Structures and Unions

Structures

- A structure is collection of different types of data elements.
- We can group together integers, floats, chars, etc.. into
  one structure.
- By a structure we can represent a real world entity.
- It is a user defined data type. i.e. programmer can create its own data type which is a collection of different data types.

Structures

Two steps to use a structure:

1. Define a structure by declaring its members
   (creating a new data type)

2. Declare variable of that structure data type

Defining a Structures

Remember:

- A structure contains a number of data types grouped together.
- These [illegible] may not be of the same type.

Defining a structure:

```
struct
student    {
               char
               name[20];  int
               roll_no;  float
               per;
           };  char
```

Structure tag: name

By this we are defining a structure (creating a new data type) having 4 members,

name
roll_n
o per
fname

Which are of different types.

Name of this new data type is ***struct student***

Defining a Structures

Some more examples:

```
struct book_bank
{
    char name[20] ;
    float price ;
    int pages ;
} ;
```

```
struct country{
char name[20];
int population;
 char continent[10];
 ……..
  ……
};
```

```
struct class{
    int year;
    int semester;
    char branch[10];
} ;
```

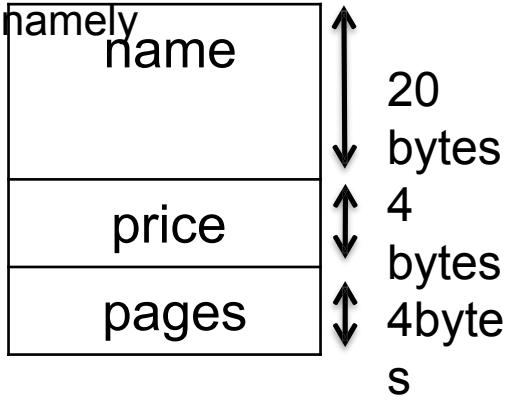
```
struct student{
char name[20];
int roll;
float per;
 ……
 ..
  ……
  ……
```

```
struct date{
    int dd;
    int m;
    int yyyy;
} ;
```

Declaring structure variables

```
struct book { char name[20] ;
              float price ; int pages ;
            } ;
int main()
 {
   struct book b1, b2, b3 ;
```

Every variable contains 3 members, namely



- **here 3 variables are declared of type "book"**

b1.name (20-bytes)	b2.name (20-bytes)	b3.name (20-bytes)
b1.price (4-bytes)	b2.price (4-bytes)	b3.price (4-bytes)
b1.pages (4-bytes)	b2.pages (4-bytes)	b3.pages (4-bytes)

Structures: Example

```
#include<stdio.h>

struct book
{   char name[20] ;
    float price ;
    int pages ;
} ;

int main( )
{   struct book b1, b2, b3 ;
    strcpy(b1.name, “Let Us C”);  b1.price=100;
    b1.pages= 354;
    strcpy(b2.name, “Computer Concepts”);
    b2.price=256.50;
    b2.pages= 682;
    printf ( "\nyou entered" ) ;
    printf ( "\n%s     %f    %d", b1.name, b1.price,
    b1.pages ) ;
    printf ( "\n%s     %f    %d", b2.name, b2.price,
    b2.pages ) ;
}
```

```
you entered
Let Us C   100.000000        354
Computer Concepts   256.500000682
```

Array of Structures: Example

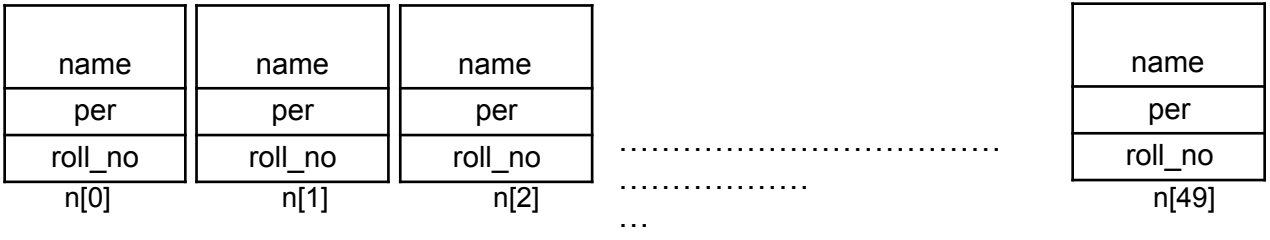
```
struct student
 {
      char name[10] ;
      float per ;
      int roll_no ;
 } ;
  struct student n[50]
```

Make a program to print name and roll number of students who are fail **per<50**

Print the name and roll numbers of students having percentage greater than 80

Calculate avg percentage of class

- ; **here an array of size 50 is declared of type "student"**



Accessing Structure members

```
struct book
{
  char name[20] ;   //Member1
  float price       //Member2
  ; int             //Member3
  } pages ;
  ;

struct book
book1;
struct book
book2;
```

