

## **Structures, Unions and Files**

### **STRUCTURES:**

**It is a derived data type. It is a collection of elements of heterogeneous data type(different).**

**Definition: It is heterogeneous collection of elements of different data elements under a single name.**

#### **Declaration:**

**Two types**

**We can declare structure in two ways:**

➤ **Tagged structure**

➤ **Typedef structure**

**Tagged structure:**

**It starts with a keyword 'struct'**

**Syntax:**

```
struct structurename  
{  
datatype var1;  
datatype var2;//structure mem  
.....;  
};
```

**Eg: struct student**

```
{  
char name[50]; int rollno;  
char gender; char grade; float m1,m2,m3; int age;  
};
```

## **Declaration of structure Variable:**

**The variables of the structure can be declared inside the main function or after structure declaration.**

### **Syntax:**

```
struct structurename
{
datatype Variable1;
datatype Variable2;
.....
.....
};
int main()
{
struct structurename Variables;//Variable
}
```

**Example:**

```
struct student
{
char name[50]; int age;
float per;
};
int main()
{
struct student s1;
}
```

**Accessing the members of structures:**

The members of structures can be accessed by the help of **dot operator**.

This **dot operator** is also called as member operator.

**Note:**

If the structure variables are pointers then the structure members can be accessed by arrow operator.(->) It is also called as indirect selection operator.

**Ex:**

s1.name; s1.age; s1.per;

**//write a c program to read and print the details of a single student using**

**structures.#include<stdio.h>**

**#include<stdlib.h>**

**struct student**

**{**

**char name[30];**

**int rollno;**

**int m1,m2,m3; float aVg;**

**};**

**int main()**

**{**

**struct student s1; printf("Enter details:\n");**

**scanf("%s%d%d%d",s1.name,&s1.rollno,&s1.m1,&s1.m2,&s1.m3);**

**s1.aVg=(s1.m1+s1.m2+s1.m3)/3;**

**printf("%s\n%d\n%f\n",s1.name,s1.rollno,s1.aVg);**

**}Output:**

**Bidya 104**

**95**

**96**

**98**

**95.00000**

**Bidya**

**104**

**95.00000**

**//write a c program to read and print the details of two students using structures.**

```
#include<stdio.h>
#include<stdlib.h>
struct student
{
char name[50]; int age;
int rollno;
}s1,s2;
int main()
{
printf("enter student1 details:");
scanf("%s%d%d",s1.name,&s1.age,&s1.rollno);
printf("enter student2 details:");
scanf("%s%d%d",s2.name,&s2.age,&s2.rollno);
printf("%s %d %d\n",s1.name,s1.age,s1.rollno);
printf("%s %d %d\n",s2.name,s2.age,s2.rollno);
}
```

**Output:**

**Enter student1 details:**

**Bidya 20**

**104**

**Pinky 20**

**110**

**Bidya 20 104**

**Pinky 20 110**

```
//write a c program to read the details of n students(name,rollno,three sub marks) and calculate avg and grade of each student#include<stdio.h>
#include<stdlib.h>
struct student
{
char name[50]; int rollno;
int m1,m2,m3; float avg; char grade;
};
struct student s[100]; int main()
{
int n,i;
printf("enter no of students:"); scanf("%d",&n);
for(i=1;i<n;i++)
{
scanf("%s%d%d%d%d",s[i].name,&s[i].rollno,&s[i].m1,&s[i].m2,&s[i].m3);
s[i].avg=(float)(s[i].m1+s[i].m2+s[i].m3)/3;
if(s[i].avg>=75)
s[i].grade='A';
else if(s[i].avg>=50)
s[i].grade='B';
else s[i].grade='C';
}
for(i=1;i<n;i++)
{
printf("name=%s\n",s[i].name);
printf("rollno=%d",s[i].rollno);
printf("average=%f\n",s[i].avg);
printf("grade=%c\n",s[i].grade);
}
}
```

## **Initialization of structure:**

The structure can be initialized in two ways:

