Paper 102: Programming & Problem solving through C

Lecture-19:Unit-III Structures-II

Self referential structures

- It is sometimes required to include one member within a structure that is a pointer that points to the parent structure.

```
struct tag {
          member 1;
          …
          struct tag *name;
          }
```

- struct employee

```
{
      int ssno;
      char ename[25];
      float salary;
      struct employee *next;
};
```

SRS-example

- struct employee

```
{
    int ssno;
    char ename[25];
    float salary;
    struct employee *next;
};
```

Self referential structures

- Self referential structures are very useful in applications that involve linked data structures, such as list and trees.
- The basic idea of a linked data structures is that each component within the structure includes a pointer indicating where the next component can be found

Linked data structures

- The relative order of the components can easily be changed, simply by altering the pointers.
- In addition the individual components can be added or deleted, by altering the pointers
- The data structure can expand or contract in size as required.
- There are different kinds of linked data structures
  - Linear
  - Circular
  - trees

Bit Fields

- In some applications it is desirable to work with data items that consist of only a few bits
  - A flag for true or false
  - A 3 bit integer value ranging from 0 to 7
- Several such data items can be packed into an individual word of memory
- The word is divided into individual bit fields
- These bit fields are defined as members of a structure
- Each bit can be accessed individually, like any other structure members.

Bit Fields specification

- In general the decomposition of a word into distinct bit fields are as follows
- struct [<struct type name>]

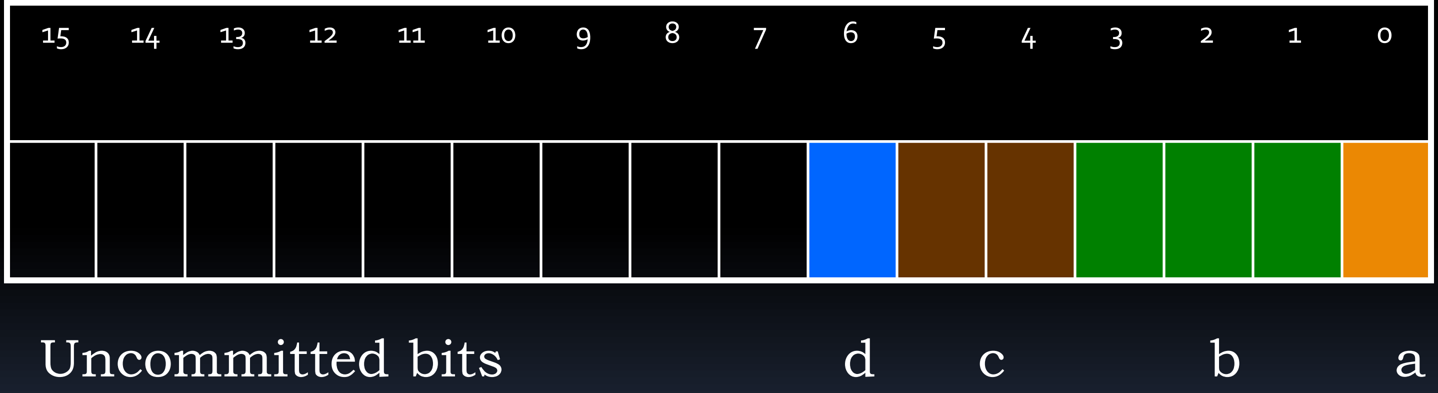
```
{
    [member 1: <size>];
    [member 2: <size>];
    …
} [<structure variable];
```

Bit Fields specification

- Each member declaration must include a specification indicating the size of corresponding bit field
- The member name must be followed by an unsigned integer indicating the size
- The interpretation of these bit fields vary from compiler to compiler
  - Left to right/ right to left

Bit Fields specification

```
struct sample {
unsigned a: 1;
unsigned b: 3;
unsigned c: 2;
unsigned d: 1;
} s;
```



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Uncommitted bits d c b a

Bit Fields cont…

- A field within a structure cannot overlap a word within the computer's memory
  - This issue does not arise if the sum of the field widths does not exceed the size of an unsigned integer quantity
- If the sum of the fields widths does exceed this word size, then the overlapping field will automatically be forced to the beginning of the next word

Unions

- Are similar to structures in almost all aspects except that
    - Each member within a structure is assigned its own unique storage space whereas the members of a union share the same storage space
- Unions are useful to applications involving multiple members, where each values need not be assigned to all of the members at any one time

Unions declaration

```
union [<union type name>]
{
   [<type> <variable-name> [, <variable-name>,…]];
    [<type> <variable-name> [, <variable-name>,…]];
   …
}  [<union variables.]
```

Storage-class union union-type variable [,variable…]

Unions declaration example

```
union id {
        char iname[12];
        int ino;
        } items;
```

Union and structure

- Union may be a member of a structure and a structure may also be a member of a union
- Accessing of union members is the same as accessing the members of structure members
- A union variable can be initialized provided the storage class is static or external
  - However only one member can be assigned an initial value
  - i.e., the first one that occurs in the declaration will get assigned

Enumeration

- Enumeration is a user-defined data type. It is defined using the keyword *enum* and the syntax is:

  enum tag_name {name_0, …, name_n} ;

- The tag_name is not used directly. The names in the braces are symbolic constants that take an integer values from zero through n. As an example, the statement:

  enum colors { red, yellow, green } ;

- creates three constants.
  - red is assigned the value 0
  - yellow is assigned 1
  - green is assigned 2.

Example

```
/* This program uses enumerated data types to access the elements of
      an array */
#include <stdio.h>
void main( )
{
int March[5][7]={{0,0,1,2,3,4,5},{6,7,8,9,10,11,12},
{13,14,15,16,17,18,19},{20,21,22,23,24,25,26},
{27,28,29,30,31,0,0}};
enum  days {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
      Saturday};
enum  week {week_one, week_two, week_three,week_four, week_five};
printf("Monday the third week  of March is March %d\n", March
      [week_three] [Monday] );
}
```

Class Assignments

- W.a.p to process student records by using structures
- W.a.p to maintain information of a cricket team. The information should be categorized into: runs achieved, wickets taken, catches taken, catches dropped and run outs. Each player should be designated as either a batsman or a bowler. The captain, vice captain and wicket keeper should have special status. Print the statistics of each player at the end of the season, giving all the information possible.