Here **book1** has no meaning. Hence we write **book1's name**, **book1's price**, **book1's pages**

To access these members of book1 we use member operator **'.'** , also known as dot operator.

book1.name

book1.price

book1.pages

book2.name

Initializing Structure members

```
struct student
{
    char name[20];
    int roll_no;
    float per;
};
int main()
{
    struct student s1={“Rahul”, 10, 80};
    struct student s2={“Dinesh”, 4,
```

Initializing Structure members

```
struct student
        {
          char name[20];
          int roll_no;
          float per;
        };
int main()
 {
    struct student  s1={"Rahul", 10, 80};
    struct student s2={"Dinesh", 4, 56};
    .....
 }
```

- In C, the initialization within template is not permitted.
- The initialization must be done only in the declaration of actual variables.
- The order of values enclosed in braces must match the order of members in structure definition.
- Partial initialization will lead to uninitialized members as 0 or '\0'.
- The uninitialized members should be only at the end

Copying and Comparing Structures variables

```
struct student
{ char name[20];
  int roll_no;
  float per;
};
int main()
{ struct student s1={"Rahul", 10, 80}, s2;
  s2=s1;  // copying s1 into s2
  printf(" %s %d %f",s2.name, s2.roll_no,
  s2.per);

  // Rahul 10 80.000000
```

But we can not use logical operators.

- Structure variables cannot be compared using == or != operators.
- We can do it by comparing members individually.

Declaring Structure Variables

```
struct
student
        {  char
            name[20];
            int roll_no;
            float per;
int main()      };
{
  struct student s1, s2,
   s3;  struct student
   s[40];
  ……
   ……..
}
```

```
struct
student
        {  char
            name[20];
            int roll_no;
            float per;
struct          };
student  int    st1,st2;
main()          //global
{
   ……
   ……..
}
```

```
struct
student
        {  char
            name[20];
            int roll_no;
            float per;
int main()     } s1, s2 ;
{
   ……
   …
   ……
   …..
}
```

Use of 'typedef'

```
typedef struct
                {
                   char
                    name[20];
                    int roll_no;
                    float per;
int main()       } student;
{
   student s1, s2,
   s3;  student
   s[40];
   ……
    ……..
}
```

```
typedef
  struct
        {  char
            name[20];
            int roll_no;
            float per;
          } student;
student    st1,st2;
//global  int main()
{
    ……
    ……..
}
```

Nested Structure in C

- nesting one structure within another structure

```
#include<stdio.h>
struct address
{
   char city[20];
   int pin;
   char phone[14];
};
struct employee
{
   char name[20];
   struct address
}; add;
```

```
int main ()
 {
  struct employee emp;
  printf("Enter employee information?\n");
  scanf("%s %s %d
        %s",emp.name,
        emp.add.city,
        &emp.add.pin,
        emp.add.phone);
  printf("name: %s\nCity: %s\nPincode: %d
        \nPhone: %s",emp.name,emp.add.
         emp.add.pin,emp.add.phone);
 }
```

Separate structure

```
struct Date
{
   int dd;
   int mm;
   int yyyy;
};
struct Employee
{
   int id;
   char name[20];
   struct Date doj;
}emp1;
```

Embedded structure

```
struct Employee
{
   int id;
   char name[20];
   struct Date
    {
      int dd;
      int mm;
      int yyyy;
    }doj;
}emp1;
```

```
emp1.doj.dd
emp1.doj.mm
```

emp1.doj.yyyy

Passing structure in function

```
sturct st{
        int x;
        float
      }; y;

Int main()
  { struct st
      a;
      a.x=10;
      a.y=36.4
      ;
modifiy(a);
    printf(" %d",
    a.x);
  printf(" %f",
a.y);
modifiy2(&a);
printf(" %d", a.x);
```

```
void modify(struct st
A)
 {
   A.x=20;
    A.y=A.y+10;
  printf(" %d", A.x);
   printf(" %f", A.y);
}
```

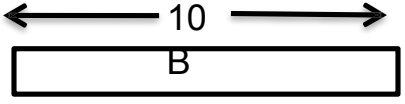
```
void modify2(struct st *p)
 {
     p->x=20;
      p->y=p->y +10;

}
```

Union

- Union is the collection of different types of data elements.
- We can group together integers, floats, chars,
  etc.. into one union just like structure.
- By a Union we can represent a real world entity.