➤ static

➤ Dynamic

i) static:

```
struct structurename
```

```
{
```

```
//member declaration
```

```
};
```

```
struct structurename Variable={value};
```

example:

```
struct emp
```

```
{
```

```
char name[50]; int id;
```

```
float salary;
```

```
};
```

```
struct emp e1;
```

```
int main()
```

```
{
```

```
emp e1={"raju","104",54000};
```

```
printf("%s%d%f",e1.name,e1.id,e1.salary);
```

```
}
```

## **2) type def structure:**

**We can also declare a structure by using typedef keyword. It is two ways different from tagged structure.**

**We use type def as keyword as the beginning of the structure. The structure is given at the end of closing bracket.**

### **Syntax:**

```
typedef struct  
{  
datatype var1;  
datatype var2;  
datatype var3;  
.....  
} structurename;
```

### **Eg:**

```
typedef struct  
{  
char name[80]; float m1,m2,m3; int age;  
char gender; char grade;  
}student;
```



## **Variable declaration in type def:**

**Syntax:**

**typedef struct structurename structure Variables;**

**example:**

**typedef struct emp e1,e2,e3;**

## **Structure Variable declaration:**

### **1.Tagged structure**

**Syntax:**

```
struct structurename  
{  
datatype var1;  
datatype var2;//structure mem  
.....;  
} structure Variables;  
(or)  
struct structurename Variables;
```

**Eg:**

```
struct student  
{  
char a[50]; int rollno; char gender; char grade;  
float m1,m2,m3; int age;  
} s1,s2,s3; Sizeof(s1)=50+4+1+1+12+4=72  
Sizeof(s2)=72 Sizeof(s3)=72
```

## **Typedef structure Syntax:**

```
typedef struct  
{  
datatype var1;  
datatype var2;  
datatype var3;  
.....  
} structurename; structure name Variables;
```

**Eg: typedef struct**

```
{  
char a[80]; float m1,m2,m3; int age;  
char gender; char grade;  
}student; student s1,s2,s3;
```

**àDuring structure declaration memory is not allocated to the structure**  
**-->the memory allocated to structure after structure declaration**

## **Accessing the members of the structure**

**Two operators are used to access the members of the structure**

**.(dot) operator(or) direct selection**

**->(arrow)operator (or) indirect selection operator**

### **Syntax:**

**structureVariable. sturcturemember**


**StructureVariable-> structuremember**

**Eg:**

```
#include<stdio.h>
struct student
{
char name[90];
int age;
int rollno;
};struct student s1,s2;
int main()
{
printf("enter student1 details");
scanf("%s%d%d",s1.name,&s1.age,&s1.rollno);
printf("enter student2 details");
scanf("%s%d%d",s2.name,&s2.age,&s2.rollno);
printf("%s %d %d\n",s1.name,s1.age,s1.rollno);
printf("%s %d %d\n",s2.name,s2.age,s2.rollno);
}
```

**Output:**

## **Important points on structures:**

- **structure is a collection of heterogeneous collection of elements under a single name.**
  - **It holds related information about entity structure name.**
  - **structure is a user defined datatype.**
  - **structure members are also called as attributes or data fields.**
  - **The main difference between array and structure is array holds the data of similar data types.**
  - **All the members of the structure are declared under a single name called structure name or entity.**
  - **The members of the structure accessed by using dot operator.**
  - **The name of the structure and structure member name should not be same.**
  - **memory is allocated for structures when we declare the structure Variables.**
- 

## Structure initialization using dot() operator:

### Example:

```
#include<stdio.h>
#include<stdlib.h>
```

```
struct EMP
```

```
{
```

```
char name[50];
```

```
int emid;
```

```
float salary;
```

```
float exp;
```

```
};
```

```
struct EMP e1,e2,e3;//global declaration
```

```
int main()
```

```
{
```

