20CS1101 –PROGRAMMING FOR PROBLEM SOLVING

(Common to all branches)

UNIT – VI

STRUCTURES: Definition, Declaration and Initialization of Structures.

UNIONS: Definition, Declaration and Initialization of Union.

FILES: Introduction, File Types, Basic operations on Files, File I/O, Command Line Arguments

→STRUCTURES

6.1 Definition: A structures is a collection of one or more variables of different types, grouped together under a single name.

The variables or data items in a structure are called as members of the structures.

6.2 Declaration of structures: The general form or the syntax of declaring a structure is

```
struct <structure name>
{
data type member1;
data type member2;
.
.
.
data type member N;
};
```

In the above declaration, struct is a keyword followed by an optional user defined structure name usually referred as a tag. The list of member declaration is enclosed in a pair of flower braces. The closing brace of the structure and the semicolon ends the structure declaration.

Example:

```
struct  employee
{
int  empno;
char ename[10];
float salary ;
};
```

Where employee is the name of the structure. The structure employee contains three members of type int, char and float representing empno, empname and salary respectively.

6.3 Defining a structure: Defining a structure means creating variables to access members in the structures. Creating structure variables allocates sufficient memory space to hold all the members of the structures.

The syntax for defining the structure during structure its declaration is

```
struct <structure name>
{
data type member1;
data type member2;
.
.
.
.
data type   member N;
}structure-variable(s);
```

Example:

```
struct  employee
{
int  empno;
char ename[10];
float salary ;
} emp1;
```

Fig: Shows the memory allocation for the structure employee

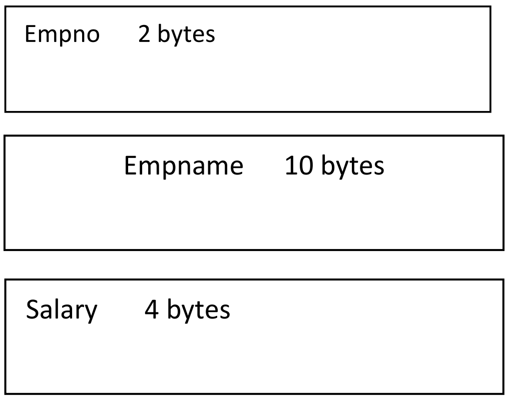


Fig: Memory occupied by structure employee

6.4 Accessing structure members: Dot (.) or period operator is used to access a data member of a structure variable. It is to be noted that the dot operator separates the structure variable and the data member.

Syntax: structure-variable . Member-name

Example: Write a C program to declaring, initializing and accessing members of structure by dot operator

```
void main()
{
struct  book
{
char title[40];
int pages;
float price;
};
struct book b1= { "pc&ds", 300,150.0};
printf("%s\n", b1.name);
printf("%d\n", b1.pages);
printf("%f\n", b1.price);
getch();
}
```

6.5 Initialization of structures: The structure members can be initialized only by using structure variables during structure declarations.

Syntax:

```
struct <structure name>
{
data type member1;
data type member2;
..
.
.
data type member N;
}structure-variable={ list of values};
```

Example:

```
struct  employee
{
int    height;
float weight ;
} emp1={154, 77.9};
```

The above initialization initialize 154 to the structure member height and 77.9 to the structure member weight. The values to be initialized for the structure members must be enclosed with in a pair of braces.

Example: Write a C program to initializing the structure using the structure name

```
#include<stdio.h>
#include<conio.h>
struct  student
{
int rollno;
char name[10];
}s3;
void main()
{
struct student s1= {123, "sai"};
struct student s2= {456, "kumar"};
printf("name is %s\n", s1.name);
printf("rollno is %d\n", s1.rollno);
printf("name is %s\n", s2.name);
printf("rollno is %d\n", s2.rollno);
s3=s1;
printf("name is %s\n", s3.name);
printf("rollno is %d\n", s3.rollno);
getch();
}
```

Example: Write a C program to assign the values to members

```
 #include<stdio.h>
#include<conio.h>
void main()
{
struct  student
{
int  rollno;
char name[10];
char branch[20];
};
struct student s
strcpy(s.name, "sai");
strcpy(s.branch, "cse");
s.rollno=32;
printf("%s %s %d\n", s.name, s.branch, s.rollno);
getch();
}
```

Example: Write a C program to assign the values from the keyboard

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct  account
{
int number;
char name[10];
float balance;
}a;
printf("Enter the account holder name");
gets(a.name);
printf("Enter the account number");
scanf("%d",&a.number);
printf("Enter the balance");
scanf("%f",&a.balance);
printf("The details are ");
printf("%s %d %f\n", a.name, a.number, a.balance);
getch();
}
```

6.6 Nested structures: Nested structures are nothing but a structure with in another structure. A structure may be defined and/or declared inside another structures.

Example:

```
struct employee
{
int empno;
char name[10];
struct  employ_add
{
int no;
char street[10];
char area[10];
long int pincode;
}address;

char deptname[10];
float salary;
}emp1, emp2;
```

In the above structure declaration employee is the main structure. It gets additional information about the employ_add. The member of a nested structure is accessed is

Main structure variable. Sub structure variable . Sub structure member

Example: emp1. address. no

Example: Write a C program on nested structures

```
#include<stdio.h>
#include<conio.h>
void main()
{
 struct  employee
{
int empno;
char name[10];
struct  employ_add
{
int no;
char street[10];
char area[10];
long int  pincode;
}address;
char deptname[10];
float salary;
}emp1;
printf("enter the empno, empname, deptname, salary of the employee");
scanf("%d %s %s %f",&emp1.empno, emp1.empname, emp1.deptname,&emp1.salary);
printf("Enter the address of the employee");
scanf("%d %s %s %s %ld", &emp1.addres.no, emp1.address.street, emp1.address.area, emp1.address.pincode);
printf("\n employee no=%d",emp1.empno);
printf("\n employee name=%s",emp1.empname);
printf("\n department name=%s",emp1.deptname);
printf("\n salary=%f",emp1.salary);

printf("employee address no=%d", emp1.address.no);
printf("employee street=%s", emp1.address.street);
printf("employee pincode =%ld", emp1.address.pincode);
getch();
}
```

6.7 Array of structures: It is possible to declare array of structures. The following example illustrates this.

```
Struct  employee_info
 {
    char name[30];
    int  age;
    char designation[10];
    float salary;
 }staff [200];
```

This declares staff to be an array with 200 elements. The process of declaring a structure array is similar to declaring any other kind of array.

Example: Write a C program to declaring array of structure student and displaying the marks of students who got more than 75 marks.

```
#include<stdio.h>
#include<conio.h>
Struct  student
 {
    int  rno;
    char sname[10];
    int  marks;
 };
void main()
{
 struct student s[60];
int i, n;
printf("How many students");
scanf("%d",&n);
for(i=0;i<n;i++)
 {
   printf("Enter the rollno");
   scanf("%d",&s[i].rno);
   printf("Enter the name");
   scanf("%s", s[i].name);
   printf("Enter the marks");
   scanf("%d",&s[i].marks);
 }
printf("Rollno    name    marks");
for(i=0;i<n;i++)
 {
   if(s[i].marks>=75)
   printf(" %d %s %d", s[i].rno, s[i].sname, s[i].marks);
 }
getch();
}
```

Array of structures can be initialized as

```
struct employee
{
int empno;
char empname[20], deptname[20];
float salry;
}emp[5]={ {1, "kumar" , "sales", 300.50},
        {2, "subbu" , "accounting", 400.50},
             {3, "manoj" , "marketing", 583.50},
             {4, "madhu" , "production", 88750.50},
            {5, "sudha" , "maintaintence", 7235.50}
        };
```

Note:If some of the members of the structures are not initialized it takes a value zero. If the member is a char data types , it takes a value NULL.

Example: Write a C program to initialize the array of structure

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
clrscr();
struct employee
{
int empno;
char empname[20];
float salry;
}emp[3]={ {1, "kumar",  300.50},
          {2, "subbu",  400.50},

        };
for(i=0;i<3;i++)
{
  printf("employee name=%s", emp[i].empname);
  printf("employee no=%d", emp[i].empno);
  printf("employee salary=%f",emp[i].salary);
}
getch();
}
```

6.8 Pointers and structures: Members of a structure can be accessed by a pointer. To access the structure members by pointer we use → (arrow) operator.

Example:

```
#include<stdio.h>
#include<conio.h>
struct book
{
  char title[10];
  int pages;
  float price;
};
void main()
{
struct book *ptr;
struct book b1;
ptr=&b1;
strcpy(ptr->title, "pc&ds");
ptr->pages=300;
ptr->price=150.0;
printf("title=%s",b1.title);
printf("pages=%d", b1.pages);
printf("price=%f", b1.price);
getch();
}
```

6.9 Functions and structures: There are three ways to pass a structure to a function

1. Passing structure members to functions
2. Passing the address of members to functions
3. Passing entire structure to function.

1. Passing structure members to functions: This method is used to pass members of the structure as actual arguments of the function call statement.

Example:

```
struct employee
{
int empno;
char empname[20];
char deptname[20];
float salary;
}emp1;
```

The members of the structure can be passed to the function employ ( ) as

```
employ (emp1.empno);
employ (emp1.empname);
employ (emp1.salary);
employ(emp1.deptname);
```

This method is the most common method and becomes inefficient when the structure is large.