It is a user defined data type. i.e. programmer can create its own data type which is a collection of different data types.

```
union A
{
   char name[10]
   ;  float per ;
   int roll_no ;
} ;
 union A a1,a2 ;
```



Union variable is same as structure variable except one difference that

- each member of a structure variable has its own space

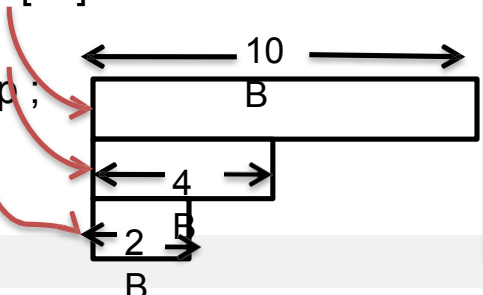
while in case of union variable a common space is

Difference between Structure and Union

structure	union
1. The keyword **struct** is used to define a structure.	1. The keyword **union** is used to define a union.
2. Each member of a structure variable has separate storage space.	2. A common storage space is shared by the members of a union variable.
3. Individual members can be accessed simultaneously.	3. At a time only one member can be accessed of a union variable.
4. Several members can be initialized at the time of declaration.	4. Only first member can be initialized at the time of declaration.

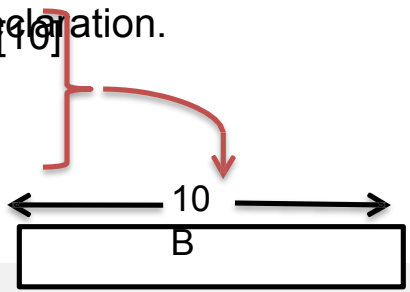
```
struct st
  {  char n[10] ;
     int i ;
     float p ;
  } ;
struct st s;
```

```
struct st s ={ "ram", 70.5,4};
```




```
union un
  {  char n[10] ;
     float p ;
     int i ;
  } ;
union un u;
```

```
union un u ={ "ram"};
```

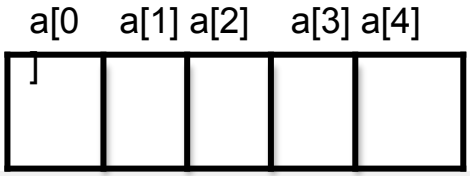


Difference between Structure and Array

Structure	Array
1. It is a collection of different types of data elements.	1. It is a collection of same type of data elements.
2. It is a user defined data type.	2. It is a derived data type.
3. Structue members are accessed using dot(.) operator.	3. Array members are used using index in array.
4. In order to use the structure, first we need to define the structure then by declaring variable of that structure type we can use it.	4. Array variable can be used as soon as it is declared.

```
struct st
{
    char n[10];
```

```
int
a[5];
```

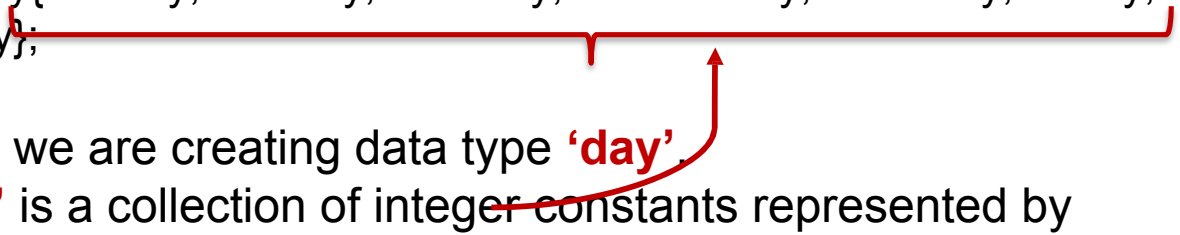


Enumerated Data Type

- Enumeration is a user defined data type in ‘C’ language.
- It is used to assign names to the integral constants which makes a program easy to read and maintain.
- The keyword “enum” is used to declare an enumeration.

Syntax:

```
enum day{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
```



- Here we are creating data type **‘day’**.
- **‘day’** is a collection of integer constants represented by identifiers.
- These are called **enumerators**.
- The identifiers are set automatically to the integers 0 to 6.

Enumerated Data Type

Synta
x: **enum day{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};**

0 1 2 3 4 5 6

- The identifiers are set automatically to the integers 0 to 6.
- By default, the first enumerator has a value of 0.

printf(" %d %d %d %d %d %d %d", Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

// 0 1 2 3 4 5 6

- Enum is a set of named constants called **enumerators**.
- Creating object/variable

enum day d1=Sunday;

Enumerated Data Type

Syntax: 1 2 3 4 5 6 7

enum day{Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

printf(" %d %d %d %d %d %d %d", Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

// 1 2 3 4 5 6 7

- Enum is a set of named constants called **enumerators**.
- Creating object/variable

  enum day d1=Sunday;

  printf("Day: %d", d1); **// Day 1**

Enumerated Data Type

1 2 3 4 5 6 7 8 9 10 11 12

```
enum month{Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
Nov, Dec};

  enum month m1=Mar;
  printf("%d",m1); // 3
```