Unit-5

Structures

~~Definition:~~

In

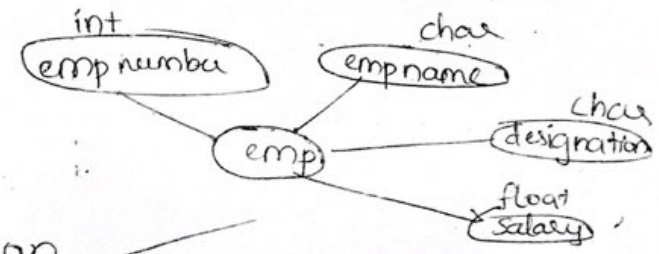
array can be used to represent the elements by a single name and by a single datatype.

→ If we want to represent the elements with different datatype we cannot use an array

→ To tackle this problem 'c' language provides a feature called "STRUCTURE" to pack the different datatypes into a group

definition

It is heterogeneous (different) collection of data items which shares a common name



declaration

Syntax:

```
struct structurename
{
    datatype member1;
    datatype member2;
    datatype member3;
    .
    .
};
```

eg:

```
struct country
{
    char cname[20];
    int population;
    char language[30];
};
```

eg:
```
struct emp
{
  int eno;
  char ename[20];
  char desi[20];
  float salary;
};
```

* Structure variable:

used to initialize and access the structure members

it is also called as object declaration

→ There are two ways

→ 1) In main function.

2) In the structure declaration.

1) syntax

```
struct structurename obj1, obj2 ..... objn;
```

2) In the structure declaration

syntax:

```
struct structurename
{
  datatype member1;
  datatype member2;
    }
  datatype membern;
}structure variable;
   (obj1, obj2 ..... objn);
```

1st way

1). eg:
```
struct emp
{
  int eno;
  char ename[20], Desi[20];
  float sal;
};
```

2)
```
struct country
{
  int population;
  char cname[20];
  char language[20];
};
struct contry c;
```

3)
```
struct
{
  int population;
  char cname[20];
  char language[20];
}c;
```

Initialization:

There are 2 types

1) Initialization during declaration (compile time)

Syntax:
```
struct structure name
{
   datatype m1;
   datatype m2;
   datatype m3;
} structure variable = {const1; const2; ---- constn};
```

eg) ①
```
struct emp
{
  int eno;
  char ename[20];
  char dsi[20]
  float sal;
} e={1912, "Raghu", "manager", 35,000.00};
```

eg:
```
struct country
{
  int population;
  char cname[20];
  char language[20];
} c={30000, "India", "Telugu"};
```

↓ No need to mention const here.

* Accessing of the structure variable

"." (dot operator (or) period operator (or) member op is used to access the structure variable

Syntax:

```
object name . member name;
```

structure variable name

Compile time initialization

```
eg: #include<stdio.h>
    #include<conio.h>
    struct country
    {
       int population;
       char ename[20];
       char lang[20];
    } c = {50000, "India", "Telugu"};
    void main()
    {
       clrscr();
       printf("population=%d", c.population);
       printf("ename=%s", c.ename);
       printf("lang=%s", c.lang);
       getch();
    }
```

Run time initialisation

```
#include<stdio.h>
#include<conio.h>
struct cdentry
{
  int pop;
  char cname[20];
  char lang[20];
};
void main()
{
  clrscr();
  printf("\nenter pop cname,lang");
  scanf("%d%s%s",&c.pop,&c.cname,&c.lang);
  printf("\n pop=%d",c.pop);
  printf("\n cname=%s",c.cname);
  printf("\n lang=%s",c.lang);
  getch();
}
```

→ in the structure already we are allocating the space that's why, we need to use '&' in scanf().

5) Write a c program to read and display the content of students like stu name, roll no, 3 subjects marks, average

```
#include<stdio.h>
#include<conio.h>
struct student
{
  char name[30];
  int rno;
  int m1, m2, m3;
};
void main()
{
  printf("\n student name=%s",s.sname);
  printf("\n student rno=%d",s.rno);
  int sum;
  float avg;
  clrscr();
  printf("\n enter student name, rno, m1, m2, m3");
  scanf("%s%d%d%d%d",s.sname,&s.rno,&s.m1,&s.m2,&s.m3);
  sum=(s.m1+s.m2+s.m3);
  avg=sum/3;
  printf("\n sum=%d",sum);
  printf("\n avg=%f",avg);
  getch();
}
```

* Structure within Structure (or) Nested Structures

Creating a structure inside another structure is called nested structure.