Example: Write a C program to pass the structures members to functions

```
#include<stdio.h>
#include<conio.h>
struct  employee
{
int empno;
char empname[20];
}emp1;

void employ();
void main()
{
printf("\n enter the employee no and name");
scanf("%d %s", &emp1.empno, emp1.empname);
employ(emp1.empno,emp1.empname);
getch();
}
void  employ()
{
printf("\n the employee no is %d", emp1.empno);
printf("\n the employee name is %s", emp1.empname);
}
```

2. Passing the address of members to functions: Members of a structure can also be passed to a function by passing their address. In this method, the address location of the members is passed to the called function, hence the address operator (&) is used before the structure name. The members of the structure can be passed to a function employ ( ) as

```
employ (&emp1.empno);
employ (&emp1.salary);
employ (emp1.deptname);
employ (emp1.empname);
```

This method is more efficient than the previous one, since it, works faster, but becomes inefficient when the structure is large.

Example: Write a C program to pass the address of members to function

```
#include<stdio.h>
#include<conio.h>
struct  employee
{
```

```
int empno;
char empname[20];
}emp1;
void employ();
void main()
{
printf("\n enter the employee no and name");
scanf("%d %s", &emp1.empno, emp1.empname);
employ(&emp1.empno,emp1.empname);
getch();
}
void  employ()
{
printf("\n the employee no is %d", emp1.empno);
printf("\n the employee name is %s", emp1.empname);
}
```

3. Passing entire structure to function: In this method , the entire structure is passed as an argument to the function.

Example: Write a C program to pass the entire structure to functions

```
#include<stdio.h>
#include<conio.h>
struct  employee
{
int empno;
char empname[20];
}emp1;
void employ(struct employee emp);
void main()
{
printf("\n enter the employee no and name");
scanf("%d %s", &emp1.empno, emp1.empname);
employ(emp1);
}
void employ(struct employee emp)
{
printf("\n the employee no is %d", emp.empno);
printf("\n the employee name is %s", emp.empname);
getch();
}
```

6.10 Typedef: The typedef statement defines synonyms for an existing data type. It does not introduce a new data type and does not reserve storage also.

Syntax:

```
typedef   existing data type      new data type
```

Example:

```
typedef    float REAL;
REAL area, volume;
```

Example: Write a C program to create user defined data type hours on int data type and use it in the program.

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

6.11 Unions: A union is a derived data type, like structure, but only one data member is active at a time. In structure each member has its own memory location whereas, members of unions have same memory locations. Unions also contains members of types int, float , char , long, arrays, pointers……etc.

Syntax:

```
union <union name>
{
data type member1;
data type   member2;
.
.
.
data type memberN;
}union variable;
```

The syntax of union is identical to that of a structure except that the keyword struct is replaced with the keyword union.

Example:

```
union exam
{
int rollno;
char name[10];
int m1,m2,m3;
};
```

In the above example, union has 5 members. First member is a character array name having 10 characters (i.e., 10 bytes). Second member is of type int that requires 2 bytes for their storage. All the other members m1, m2, m3 are integers which requires 2 bytes for their storage.

The fig: shows the memory allocation of members of union exam

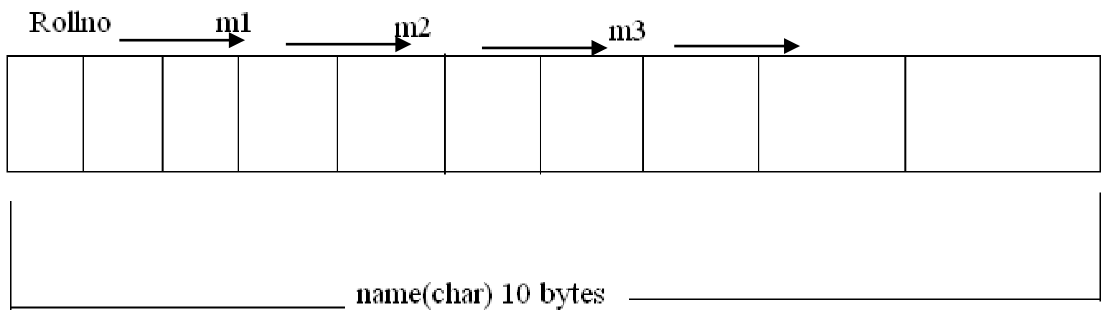


Fig:memory occupied by union exam

In union, all these 5 members are allocated in a common place of memory.

Example: Write a C program to show how many bytes are occupied by structure and union.

```
union exam
{
int rollno;
char name[10];
int m1,m2,m3;
}u1;
struct  exam1
{
int  rollno;
char name[10];
int   m1,m2,m3;
}s1;
void main()
```

```
{
printf("The size of union is %d\n , sizeof(u1));
printf("The size of structrue is %d\n", sizeof(s1));
getch();
}
```

Output:

```
The size of union is      10
The size of structure is 18
```

6.12 Initialization a union: There is a major difference between structure and union in terms of storage. In structure, each member has its own storage location where as all the members of union occupy the same memory location.

A union may contain many members of different data types; it can handle only one member at a time. Hence, we can initialize only one member in a union.

