[DATA STRUCTURES]
Chapter - 03 : “Structures”

▪ STRUCTURES

“STRUCTURES are arrays containing dissimilar data types”.

A **Structure** contains a number of data type grouped together. These data types may or may not be of the same type.

Need of structures :

Consider the declaration of following data’s-

```
int n;   ----- primary or independent variable.
int n[10]; ---- single dimension array.
int n[4][3]; ---- double dimension array.
```

In the above examples we can store only one type of data. But if we want to store the data more than one type (for e.g., the data of a student) then we declared as –

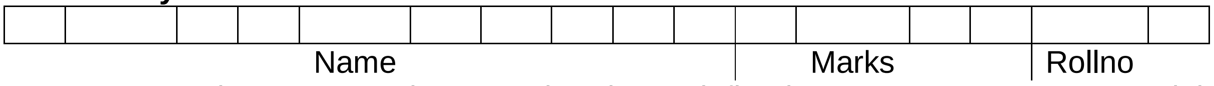
```
char name[11];
int rollno;
float marks;
```

Here we have to use large amount of memory if we declare this again and again the program also became lengthy.
All the above three data types are in different address. Hence it is wrong approach.
To remove this disadvantage we have to declare arrays that can store different data types. (These are called **structures**.)

Structure Declaration

```
struct student    ——    < tag name>
{
char name[10];┐
float marks;  ├────────          (structure elements)
int rollno:   ┘
}s; ─────────────────── (structure variable)
```

In memory: -



Note:- Once the structure data type has been defined one or more structure variables can be declared to be of that type. e.g. structure variables of above structure can be declared as: -

```
struct student s1,s2,s3;
```

Here the struct student have **16 bytes** declaration (10 for name+4 for marks+2 for rollno)
Hence each structure variable s1, s2, s3 consume 16-16 bytes.

Hence structure can be declared by any of following method: -

M –1: -
```
struct book
{
char name[10];
float price;
int page;
};
struct book b1,b2,b3;
```

M – 2: -
```
struct book
{
char name[10];
float price;
int page;
} b1,b2,b3;
```

M – 3: -
```
struct
{
char name[10];
float price;
int page;
} b1,b2,b3;
```

Accessing structure elements: - For accessing the structure elements we use dot operator(.). e.g. in the above structure to access the pages we write as –

b1.page similarly b1.price; and b1.name;

Note:- Before the dot operator structure variable (structure name) is written and after the dot operator there must be structure element.

Hence to accept values in structure elements we use command as –

```
scanf("%s%f%d",b1.name,&b1.price,&b1.page);
```

Similarly for printing the values of structure elements:-

```
        printf("%s%f%d",b1.name,b1.price,b1.page);
```

▪ UNION

Union is a concept borrowed from structures and therefore follow the same syntax as structures. However, there is a major distinction between them in terms of storage. In structures, each member has its own storage location, whereas all the members of a union use the same location. This implies that, although a union may contain many members of different types, it can handle only one member at a time. Like structures, a union can be declared using the keyword union as follows :

```
union item
{
 int m;
 float x;
 char c;
} code;
```

This declares a union variable **code** of type **union item**. The union contains three members, each with a different data type. However, we can use only one of them at a time. This is due to the fact that only one location is allocated for a union variable, irrespective of its size.

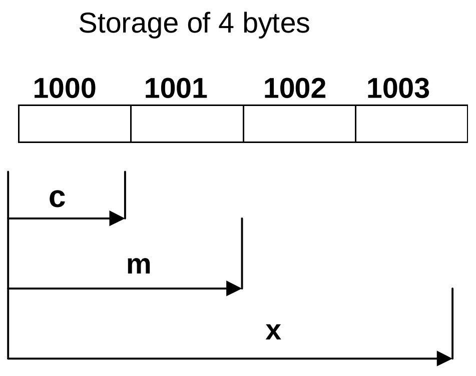


Fig. (2) Sharing of a storage locating by union members

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union. In the declaration above, the float member **x** requires 4 bytes which is the largest among the members. Fig (2) shows how all the three variables share the same address. This assumes that a float variable requires 4 bytes of storage.

To access a union member, we can use the same syntax that we use for structure members i.e,

```
code.m
code.x
code.c
```

are all valid member variables.

Program 1 : Structure initialization

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s1={"Amit",12,78.50},s2={"Kamal",17,88.50};
printf("\nNAME\tROLLNO\tMARKS\n");
printf("%s\t%d\t%.2f\n",s1.name,s1.rollno,s1.marks);
printf("%s\t%d\t%.2f",s2.name,s2.rollno,s2.marks);
getch();
}
```

Program 2 : WAP for structure input from user

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s;
printf("\nEnter the details of a student : \n");
printf("Name : ");
gets(s.name);
printf("Roll no. : ");
scanf("%d",&s.rollno);
printf("Marks : ");
scanf("%f",&s.marks);
printf("\nNAME\tROLLNO\tMARKS\n");
printf("%s\t%d\t%.2f\n",s.name,s.rollno,s.marks);
getch();
}
```