Consider the following example

```
struct emp
{
    int eno;
    char ename[36];
    float sal;
    float da;
    float hra;
    float ca;
};
```

This is structure that defines eno, ename, sal and 3 kinds of allowance

```
Struct emp
{
 int eno;
 char ename[30];
 float sal;
 Struct allowance
 {
    float da;
    float hra;
    float ca;
 }a;
};
```

Eg:

e.eno; e.ename e.sal

Program

```
#include <stdio.h>
#include <conio.h>
struct emp
{
  int eno;
  char ename[30];
  float sal;
  struct allowance
  {
      float da;
      float hra;
      float ea;
   }a;
}e;
void main()
{
    clrscr();
    printf("enter eno, ename, salary");
    scanf("%d %s %f", &e.eno, e.ename, &e.sal);
    printf("enter da, hra, ea values");
    scanf("%f %f %f", &e.a.da, &e.a.hra, &e.a.ea);
    printf("\n employee details are");
    printf("\n number = %d", e.eno);
    printf("\n name = %s", e.ename);
    printf("\n salary = %f", e.sal);
    printf("\n dearness allowance = %f", e.a.da);
    printf("\n house rent allowance = %f", e.a.hra);
    printf("\n city allowance = %f", e.a.ea);
    getch();
}
```

* Array of structures:

The most common used of structure is array of structures. To declare an array of structures, first the structure must be defined and an array of that type

eg: struct book b[10]; 10 elements in an array of structures of type 'book'

6) Write a program for accessing and printing details of 10 students.

```
# include<stdio.h>
#include <conio.h>
Struct student
{
        int sno;
        char sname[30];
        float marks;
};
void main()
{
        struct student s[10];
        int i;
        clrscr();
        for(i=0;i<10;i++)
        {
        printf("enter details of student %d ",i+1);
        scanf("%d%s%f", &s[i].sno, s[i].sname, &s[i].marks);
        }
```

```
for (i=0; i<10; i++)
{
  printf ("the details of student %d are", i+1);
  printf ("numbes = %d", s[i].sno);
  printf ("name = %s", s[i].sname);
  printf ("marks = %f", s[i].marks);
}
getch();
}
```

* Pointer to structures:

It holds the address of the entire structure

Mainly these are used to create complex data structures such as linked lists, trees, graphs and so on.

The members of the structure can be accessed using a special operator called arrow operator (->).

Declaration:

Syntax: struct structurename * ptr;

eg: struct student *s;

Accessing:

syntax: ptr -> membername;

eg: s->sno, s->sname, s->marks

* Program:

```
#include <stdio.h>
#include <conio.h>
Struct student
{
    int sno;
    char sname[30];
    float marks;
};
void main()
{
    struct student s;
    struct student *st;
    clrscr();
    printf("enter sno, sname, marks");
    scanf("%d %s %f", &s.sno, &s.sname, &s.marks);
    st = &s;
    printf("details of the students are ");
    printf("Number = %d ", st -> sno);
    printf("name = %s", st -> sname);
    printf("marks = %f", st -> marks);
    getch();
}
```

* Structure and functions:

There are 3 ways by ~~to~~ which the values of structure can be transferred from one function to another.

1) passing individual members as arguments to functio each member is passed as an argument in the function call

-> They are collected independently in ordinary variable in function

eg:

```
#include<stdio.h>
#include<conio.h>
struct date
{
    int day;
    int mon;
    int yr;
};
void main()
{
        void display(int,int,int);
    void struct date d={02,01,2010};
    struct date d={
    clrscr();
    display(d.day,d.mon,d.yr);
    getch();
}
void display(int a, int b, int c)
{
  printf("day=%d",a);
 printf("mon=%d",b);
 printf("yr=%d",c);
}
```