If we try to initialize more than one member, the last initialized value will be assigned to all of its members.

Example: Write a C program to demonstrate initialization of a union

```
union exam
{
int rollno,m1,m2,m3;
}u1;
void main()
{
u1.rollno=252;
u1.m1=89;
printf("\n Roll no=%d\n", u1.rollno);
printf("\n m1=%d\n", u1.m1);
printf("\n m2=%d\n", u1.m2);
printf("\n m3=%d\n", u1.m3);
getch();
}
```

Output:

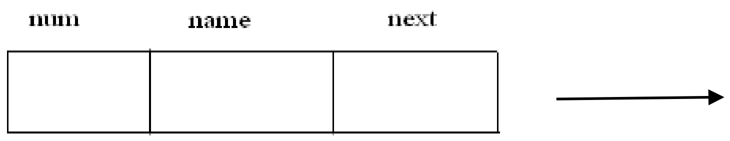
```
Rollno=89
M1=89
M2=89
M3=89
```

6.13 self-referential structures: In a structure, if one or more members are pointers pointing to the same structure, then that structure is called self-referential structure. In simple, a structure refers to itself is known itself referential structure.

Example:

```
Struct  node
{
int    num;
char name[10];
struct node *next;
};
```

Each node consists of three data items 1.number 2.name 3.next node. The pointer variable next is called a link. These structures are represented as follows:



The pointer "next" contains either an address of the location in memory of the successor node element or NULL. The NULL is used to denote the end of the list. We now declare three nodes as structure type node:

node n1, n2, n3;

We assign data values to these nodes:

```
n1.num=10; strcpy(n1.name, "ravi");
n2.num=20; strcpy(n2.name, "kumar");
n3.num=30;     strcpy(n3.name, "krishna");
```

Let us link these nodes together

```
n1.next=&n2;
n2.next=&n3;
n3.next=NULL;
```

These pointers assignments result in linking n1 to n2 to n3.

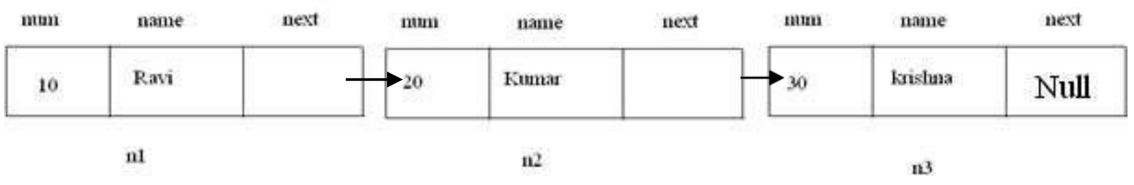


Fig: A linked list

Now the links allow us to retrieve data from successive nodes. Thus

n1.next->num and n1.next->name have values 20 and "kumar"

n1.next->next->num and n1.next->next->name have values 30 and "krishna"

Example: Write a C program on self-referential structures.

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct  node
{
int num;
char name[10];
struct node *next;
};

struct node n1,n2, n3;

n1.num=10; strcpy(n1.name, "ravi");
n2.num=20; strcpy(n2.name, "kumar");
n3.num=30; strcpy(n3.name, "krishna");

n1.next=&n2;
n2.next=&n3;
n3.next=NULL;

printf("\n %d %s", n1.next->num,n1.next->name);
printf("\n %d %s",n1.next->next->num,n1.next->next->name);
getch();
}
```

Output:
```
20  kumar
30  Krishna
```

6.14 Difference between structure and unions:

Structure	Union
1. The key word struct is used to declare a structure.	1. The key word union is used to declare an union.
2. All data members in a structure are active at a time.	2. Only one data member is active at a time.
3. All the members of a structure can be initialized.	3. Only the first of union can be initialized.
4. Each members in a structure occupies and use its own memory space.	4. All union members use the same memory space.
5. More memory space is required, since each member is stored in a separate memory locations.	5. Less memory space is required since all members are stored in the same memory locations.
Example: Struct book { Char title[40]; Int pages; Float price; }s; For s total memory required is 40+2+4 bytes.	Example: Union book { Char title[40]; Int pages; Float price; } s; For s total memory required is 40 bytes only.

6.15 Difference between arrays and structure:

Arrays	Structures
1.An array is a collection of data items of same data types.	1.A structure is a collection of data items of different data types.
2.The individual entries in an array are called elements.	2.The individual entries in a structure are called members.
3. An array declaration reserves enough memory space for its elements.	3.The structure definition reserves enough memory space for its members.
4.There is no keyword to represent arrays but the square braces [ ] preceding the variable name tell us that we are dealing with arrays.	4.The keyword struct tell us that we are dealing with structures.
5. Initialization of elements can be done during array declaration.	5. Initialization of members can be done only during the structure definition.
6.The elements of an array are store in sequence of memory location.	6.The members of a structure are not in sequence of memory location.
7.The array elements are accessed by using index or subscript.	7.The members of a structure are accessed by using dot operator.

8.Its general format is
Data type array name[size];
 	
9.Example: int sum[10];
	8.Its general format is :
	Struct <structure name>
	{
	Data type member1;
	Data type member2;
	.
	.
	Data type memberN;
	}structurevariable(s);
	
	9.Example:
	Struct student
	{
	Int rollno;
	Char name[10];
	}s1;

Example 1: Write a C program to display int , float and char values using switch case in structure.

