Electrónica Digital II

Santiago Rúa Pérez, PhD.

18 de septiembre de 2022

UNIONES, ESTRUCTURAS Y MANIPULACIÓN DE BITS EN C

Estructuras, Uniones y Manipulación de bits en C

Objetivos

- Crear y usar estructuras, uniones y enumeraciones.
- Entender la auto referenciación de las estructuras.
- Como operar con estructuras.
- Inicilizar estructuras.
- Usar typedef.
- Usar constantes enum.
- Manipulación de bits.

Estructuras en C

En C, las estructuras son tipos de datos agrupados bajo un mismo elemento. El objetivo es formar una variable nueva conformada por diferentes tipos.

```
1    struct employee{
2      char firstName[20];
3      char lastName[20];
4      unsigned int age;
5      char gender;
6      double hourlySalary;
7    }
8
```

La referenciación a si mismo dentro de una estructura puede darse siempre y cuando se utilicen punteros.

```
1    struct employee{
2      char firstName[20];
3      struct employee *teamLeaderPtr; // pointer
4    }
5
```

Este tipo de autollamado permite construir elementos mas complejos como listas enlazadas (*linked list*)

Inicialización y acceso a estructuras

Se define la estructura con el identificador y el nombre de la estructura. En el ejemplo se declara un variable aCard, un arreglo de estructuras deck y un puntero a dicha estructura.

```
1    struct card aCard, deck[52], *cardPtr;
2
```

- No estan almacenados en espacios continuos de memoria.
- Se pueden inicializar utilizando una lista de elementos en el orden de la definición.

```
1        struct card aCard = { "Three", "Hearts" };
2
```

- Se accede a miembros de la estructura usando $\cdot$ o $->$ cuando se tiene una variable o un puntero respectivamente.

```
1        printf("%s", aCard.suit); // displays Hearts
2        printf("%s", cardPtr->suit); // displays Hearts
3
```

- Las estructuras pueden ser pasadas valores individuales o puntero a la estructura.

Estructuras en C

Ejemplo

```
1   #include<stdio.h>
2   struct  card
3   {
4     char *face;
5     char *suit;
6   };
7   int main(void){
8     struct card aCard;
9     aCard.face = "Ace";
10    aCard.suit = "Spades";
11    struct card *cardPtr = &aCard;
12
13    printf("%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
14    cardPtr->face, " of ", cardPtr->suit,
15    (*cardPtr).face, " of ", (*cardPtr).suit);
16
17  }
18
```

Las estructura pueden pasarse a funciones completamente o valores de la estructura. Tambien paso por referencia.

Typedef en C

La palabra `typedef` sirve como mecanismo para crear sinónimos de variables previamente definidas.

```
1    typedef struct {
2      char *face;
3      char *suit;
4    }Card;
5
```

Una vez declarada el tipo de datos Card, ahora se puede utilizar para crear una variable.

```
1    Card deck[52]
2
```

Typedef en C - Ejemplo

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

#define CARDS 52
#define FACES 13

struct card {
  const char *face; // define pointer face
  const char *suit; // define pointer suit
};
typedef struct card Card; // new type name

void fillDeck(Card * const wDeck, const char *
    wFace[], const char * wSuit[]);
void shuffle(Card * const wDeck);
void deal(const Card * const wDeck);

int main(void)
{
  Card deck[CARDS]; // define array of Cards

  const char *face[] = { "Ace", "Deuce", "Three",
     "Four", "Five", "Six", "Seven", "Eight", "
     Nine", "Ten","Jack", "Queen", "King"};
  const char *suit[] = { "Hearts", "Diamonds", "
     Clubs", "Spades"};

  srand(time(NULL)); // randomize
  fillDeck(deck, face, suit); // load the deck
  shuffle(deck); // put Cards in random order
  deal(deck); // deal all 52 Cards
}

void fillDeck(Card * const wDeck , const char *
    wFace[], const char * wSuit[])
{
  for (size_t i = 0; i < CARDS; ++i) {
    wDeck[i].face = wFace[i % FACES];
    wDeck[i].suit = wSuit[i / FACES];
  }
}

void shuffle(Card * const wDeck)
{
  for (size_t i = 0; i < CARDS; ++i) {
    size_t j = rand() % CARDS;
    Card temp = wDeck[i];
    wDeck[i] = wDeck[j];
    wDeck[j] = temp;
  }
}

void deal(const Card * const wDeck )
{
  for (size_t i = 0; i < CARDS; ++i) {
    printf("%5s of %-8s%s",wDeck[i].face, wDeck[
     i].suit , (i + 1) % 4 ? " " : "\n");
  }
}
```

Uniones en C

Las uniones son tipos de datos derivados los cuales comparten el mismo espacio de memoria. Define un tipo de variable con diferentes connotaciones. Solo puede tener un valor al tiempo.

