for (i=0;i<52;i++)
    {
        printf("%5s of %-8s%c", wDeck[i].face, wDeck[i].suit,
               (i+1)%2?'\t':'\n');
    }
}

Unions
------
* members share the same storage space
* for different situations in a program, some variables may not be relevant, but other are
* a union shares the space instead of wasting storage on variables that are not being used
* members can be of any type
* the number of bytes used to store a union must be at least enough to hold the largest member
* only one member can be referenced at a time
* e.g.
  union number
  {
      int x;
float y;
}

Initializing Union
-------------------
* only with a value of the first union member
* e.g.
  union number value = {10};

Accessing Members
------------------
* same as structure
* e.g.
  union number value;

  value.x = 100;
  printf("%d", value.x);

  value.y = 100.0;
  printf("%f", value.y);

Bitwise Operators
-----------------
* bit is the basic representation in computer
* can be either "0" or "1"
* "unsigned" are normally used

* left shift (<<)
e.g.
  int y,x = 20;       /* 20(10) = 00010100(2) */
  y = x << 3;         /* 160(10) = 10100000(2) */
* left shift can be used a "quick" multiplication of 2^n

* right shift (>>)
e.g.
  int y,x = 20;       /* 20(10) = 00010100(2) */
  y = x >> 3;         /* 2(10) = 00000010(2) */
* right shift can be used a "quick" (integer) division of 2^n

* bitwise AND (&)
<table>
<thead>
<tr>
<th>Operands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
* AND can be used as mask: to hide some bits in a value while selecting other bits
* e.g.
  #include <stdio.h>
  void displayBits(unsigned value);
  
  main()
  {
    unsigned x;

    printf("Enter an unsigned integer: ");
    scanf("%u", &x);
    displayBits(x);
  }
void displayBits(unsigned value)
{
    unsigned c, displayMask = 1 << 15;

    printf("%7u = ", value);
    for (c=1;c<=16;c++)
    {
        putchar(value & displayMask ? '1' : '0');
        value <<= 1;
        if (c%8 == 0)
            putchar(' ');
    }

    putchar('\n');
}

* bitwise OR (|)

<table>
<thead>
<tr>
<th>Operands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

* OR can be used to set specific bits to 1 in an operand

* NOT or complement(~)

<table>
<thead>
<tr>
<th>Operands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

* NOT can be used as taking the one's complement of the operand

* bitwise Exclusive OR (^)

<table>
<thead>
<tr>
<th>Operands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

* exclusive OR can be used as a encode and decode process since for any bit-stream T,
  
  T ^ K ^ K = T, where K is the key

* so E = T ^ K can be considered as the encoded bit-stream of T by key K

* the decoder can recover the origin bit-stream T by

  E ^ K = T

* this is basically the same process of encoder

  e.g.

  void encoder(char T[], char K[], char E[])
  {
    int i;

    for (i=0;i<2;i++)
    {
        E[i] = T[i] ^ K[i];
    }
  }

* however, the encoded bit-stream, E, may not be represented by ASCII
* we can remedy the situation by padding zero in the beginning of each of four bits and add 33 (the first character code of ASCII)
* e.g.
void padding(char E[], char PE[])
{
    unsigned char lowmask = 15;
    unsigned char highmask = 15 << 4;
    int i;

    for (i=0;i<2;i++)
    {
        PE[i*2] = (E[i] & lowmask) + 33;
        PE[i*2+1] = ((E[i] & highmask) >> 4) + 33;
    }

    PE[i*2] = '\0';
}

* the encoding program is
* e.g.
#include <stdio.h>
void encoder(char T[], char K[], char E[]);
void padding(char E[], char PE[]);

main()
{
    char Text[3], Key[3], EncodedText[3], OutText[5];

    printf("Input the message: ");
    scanf("%s",Text);
    printf("Input the key: ");
    scanf("%s",Key);

    encoder(Text, Key, EncodedText);
    padding(EncodedText, OutText);

    printf("The secret message is: %s\n",OutText);
}

* each bitwise operator (except the bitwise complement operator) has a corresponding assignment operator:
    &=, |=, ^=, <<=, >>=

---
Enumberation Constants
-------
* An enumeration, introduced by the keyword "enum", is a set of integer constants represented by identifiers.
* The values in an "enum" start with 0, unless specified otherwise, and are incremented by 1.
* E.g.
    enum months {JAN, FEB, MAR, APR, MAY, JUN, JUL,
    AUG, SEP, OCT, NOV, DEC};
creates a new type, "enum months", in which the identifiers are set automatically to the integer 0 to 11.
* To number the months 1 to 12, use
    enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
    AUG, SEP, OCT, NOV, DEC};
* E.g.
    #include <stdio.h>

    enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};
main()
{
    enum months month;

    for (month = JAN; month <= DEC; month++)
        printf("%2d%11s\n", month, monthName[month]);

    return 0;
}

Exercise
========
1. Try to write the decoding program.
2. Try to decode "0&,&" by the key "00"