```
void main()
{
union
{
int a;
float b;
char  c;
}s;

int  ch;
printf("\n 1.int 2.float 3. char");
printf("\n enter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nenter the integer value");
        scanf("%d", &s.a);
        printf("\n value=%d", s.a);
        break;
case 2: printf("\nenter the float value");
        scanf("%f", &s.a);
        printf("\n value=%f", s.b);
        break;
case 3: printf("\nenter the char");
        scanf("%c", &s.a);
        printf("\n char=%c", s.c);
        break;
}
getch();
}
```

Output:

```
1. Int 2. Float 3. Char
Enter the choice 1
Enter the value 2
Value=2
```

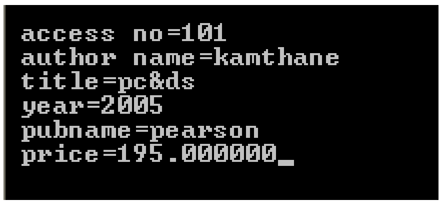
Example 2: Write a C program using a structure to create a library catalogue with the following fields.

1. Accessno 2.authors’ name 3.title 4. Year of publication 5.publisher name 6.price.

```
void  main()
{
struct  library
{
 int  accessno;
 char authorname[10];
 char  title[10];
 int year;
char pubname[10];
float  price;
};

struct library b1={101, "kamthane", "pc&ds" , 2005, "pearson" , 195.00};
clrscr();
printf("\n access no=%d", b1.accessno);
printf("\n author name=%s",b1.authorname);
printf("\n title=%s", b1.title);
printf("\n year=%d", b1.year);
printf("\n pubname=%s", b1.pubname);
printf("\n price=%f", b1.price);
getch();
}
```

Output:



```
access no=101
author name=kamthane
title=pc&ds
year=2005
pubname=pearson
price=195.000000_
```

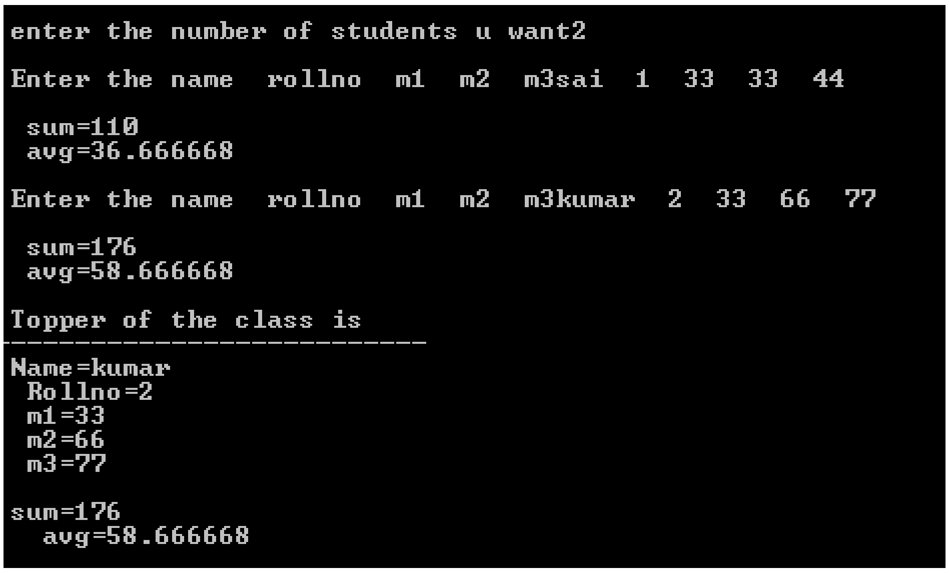
Example 3: Write a c program to find the topper of the student by using array of structures.