```
1   #include <stdio.h>
2   union number {
3     int x;
4     double y;
5   };
6   int main(void)
7   {
8     union number value;
9     value.x = 100; // put an integer
10
11    printf("% s\n% s\n% s\n % d\n\n% s\n % f\n\n\n",
12    "Put 100 in the integer member",
13    "and print both members.",
14    "int:", value.x,
15    "double:", value.y);
16
17
18    value.y = 100.0; // put a double
19    printf("% s\n% s\n% s\n % d\n\n% s\n % f\n",
20    "Put 100.0 in the floating member",
21    "and print both members.",
22    "int:", value.x,
23    "double:", value.y);
24  }
25
```

Operadores bit a bit

Son operadores que trabajan a nivel de bits en las estructura de datos.

Operator		Description
&	bitwise AND	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are *both* 1.
|	bitwise inclusive OR	Compares its two operands bit by bit. The bits in the result are set to 1 if *at least one* of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR (also known as bitwise XOR)	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are different.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent when the left operand is negative.
~	complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Operadores bit a bit - Ejemplo

```
1   #include <stdio.h>
2
3   void displayBits(unsigned int value); // prototype
4
5   int main(void)
6   {
7     unsigned int x; // variable to hold user input
8
9     printf("%s", "Enter a nonnegative int: ");
10    scanf("%u", &x);
11
12    displayBits(x);
13  }
14
15  void displayBits(unsigned int value)
16  {
17    // define displayMask and left shift 31 bits
18    unsigned int displayMask = 1 << 31;
19    printf("%10u = ", value);
20
21    // loop through bits
22    for (unsigned int c = 1; c <= 32; ++c) {
23      putchar(value & displayMask ? '1' : '0');
24      value <<= 1; // shift value left by 1
25      if (c % 8 == 0) { // output space after 8 bits
26      putchar(' ');
27    }
28  }
29  putchar('\n');
30 }
31
```

Campos de bit (Bit fields)

En C se puede especificar la cantidad de bits a utilizar para almacenar un miembro en una estructura. Estos solo pueden ser declarados en enteros o enteros no signados.

```
1    struct bitCard {
2      unsigned int face : 4;
3      unsigned int suit : 2;
4      unsigned int color : 1;
5    };
6
```

Es posible especificar un campo de bits sin nombre con el objetivo que se bloquee dicho espacio en memoria.

```
1    struct example {
2      unsigned int a : 13;
3      unsigned int   : 19;
4      unsigned int b : 4;
5    };
6
```

Tambien puede ser de tamaño cero para alinear el siguiente campo con la memoria.

Enumeración

Es un conjunto de números enteros representados por algun identificador. Comienzan en cero e incrementan de a uno, a no ser que se indique o inicialice de otra forma.

```
1  #include <stdio.h>
2
3  enum months {
4    JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
5  };
6
7  int main(void)
8  {
9    // initialize array of pointers
10   const char *monthName[] = { "", "January", "February", "March",
11     "April", "May", "June", "July", "August", "September", "October",
12     "November", "December" };
13
14   // loop through months
15   for (enum months month = JAN; month <= DEC ; ++month) {
16     printf("%2d %11s\n", month, monthName[month]);
17   }
18 }
19
```

Ejercicios

Encuentre el error

a) Assume that `struct card` has been defined containing two pointers to type `char`, namely `face` and `suit`. Also, the variable `c` has been defined to be of type `struct card` and the variable `cPtr` has been defined to be of type pointer to `struct card`. Variable `cPtr` has been assigned the address of `c`.

```
printf("%s\n", *cPtr->face);
```

b) Assume that `struct card` has been defined containing two pointers to type `char`, namely `face` and `suit`. Also, the array `hearts[13]` has been defined to be of type `struct card`. The following statement should print the member `face` of array element 10.

```
printf("%s\n", hearts.face);
```

c)
```
union values {
    char w;
    float x;
    double y;
};
union values v = { 1.27 };
```

d)
```
struct person {
    char lastName[15];
    char firstName[15];
    unsigned int age;
}
```

e) Assume `struct person` has been defined as in part (d) but with the appropriate correction.

```
person d;
```

f) Assume variable `p` has been declared as type `struct person` and variable `c` has been declared as type `struct card`.

```
p = c;
```

Ejercicios

- Escribe un programa que invierta el orden de los bits en un dato entero no signado. El programa debe ingresar el valor del usuario y llamar a la función reverseBits para imprimir los bits en orden inverso. Imprima el valor en bits antes y después de que los bits sean invertido para confirmar que los bits se invierten correctamente.
- Crear una union floatingPoint con miembros float f, double d y long double x. Escriba un programa que ingrese valores de tipo float, double y long double y almacene el valor en variables de unión de tipo union floatingPoint. Cada variable de unión debe imprimirse como un flotador, un doble y un doble largo. ¿Los valores siempre se imprimen correctamente?
- Investigue el algoritmo de barajado de Fisher-Yates en línea, luego úsalo para reimplementar el método aleatorio

UNIONES, ESTRUCTURAS Y MANIPULACION DE BITS EN C

GRACIAS