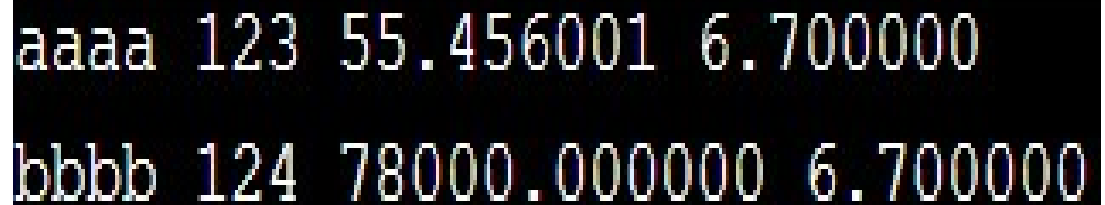
```
struct EMP e1={"aaaa",123,55.456,6.7};
```

```
struct EMP e2={"bbbb",124,78000,8.9};
```

```
printf("%s %d %f %f\n",e1.name,e1.emid,e1.salary,e1.exp);
```

```
printf("%s %d %f %f",e2.name,e2.emid,e2.salary,e1.exp);
```

```
}
```



```
aaaa 123 55.456001 6.700000
bbbb 124 78000.000000 6.700000
```

**//Write a c program to read the details of employee(emp no,name,basic salary,HRA,DA) of 'n' employees. Calculate the gross salary of each employee and print emp name,no,gross**

**salary.(codetantra)#include<stdio.h>**

**#include<stdlib.h>**

**struct employee**

**{**

**char name[100];**

**int no;**

**float BS,HRA,DA,GS;**

**};struct employee E[100];**

**int main()**

**{**

**int n,i;**

**printf("enter no of employees:");**

**scanf("%d",&n);**

**printf("enter employee details:\n");**

**for(i=1;i<=n;i++)**

**{**

**scanf("%s%d%f%f%f",E[i].name,&E[i].no,&E[i].BS,&E[i].HRA,&E[i].DA);**

**E[i].GS=E[i].BS+E[i].HRA+E[i].DA;**

**}**

**for(i=1;i<=n;i++)**

**{**

**printf("enter the details:\n");**

**printf("Name=%s\n",E[i].name); printf("Number=%d\n",E[i].no);**

**printf("Gross salary=%f\n",E[i].GS);**

**}}**

enter employee details:

spandana

hema

1

2

30000

20000

enter the details:

Name=spandana

Number=0

Gross salary=0.000000

enter the details:

Name=hema

Number=1

Gross salary=50002.000000

**//program on structures using -> arrow operator**

```
#include <stdio.h>
#include<stdlib.h>
struct student
{
char name[20];
int number;
int rank;
};
int main()
{
struct student s1;
struct student *s2;
s2=&s1;
scanf("%s",s2->name);
scanf("%d",&s2->number);
scanf("%d",&s2->rank);
printf("NAME:%s\n",s2->name);
printf("NUMBER:%d\n",s2->number);
printf("RANK:%d\n",s2->rank);
return 0;
}
```

**Output:**

**S 1620**

**1**

**NAME=spa NUMBER=1620 RANK=1**



**//program on structures using -> arrow operator**

```
#include <stdio.h>
#include<stdlib.h>
struct student
{
char name[20];
int number;
int rank;
};
int main()
{
struct student s1;
struct student *s2;
s2=&s1;
scanf("%s",s2->name);
scanf("%d",&s2->number);
scanf("%d",&s2->rank);
printf("NAME:%s\n",s2->name);
printf("NUMBER:%d\n",s2->number);
printf("RANK:%d\n",s2->rank);
return 0;
}
```

**Output:**

**S 1620**

**1**

**NAME=spa NUMBER=1620 RANK=1**

## **Complex structures:**

arrays, pointers, functions are called complex structures

### **1. nested structure**

structure within another structure is called nested.

**Declaration:**

#### **tagged nested structure**

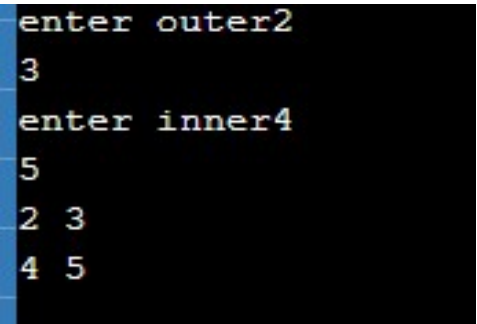
```
struct outer_structname
{
datatype outer_mem1;
datatype outer_mem2;
struct inner_structname
{
datatype inner_mem1;
datatype inner_mem2;
}inner_struct_variables;
}outer_structVariable;
```