```
#include<stdio.h>
#include<conio.h>
void  main()
{
  struct  student
  {
    int rollno, m1,m2,m3,sum;
    char  name[10];
    float  avg;
  }s[10];

int i, n, t=0, k=0;
clrscr();
printf("\n enter the number of students u want");
scanf("%d", &n);
for(i=0;i<n;i++)
{
 printf("\n Enter the name rollno m1 m2  m3");
 scanf("\n%s%d%d%d%d", s[i].name, &s[i].rollno, &s[i].m1,&s[i].m2,&s[i].m3);
 s[i].sum=s[i].m1+s[i].m2+s[i].m3;
 s[i].avg=s[i].sum/3.0;
 printf("\n sum=%d\n avg=%f\n", s[i].sum, s[i].avg);
}

t=s[0].avg;
for(i=1;i<n;i++)
 {
   if(t<s[i].avg)
    {
      t=s[i].avg;
      k=i;
     }
 }
printf("\n Topper of the class is");
printf("\n...................................");
printf("\n  Name=%s\n  Rollno=%d\n   m1=%d\n   m2=%d\n   m3=%d\n  ", s[k].name, s[k].rollno,    s[k].m1,
s[k].m2,  s[k].m3);
printf("\n sum=%d\n avg=%f", s[k].sum, s[k].avg);
getch();
}
```

Output:



```
enter the number of students u want2

Enter the name  rollno  m1  m2  m3sai  1  33  33  44

 sum=110
 avg=36.666668

Enter the name  rollno  m1  m2  m3kumar  2  33  66  77

 sum=176
 avg=58.666668

Topper of the class is
--------------------------
Name=kumar
 Rollno=2
 m1=33
 m2=66
 m3=77

sum=176
  avg=58.666668
```

→**FILES:** Introduction, File Types, Basic operations on Files, File I/O, Command Line Arguments

6.16 Introduction: A file contains data/information which is stored permanently in a storage device. Generally used storage devices are CDs, DVDs & hard disks. When large quantity of data is required to be stored and processed, the concept of file is used.

A file stored in a storage device is always identified using a name(e.g. student. Dat, info.txt). Normally a filename has a primary name and a secondary name, which are separated by a dot(.).

STUDENT. DAT

Primary name separator secondary name

6.17 Definition: File is a set of records that can be accessed through the set of library functions.

6.18 Streams: Stream means reading and writing of data. The streams are designed to allow the user to access the files efficiently. A stream is a file or physical device like keyboard, printer and monitor.

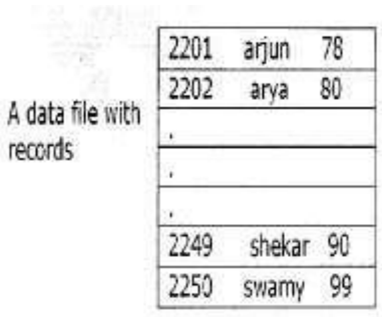
6.19 Types of files: Based on the type of data

1. Data file 2. Text file

Based on the accessing the data

1. Sequential file 2. Random file

1. Data file: A data file contains data stored in the form of records. A record is a collection of data related to a person or item. For example, a student record may contain data like roll number, student name and marks obtained by him. A file contain may such records. Imagine the records are arranged one by one as shown in figure.



2. Text file: A text file contains information stored in the form of string characters. The characters entered through the keyboard are stored continuously as illustrated bellow.

1. Sequence file: In sequential file data/information is stored sequentially one by another. The data is read in the same order in which they are stored.

2. Random access file: In random access file the data/information can be read randomly. Normally a key is used to identify the required record in random file accessing.

→Basic operations on Files

6.20 File declaration: A file is declared and the data is accessed using a file pointer. It has the following general form. Various functions used in the file operations.

Function	Operation
fopen( )	Creates a new file for read/write operation.
fclose()	Closes a file associated with the pointer.
closeall( )	Closes all opened files with fopen ( ).
fgetc( )	Reads the character from current pointer position and advances the pointer to next character.
getc( )	Same as fgetc ( ).
fprintf( )	Writes all types of data values to the file.
fscanf( )	Reads all types of data values from a file.
putc( )	Writes character one by one to a file.
fputc( )	Same as putc( ).
gets( )	Reads string from the file.
puts( )	Writes string to the file.
putw( )	Writes an integer to the file.
getw( )	Reads an integer from the file.
fread( )	Reads structured data written by fwrite ( ).
fwrite( )	Writes block of structured data to the file.
fseek( )	Sets the pointer position anywhere in the file.
feof( )	Detects the end of file.
ferror( )	Reports error occurred while read/write operations.
perror()	Prints compilers error messages along with user defined messages.
ftell( )	Returns the current pointer position.
rewind( )	Sets the record pointer at the beginning of the file.
unlink ( )	Removes the specified file from the disk.
remove( )	Removes the specified file from the disk changes the name of the file.

6.21 The file pointer (fp): A file pointer is a pointer to a structure of type FILE. It points to information that defines various things about the file, including its name, status and the current position of the file.

Example: FILE *fp;

6.22 fopen( ) function: The fopen ( ) function is used to open a file and set the file pointer to the beginning/end of a file. It has the following form.

Syntax:

fp= fopen("filename", "mode");

Where fp refers to the name of the file to be opened. Mode refers to the operation mode to access data. The following "mode" are used in data processing.

Mode	Meaning
r	Open a text file for reading.
w	Create a text file for writing.
a	Append to a text file.
rb	Open a binary file for reading.
wb	Creates a binary file for writing.
ab	Append to a binary file.
r+	Open a text file for read/wrtite.
w+	Create a text file for read/write.
a+	Append for create a text file for read/write.
r+b	Open a binary file for read/write.
w+b	Create a binary file for read/write.
a+b	Append or create a binary file for read/write.

6.23 fclose( ) function: All files that are opened should be closed after all input and output operations with the file. This is to prevent data from getting corrupted. fclose() function is used to close an active file. It has the following form.

Syntax:

```
int   fclose (FILE     *fp);
```

Example: fclose(fp);

Where fp refers to file pointer. The function return EOF if an error occurs then uses the standard function ferror ( ).

Example1: Write a C program to read the data from the keyboard, write it to a file called input, again read the same data from the input file, and displaying it on the screen.

```
#include<stdio.h>
void main()
{
FILE *fp;
int ch;

/* to store data in a file  */

printf("\nEnter the data");
fp=fopen("input.dat", "w");
while((ch=getchar())!=EOF)
putc(ch, fp);
fclose(fp);

/* to display data on the screen */

printf("\nThe data stored in input.dat file");
fp=fopen("input.dat", "r");
while((ch=getc(fp))!=EOF)
printf("%d",ch);
fclose(fp);
getch();
}
```

Output: welcome to Nbkrist

Example2: To write a C program to read in a line of lower case text from a file and display its upper case equivalent on the screen.

```
#include<stdio.h>
void main()
{
FILE *fp;
int ch;
clrscr();
fp=fopen("sample.txt", "r");
while((ch=getc(fp))!=EOF)
putchar(toupper(ch));
fclose(fp);
getch();
}
```

Output: first to create the file sample.txt and then write the text "hai"

The output is going to show HAI

Example: Write a C program to count chars, spaces, tabs and new lines in a file.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int ch;
int nol=1,not=0,noc=0,nob=0;
fp=fopen("srist.txt", "r");
while(1)
{
  ch=getc(fp);
  if(ch==EOF)
  break;
  noc++;
  if(ch==' ')
  nob++;
  if(ch=='\n')
  nol++;
  if(ch=='\t')
  not++;
}
fclose(fp);
printf("\n number of characters=%d",noc);
printf("\n number of blanks=%d",nob);
printf("\n number of tabs=%d",not);
printf("\n number of lines=%d",nol);
getch();
}
```

Output: First to create the file srist.txt and then write welcome to NbkristVidyanagar

```
Number of characters=26
Number of blanks=3
Number of tabs=1
Number of lines=1
```

→File I/O

6.24 fscanf( ): The fscanf( ) function is used to read data from a file. It is similar to the scanf( ) function except that fscanf( ) is used to read data from disk. It has the following form.

Syntax: fscanf(fp, “format string”, &v1,&v2,............... &vn);

6.25 fprintf( ): The fprintf( ) function is used to write data to a file. It is similar to printf( ) function except that fprintf( ) is used to write data to the disk. It has the following form.

Syntax: fprintf(fp, “format string” , v1,v2,.............. vn);

Example: Write a C program to create a file “student.txt” , contains information such as student roll number , name, total marks.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  FILE *fp;
  int i, n, rno, total;
  char sname[20];
  fp=fopen("student.txt", "w");
printf("\n how many students");
scanf("%d",&n);
for(i=0;i<n;i++)
{
  printf("\n Enter rno, sname, total");
  scanf("%d %s %d", &rno,sname,&total);
  fprintf(fp, "%d %s %d",rno, sname,total);
}
fclose(fp);
getch();
}
```

→6.26 Random access files: Random access files can be referred randomly. For this separate functions are available. The functions that are used in random access files are

1. fseek( ) , 2. ftell( ), 3. rewind( ).

fseek( ) function: The fseek( ) function is used to move the file pointer to any position in a file from a given reference position. It has the following form.

Syntax: fseek(fp, n, position);

Where "fp" is a file pointer "n"is a + or – long integer number that represent the number of bytes to be skipped and "position" is the position from which n bytes to be skipped.

0→ beginning (SEEK_SET)

1→ current (SEEK_CUR)

2→ end (SEEK_END)

Example: fseek(fp, 10, 0); or fseek(fp, 10, SEEK_SET);

The file pointer is repositioned in the forward direction by 10 bytes.

Example: Write a C program to illustrate fseek( )