* They are collected independently in ordinary variables in function

eg:

```
#include<stdio.h>
#include<conio.h>
struct date
{
    int day;
    int mon;
    int yr;
};
void main()
{
    void display(int,int,int);
    struct date d={02,01,2010};
    clrscr();
    display(d.day,d.mon,d.yr);
    getch();
}
void display(int a, int b, int c)
{
    printf("day=%d",a);
    printf("mon=%d",b);
    printf("yr=%d",c);
}
```

* Adv 2) passing the entire structure as an argument to
The n Name of the structure variable is given as argu
structu in function call. It is collected in another structu
first variable in called function
array Disadvantage: A copy of entire structure is crea
eg: again.
of typ Program:
6) wri #include <stdio.h>
detai #include <conio.h>
void display (struct date);
#; struct date
Str {
{ int day;
int mon;
int yr;
};
void main()
}; {
vc
{ struct date d = {02, 01, 2010};
display(d);
getch();
}
void display (struct date dt)
{
printf("day = %d", dt.day);
printf("month = %d", dt.mon);
printf("year = %d", dt.yr);
}

3) Passing the address of structures as an argument to function;

The Address of the structure is passed as an argument to the function.

It is collected in a pointer to structure in called function

Advantages:

1) No wastage of memory as there is no need of creating a copy again

2) No need of returning the values back as the function can accept access indirectly the entire structure and work on it

Program:

```
#include<stdio.h>
#include<conio.h>
struct date
{
    int day;
    int mon;
    int yr;
};
void main()
{
    struct date d={02, 01, 2010};
    display(&d);
    getch();
}
display(struct date *dt)
{
    printf("day =%d", dt->day);
    printf("month=%d", dt->mon);
    printf("year =%d", dt->yr);
}
```

Union

Union is a derived datatype which allows you declare different types of elements at single memory location under a unique name

→ The difference between Structure and union is memory space i.e Structure members have th own memory locations where as union mem have to share a common memory location.

→ The size of the union is the size of large datatype of its member.

→ union is a keyword used to declare un variables

→ We cannot access and read all the memb at a ~~ty~~ time

* Declaration of union

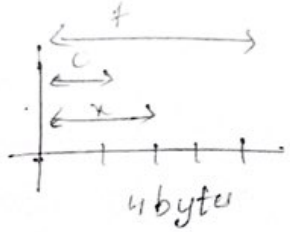
Syntax:

```
union unionname
  {
   datatype member1;
   datatype member2;
    -  -  -  -
    -  -
   datatype membern;
  } Variablename;
```

eg:

```
union allot
  {
   int x;
   char c;
   float f;
  } a;
```

* memory allocation:



8) write a 'C' program to display the size of union

program

```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct allot
      {
       int x;
       char c;
       float f;
      }a;
    union size
      {
       int b;
       char d;
       float e;
      }s;
    clrscr();
    printf("\n size of structure = %d \n size of union = %d",
                                    sizeof(a), sizeof(s));
    getch();
}
```

* typedef

→ typedef is a keyword used to assign an alternate name to existing datatype

i.e makes a synonim for existing datatype

Syntax:

typedef datatype identifier;

* definition

→ assign an alternative name to int

eg: typedef int eee;

where eee is a user defined datatype of integer type

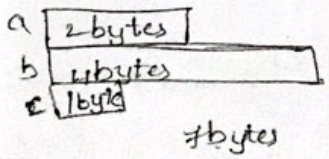
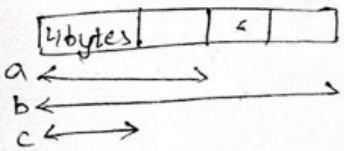
* sample code -

```
void main()
{
  typedef int hours;
  hours h;
  clrscr();
  printf("\n enter the hours");
  scanf("%d", &h);
  printf("\n minutes = %d", h*60);
  getch();
}
```

Ex:

```
typedef int num;
num a;
```

* Difference between union and structure:

Structure	Union
1) Definition: Structure is heterogeneous collection of data items grouped together under a single name	1) Definition: A union is a memory location that is shared by several variables of different datatypes
2) Syntax: Struct structurename { datatype member1; datatype member2; - - - - - - - - }obj;	2) Syntax: Union unionname { datatype member1; datatype member2; - - - - - - - - }obj;
3) Eg: Struct sample { int a; float b; char c; }s1;	3) Eg: Union sample { int a; float b; char c; }s2;
4) Keyword: struct	4) Keyword: union
5) Memory allocation  	5) Memory allocation 
6) Memory allocated is sum of sizes of all the datatypes in a structure (Here 7 bytes) 7) Memory is allocated for all the members of the structure differently	6) Memory allocated is the maximum size allocated among all datatypes in union (Here 4bytes) 7) Only one member will be residing in the ~~pr~~ memory at any particular ~~memo~~ location

* Union of Structures:

* A structure can be nested inside a union and it is called union of structure

* It is also possible to create a union inside a structure

Program:

```
#include <stdio.h>
#include <conio.h>
struct x
{
  int a;
  float b;
};
union z
{
  struct x s;
};
void main()
{
  union z u;
  clrscr();
  u.s.a=10;
  u.s.b=30.5;
  printf("a=%d",u.s.a);
  printf("b=%f",u.s.b);
  getch();
}
```

union z

Heap → struct x
{ int a;
  float b;
};
}u;

* Enumerated datatype:

→ These are used by the programmers to create their own data types and define what values the variables of these datatypes can hold.

Keyword: enum

Syntax:

```
enum tagname
{
  identifyer 1, identifyer 2, ..... n
};
```

eg: enum week{mon, tue, wed, thu, fri, sat, sun};

→ Here, with identifier values are constant unsigned integers and start from 0.

→ Mon refers 0, tue refers 1 and so on.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  enum week{mon, tue, wed, thu, fri, sat, sun};
  clrscr();
  printf("monday=%d", mon);
  printf("Thursday=%d", thu);
  printf("sunday=%d", sun);
  getch();
}
```

monday = 0
thursday = 3
sunday = 6

→ enum identifiers can also be assigned initial value

* Ar

The r

struct

first

array

eg:

of ty

6) wri

data

#

#

Str

{

Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
    enum week {mon=1,tue,wed,thu,fri,sat,sun};
    clrscr();
    printf("monday=%d",mon);
    printf("thussday=%d",thu);
    printf("sunday=%d",sun);
    getch();
}
```

output monday
thussday=
sunday=

* Bit fields.

→ These are used to change the order of allocation o memory from bytes to bits.

→ A bit field is a set of adjacent bits whose size can be from 1 to 16 bits in length.

→ There are occasions where data items require much less than 16 bits of space. In such cases memory will be wasted. Bit fields can pack several data items in a word of memory.

→ Syntax:

```
struct tagname
{
    datatype member1 : bit length;
    datatype member2 : bit length;
};
```

→ The datatype can be either int (or) unsigned int (or) signed int

→ Bit length specifies the number of bits

→ The largest value that can be stored is $2^n - 1$, where 'n' is bit length

Note:

1) Bit fields cannot be arrayed.
2) scanf() cannot be used to read values into bit fields
3) cannot use pointer to access the bit fields
4) Bit fields should be assigned values within the range of their size

Bit length	Range of values
1	0 to 1 $(2^1 - 1)$
2	0 to 3 $(2^2 - 1)$
3	0 to 7 $(2^3 - 1)$
n	0 to $2^n - 1$

eg:

1)
```
struct pack
{
    int count;
    unsigned a:2;
    unsigned b:3;
};
```

Here, count will be in 2 bytes. 'a' and 'b' will be packed into next 1 byte

eg. program

```c
#include<stdio.h>
#include <conio.h>
struct vehicle
{
    unsigned type:3;
    unsigned fuel:2;
    unsigned model:3;
}
void main()
{
    struct vehicle v;
    v.type=4;
    v.fuel=2;
    v.model=5;
    clrscr();
    printf("\n type of vehicle =%d", v.type);
    printf("fuel =%d", v.fuel);
    printf(" model = %d", v.model);
}
```

Note: instead of 6 bytes only 1 byte of memory will be allowed.

files

→ Scanf() and printf() are used to read and write the data.

→ They are console oriented input output functions which are used the terminals (Screen and keyboard) as target place
(printf() for screen, scanf() for keyboard)

→ In some cases there may be a need to handle large amount of data then the above functions