#### **Accesssing:**

```
outer_structVariable.outer_mem1;
outer_structVariable.outer_mem2;
outer_structVariable.inner_structVariable.inner_mem1;
outer_structVariable.inner_structVariable.inner_mem2;
```

eg:

```
#include<stdio.h>
#include<stdlib.h>
struct ABC//outer structure
{
int x;
int y;
struct PQR//inner structure
{
int w;
int v;
}E;
}S;
int main()
{
printf("enter outer");
scanf("%d%d",&S.x,&S.y);
printf("enter inner");
scanf("%d%d",&S.E.w,&S.E.v);
printf("%d %d\n",S.x,S.y);
printf("%d %d\n",S.E.w,S.E.v);
}
```



```
enter outer2
3
enter inner4
5
2 3
4 5
```

typedef nested structure

typedef struct

{

datatype outer\_mem1;

datatype outer\_mem2;

typedef struct inner\_structname

{

datatype inner\_mem1;

datatype inner\_mem2;

} inner\_structname ;

} outer\_structname;

outer\_structname outer\_structVariable;

inner\_structname inner\_struct\_variables;

Accessing:

outer\_structVariable.outer\_mem1;

outer\_structVariable.outer\_mem2;

outer\_structVariable.inner\_structVariable.inner\_mem1;

outer\_structVariable.inner\_structVariable.inner\_mem2;

### Array within a structure(or)structure containg arrays

\*structure members are of arrays(example)

```
struct student
```

```
{
```

```
char name[50];//structure member is array
```

```
int marks[5];
```

```
char subjects[4][50];
```

```
float aVg;
```

```
};
```

```
struct student S={"AAA",{12,13,14,15},{Ch,Phy,Math,PPS},11};
```

```
S.marks[0],S.marks[1]
```

### Array of structure

structure variable is of array type

eg:

```
struct student
```

```
{
```

```
char name[50]; int marks[5]; char rollno[4]; float aVg;
```

```
}s[50];//structure variable is array
```

```
struct student s={{"AAA",{12,13,14,15},{Ch,Phy,Math,PPS},11},
```

```
{"BBB",{10,11,12,13},{Ch,Phy,Math,PPS},12,
```

```
{"CCC",{10,10,11,14},{Ch,Phy,Math,PPS},10}};
```

```
/**student details**/
```

```
/**employee details**/
```

## Structures and functions

call by value

call by reference

A structure can be passed to a function in three ways

1. Passing individual members of structure as parameter

Passing address individual members of structure as parameter

2. Passing whole structure as parameter

Passing address of structure as parameter

1) Passing individual members of structure as parameter

/\*multiply two fractional numbers\*/ #include<stdio.h>

#include<stdlib.h>

struct fraction

{

int num,den;

};struct fraction f1,f2,f3;

int mul(int ,int);//function declaration

int main()

{

printf("enter fration1");

scanf("%d%d",&f1.num,&f1.den);

printf("enter fration2");

scanf("%d%d",&f2.num,&f2.den);

f3.num=mul(f1.num,f2.num);//actual

f3.den=mul(f1.den,f2.den);//function call

printf("f3num=%d,f3den=%d",f3.num,f3.den);

}

int mul(int x,int y)//function definition

{

return (x\*y);

}

```
enter fration1
2
2
enter fration2
3
4
f3num=6, f3den=8
```

Passing address individual members of structure as parameter

```
#include<stdio.h>
#include<stdlib.h>
struct fraction
{
int num,den;
};
struct fraction f1,f2,f3;
int mul(int *x,int *y);//function declaration
int main()
{
printf("enter fration1 ");
scanf("%d%d",&f1.num,&f1.den);
printf("enter fration2");
scanf("%d%d",&f2.num,&f2.den);
f3.num=mul(&f1.num,&f2.num);//actual
f3.den=mul(&f1.den,&f2.den);//function call
printf("f3num=%d,f3den=%d",f3.num,f3.den);
}
int mul(int *x,int *y)//function definition
{
return (*x)*(*y);
}
```

**Write a C program to read real and imaginary parts of a complex number using structures and perform the following operations on complex numbers.**