```
#include<stdio.h>
void main()
{
 FILE *fp;
 int n;
 clrscr();
fp=fopen("data.txt", "r");
fseek(fp,4L, 0);
n=getc(fp);
while(n!=EOF)
{
  putchar(n);
  n=getc(fp);
}
fclose(fp);
getch();
}
```

6.27 ferror( ): The ferror( ) function determines whether a file operation has produced an error.

Syntax: int ferror(FILE *fp);

Where fp is a file pointer. It returns true if an error has occupied during the last file operation. Otherwise it returns false.

6.28 Writing a character ( ): The putc( ) function writes characters to a file that was previously opened for writing using the fopen( ).

Syntax: int putc(int ch, FILE *fp);

If a putc( ) function operation is successful, it returns the character written. Otherwise , it returns EOE.

6.29 Reading a character: The function getc( ) reads characters from a file opened in read mode by fopen( ).

Syntax: int getc(FILE *fp);

Where fp is file pointer of type FILE returned by fopen( ).

6.30 feof( ): The C file system includes the function feof( ), which determines when the end of the file has been encountered.

Syntax: int feof (FILE *fp);

feof( ) returns true if the end of the file has been reached. Otherwise it returns 0(zero).

→6.31 Command line arguments:

The function main( ) in C can also pass arguments or parameters like the other functions. It has two arguments argc (for argument count) and argv (for argument vector). It may have one of the following two forms.

void main(int argc, char *argv[ ])

or

void main(int argc, char **argv[ ])

Where argc represents the number of arguments, argv is a pointer to an array of strings or pointer to pointer to character.

The argument argv is used to pass strings to the programs. Hence the arguments argc and argv are called as program parameter.

The program name is then interpreted as an operation system command. Hence the line in which it appears is generally referred to as a command line.

Example: Program-name   parameter1, parameter2 .... parameter n

The individual items must be separated from one another either by blank space or by tabs. Some operating systems permit blank space to be included within a parameter provided the entire parameter is enclosed in quotation mark.

Example 1: sample red white blue

Argc= 4

Argv[0]= sample.exe

Argv[1]=red

Argv[2]= white

Argv[3]= blue

Example 2: sample   red   “white blue”

Argc= 3

Argv[0]= sample.exe

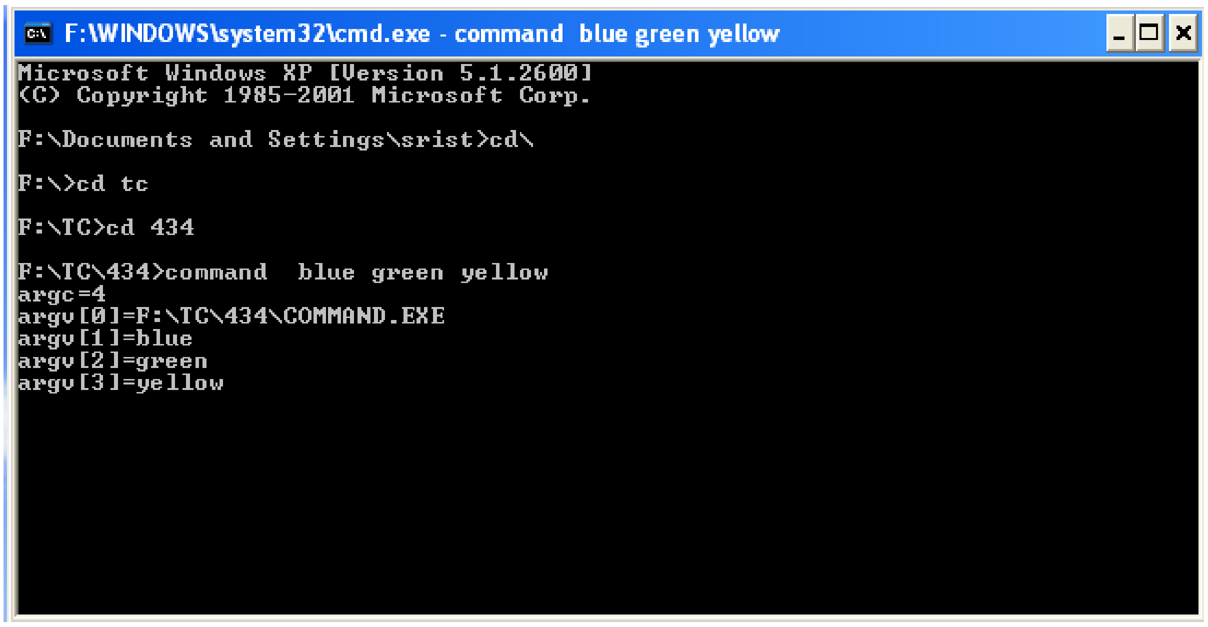
Argv[1]=red

Argv[2]= white blue

Example: Write a C program on command line argument

```
#include<stdio.h>
#include<conio.h>
void main(int argc, char *argv[ ])
{
int count;
printf("argc=%d\n", argc);
for(count=0; count<argc; ++count)
printf("argv[%d]=%s\n", count, argv[count]);
getch();
}
```

Output: go to the command prompt



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

F:\Documents and Settings\srist>cd\

F:\>cd tc

F:\TC>cd 434

F:\TC\434>command  blue green yellow
argc=4
argv[0]=F:\TC\434\COMMAND.EXE
argv[1]=blue
argv[2]=green
argv[3]=yellow
```