Program 3 : WAP to copy the contents of one structure to another

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s1={"Amit",12,78.50};
struct student s2;
s2=s1;
printf("\nNAME\tROLLNO\tMARKS\n");
printf("%s\t%d\t%.2f\n",s1.name,s1.rollno,s1.marks);
printf("%s\t%d\t%.2f",s2.name,s2.rollno,s2.marks);
getch();
}
```

Program 4 : Array of Structure initialization

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s[3]={{"Amit",12,78.50},{"Kamal",17,90.50},{"richa",22,78.50}};
printf("\nNAME\tROLLNO\tMARKS\n");
for(int i=0;i<3;i++)
 printf("%s\t%d\t%.2f\n",i,s[i].name,s[i].rollno,s[i].marks);
getch();
}
```

Program 5 : Array of structure input from user

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s[3];
int i;
float mks;
printf("Enter details of 3 students : ");
for(i=0;i<3;i++)
{
printf("\nStudent-%d : ",i+1);
printf("\nName : ");
fflush(stdin);
gets(s[i].name);
printf("Rollno : ");
scanf("%d",&s[i].rollno);
printf("Marks : ");
scanf("%f",&mks);
s[i].marks=mks;
}
printf("\nNAME\tROLLNO\tMARKS\n");
for(i=0;i<3;i++)
 printf("%s\t%d\t%.2f\n",s[i].name,s[i].rollno,s[i].marks);
getch();
}
```

Program 6 : Nested Structure

```
#include<stdio.h>
#include<conio.h>
struct common
{
char nm[20];
};

struct student
{
int rollno;
float marks;
struct common c;
};

struct employee
{
int age;
float salary;
struct common c;
};

void main()
{
clrscr();
int i;
struct student s;

printf("Enter the details of a student : \n");
printf("Name : ");
gets(s.c.nm);
printf("Rollno : ");
scanf("%d",&s.rollno);
printf("Marks : ");
scanf("%f",&s.marks);

struct employee e;
```

```c
printf("\nEnter the details of an employee : \n");
printf("Name : ");
fflush(stdin);
gets(e.c.nm);
printf("Age : ");
scanf("%d",&e.age);
printf("Salary : ");
scanf("%f",&e.salary);
clrscr();
printf("\nSTUDENT DETAILS : \n");
printf("NAME\tROLLNO\tMARKS\n");
printf("%s\t%d\t%.2f",s.c.nm,s.rollno,s.marks);

printf("\n\nEMPLOYEE DETAILS : \n");
printf("NAME\tAGE\tSALARY\n");
printf("%s\t%d\t%.2f",e.c.nm,e.age,e.salary);
getch();
}
```

Program 7 : WAP for structure pointer

```c
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s={"Amit",12,90.50};
struct student *ptr;
ptr=&s;
printf("\nNAME\tROLLNO\tMARKS\n");
printf("%s\t%d\t%.2f",s.name,s.rollno,s.marks);
printf("\n%s\t%d\t%.2f",(*ptr).name,(*ptr).rollno,(*ptr).marks);  // First method
printf("\n%s\t%d\t%.2f",ptr->name,ptr->rollno,ptr->marks);       // Second method
getch();
}
```

Program 8 : Array of Structure pointer

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int rollno;
float marks;
};

void main()
{
clrscr();
struct student s[3]={{"Amit",12,78.50},{"Kamal",17,90.50},{"richa",22,78.50}};
struct student *ptr;
ptr=&s[0];
printf("\nNAME\tROLLNO\tMARKS\n");
for(int i=0;i<3;i++,ptr++)
   printf("%s\t%d\t%.2f\n",ptr->name,ptr->rollno,ptr->marks);
getch();
}
```

Program 9 : Structure passing through function using call by value

```
#include<stdio.h>
#include<conio.h>
struct student
{
char nm[20];
int rollno;
float marks;
};

void display(struct student);

void main()
{
clrscr();
struct student s={"Rachit",16,89.50};
display(s);
getch();
}
```

```
void display(struct student s1)
{
 printf("\nNAME\tROLLNO\tMARKS\n");
 printf("%s\t%d\t%.2f",s1.nm,s1.rollno,s1.marks);
}
```

Program 10 : Structure passing through function using call by reference

```
#include<stdio.h>
#include<conio.h>
struct student
{
char nm[20];
int rollno;
float marks;
};

void display(struct student *);

void main()
{
clrscr();
struct student s={"Rachit",16,89.50};
display(&s);
getch();
}

void display(struct student *s1)
{
 printf("\nNAME\tROLLNO\tMARKS\n");
 printf("%s\t%d\t%.2f",s1->nm,s1->rollno,s1->marks);
}
```

Program 11 : Structure passing through function using mixed call

```
#include<stdio.h>
#include<conio.h>
struct student
{
char nm[20];
int rollno;
float marks;
};

void display(struct student *,int);

void main()
{
clrscr();
struct student s[3];
int i;
float mks;
printf("Enter details of 3 students : \n");
for(i=0;i<3;i++)
{
printf("\nStudent-%d : ",i+1);
printf("\nName : ");
fflush(stdin);
gets(s[i].nm);
printf("Rollno : ");
scanf("%d",&s[i].rollno);
printf("Marks : ");
scanf("%f",&mks);
s[i].marks=mks;
}
display(s,3);
getch();
}

void display(struct student *s1,int size)
{
 printf("\nNAME\tROLLNO\tMARKS\n");
 for(int i=0;i<size;i++,s1++)
  printf("\n%s\t%d\t%.2f",s1->nm,s1->rollno,s1->marks);
}
```