**Add two complex numbers.**

**Multiply two complex numbers.**

**Subtract two complex numbers.**

```
#include <stdio.h>
#include<stdlib.h>
struct complex
{
int real, img;
};
struct complex a, b, c;

void add(struct complex a,struct complex b)
{
c.real = a.real + b.real;
c.img = a.img + b.img;
printf("Addition = %d + i %d\n",c.real,c.img);
}

void sub(struct complex a,struct complex b)
{
c.real = a.real - b.real; c.img = a.img - b.img;
printf("Subtraction = %d + i %d\n",c.real,c.img);
}
```



```
void mul(struct complex a,struct complex b)
{
c.real = a.real*b.real - a.img*b.img;
c.img = a.img*b.real + a.real*b.img;
printf("Multiplication = %d + i %d\n",c.real,c.img);
}
```

```
int main()
{
printf("Enter complex1 : ");
scanf("%d%d", &a.real,&a.img);
printf("Enter complex2 : ");
scanf("%d%d", &b.real,&b.img);
add(a,b);
sub(a,b);
mul(a,b);
return 0;
}
```

Write a C program to read time in hours, minutes, seconds using structures and perform the following operations on time.

Addition of two time periods.

Subtraction of two time periods.

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
struct time
```

```
{
```

```
int hr,min,sec;
```

```
};
```

```
struct time t1,t2,t3;
```

```
void add()
```

```
{
```

```
t3.min = t1.min+t2.min;
```

```
t3.sec = t1.sec+t2.sec;
```

```
t3.hr = t1.hr+t2.hr;
```

```
while(t3.sec>=60)
```

```
{
```

```
t3.min++;
```

```
t3.sec = t3.sec-60;
```

```
}
```

```
while(t3.min>=60)
```

```
{
```

```
t3.hr++;
```

```
t3.min = t3.min-60;
```

```
}
```

```
printf("Addition = %d:%d:%d\n",t3.hr,t3.min,t3.sec);
```

```
}
```

```
void sub()
{
if(t1.sec<t2.sec)
{
t1.min--;
t1.sec = t1.sec+60;
}
t3.sec = t1.sec-t2.sec;
if(t1.min<t2.min)
{
t1.hr--;
t1.min = t1.min+60;
}
t3.min = t1.min-t2.min;
t3.hr = t1.hr-t2.hr;
printf("Subtraction = %d:%d:%d\n",t3.hr,t3.min,t3.sec);
}
int main()
{
printf("Enter time1 : ");
scanf("%d%d%d",&t1.hr,&t1.min,&t1.sec);
printf("Enter time2 : ");
scanf("%d%d%d",&t2.hr,&t2.min,&t2.sec);
add();
sub();
return 0;
}
```

Passing whole structure as parameter

```
#include<stdio.h>
```

```
#include<stdlib.h>//header file for stryctures
```

```
struct fraction
```

```
{
```

```
int num,den;
```

```
};
```

```
struct fraction f1,f2,f3;
```

```
struct fraction mul(struct fraction f1,struct fraction f2);//function declaration
```

```
int main()
```

```
{
```

```
printf("enter fration1");
```

```
scanf("%d%d",&f1.num,&f1.den);//(1,2)
```

```
printf("enter fration2");
```

```
scanf("%d%d",&f2.num,&f2.den);//(3,4)
```

```
f3=mul(f1,f2);(1,2,3,4);//function call
```

```
printf("F3num=%d,F3den=%d",f3.num,f3.den);
```

```
}
```

```
struct fraction mul(struct fraction f1,struct fraction f2)//function definition
```

```
{
```

```
struct fraction res;
```

```
res.num=f1.num * f2.num;
```

```
res.den=f1.den * f2.den;
```

```
return res;
```

```
}
```

Passing address of whole structure as parameter

```
#include<stdio.h>
struct fraction
{
int num,den;
};
struct fraction *f1,*f2,*f3;
struct fraction mul(struct fraction *f1,struct fraction *f2);
int main()
{
printf("enter fration1");
scanf("%d%d",f1->num,f1->den);
printf("enter fration2");
scanf("%d%d",f2->num,f2->den);
*f3=mul(f1,f2);
printf("F3num=%d,F3den=%d",f3->num,f3->den);
}
struct fraction mul(struct fraction *f1,struct fraction *f2)
{
struct fraction *res;
res.num=f1->num * f2->num;
res.den=f1->den * f2->den;
return res;
}
```

## UNIONS:

it is also heterogeneous collection of data elements declaration:

```
union union_name
{
//union members;
datatype mem1;
  datatype mem2;
  datatype mem3;
  .....
};
union union_name Variable;//union Variable declaration
```

\*only difference b/w structure and union is in terms of memory allocation

\*memory associated with structure is sum of datatypes of all datamembers

\*memory associated with union is highest data member memory

\*members of union can accessed by using dot operator

eg:

```
#include<stdio.h>
#include<stdlib.h>
union student
{
char name[70];
int age;
char gender;
char grade;
float m1,m2,m3;
};union student s1;//70+4+1+1+12=88(structure)//union =70
int main(){
printf("enter name"); scanf("%s",s1.name);
printf("%s",s1.name);
printf("enter age"); scanf("%d",&s1.age);
printf("%d",s1.age);
printf("enter gender"); scanf("%c",&s1.gender);
printf("%c",s1.gender);
}
```

## self referential structure

A structure referring itself is called as self referential structure. it is mainly used in Data structure

### syntax:

```
struct structure_name  
{  
    struct structure_name *pointer  
};
```

### eg:

```
struct node  
{  
    int data;  
    struct node *link;  
};
```

### Representation of node single linked list

```
struct node  
{  
    int data;  
    struct node *next;  
};
```

### Representation of node double linked list

```
struct node  
{  
    int data;  
    struct node *prev; struct node *next;  
};
```



## Self Referential Structures

Self Referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.

In other words, structures pointing to the same type of structures are self- referential in nature.

Example:


```
filter_none brightness_4
```

```
struct node
{
int data1;
char data2;
struct node* link;
};
int main()
{
struct node ob;
return 0;
}
```

In the above example 'link' is a pointer to a structure of type 'node'. Hence, the structure 'node' is a self-referential structure with 'link' as the referencing pointer.

An important point to consider is that the pointer should be initialized properly before accessing, as by default it contains garbage Value.

# Self Referential Structures



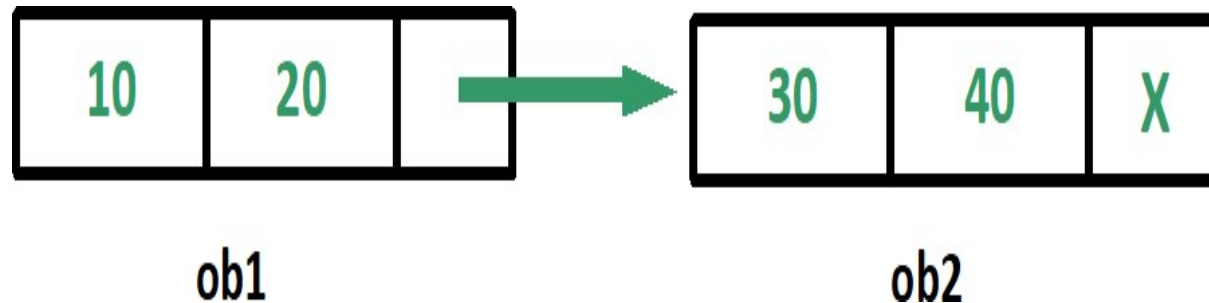
```
struct node {  
    int data1;  
    char data2;  
    struct node* link;  
};
```

Types of Self Referential Structures:

- **Self Referential Structure with Single Link**
- **Self Referential Structure with Multiple Links**

Self Referential Structure with Single Link:

These structures can have only one self-pointer as their member. The following example will show us how to connect the objects of a self-referential structure with the single link and access the corresponding data members. The connection formed is shown in the following figure.



```
filter_none edit play_arrow brightness_4  
#include <stdio.h>
```

```
struct node { int data1; char data2;  
struct node* link;  
};
```

```
int main()  
{
```

```
struct node ob1; // Node1
```

```
// Initialization ob1.link = NULL; ob1.data1 = 10;  
ob1.data2 = 20;
```

```
struct node ob2; // Node2
```

```
// Initialization ob2.link = NULL; ob2.data1 = 30;  
ob2.data2 = 40;
```

```
// Linking ob1 and ob2 ob1.link = &ob2;
```

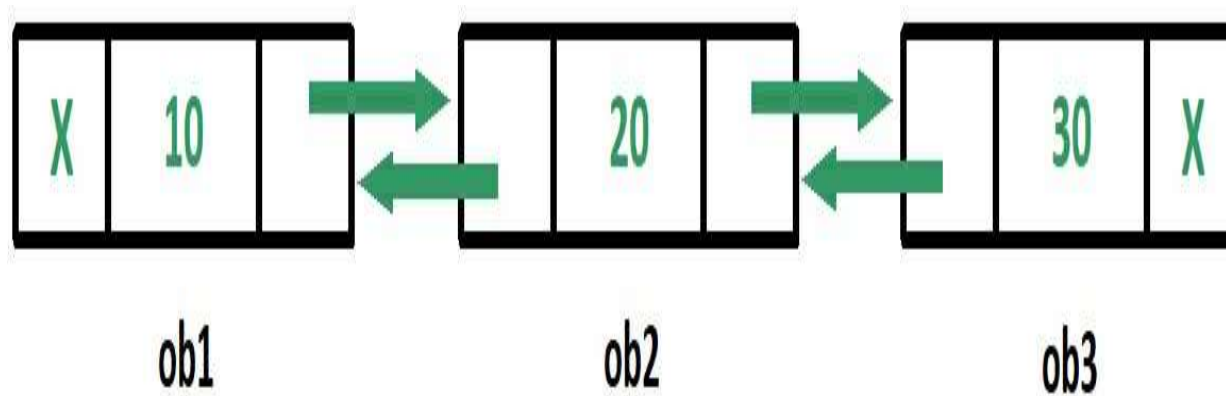
```
// Accessing data members of ob2 using ob1 printf("%d", ob1.link->data1);  
printf("\n%d", ob1.link->data2); return 0;  
}
```

Output:

Self Referential Structure with Multiple Links: Self referential structures with multiple links can have more than one self-pointers. Many complicated data structures can be easily constructed using these structures. Such structures can easily connect to more than one nodes at a time.

The following example shows one such structure with more than one links.

The connections made in the above example can be understood using the following figure.



```
#include <stdio.h>
struct node { int data;
struct node* prev_link;
struct node* next_link;
};
int main()
{
struct node ob1; // Node1
// Initialization
ob1.prev_link = NULL; o
b1.next_link = NULL;
ob1.data = 10;
struct node ob2; // Node2
// Initialization
ob2.prev_link = NULL;
ob2.next_link = NULL;
ob2.data = 20;
struct node ob3; // Node3
// Initialization
ob3.prev_link = NULL;
ob3.next_link = NULL;
ob3.data = 30;
// Forward links
ob1.next_link = &ob2;
ob2.next_link = &ob3;
// Backward links
ob2.prev_link = &ob1;
ob3.prev_link = &ob2;
```

```
// Accessing data of ob1, ob2 and ob3 by ob1
printf("%d\t", ob1.data);
printf("%d\t", ob1.next_link->data);
printf("%d\n", ob1.next_link->next_link->data);
// Accessing data of ob1, ob2 and ob3 by ob2
printf("%d\t", ob2.prev_link->data);
printf("%d\t", ob2.data);
printf("%d\n", ob2.next_link->data);
// Accessing data of ob1, ob2 and ob3 by ob3
printf("%d\t", ob3.prev_link->prev_link->data);
printf("%d\t", ob3.prev_link->data);
printf("%d", ob3.data);
return 0;
}
```

```
10 20 30
10 20 30
10 20 30
```

Output:

In the above example we can see that 'ob1', 'ob2' and 'ob3' are three objects of the self referential structure 'node'. And they are connected

using their links in such a way that any of them can easily access each other's data. This is the beauty of the self referential structures. The connections can be manipulated according to the requirements of the programmer.

typedef:

stands for type definition.

It is used for representing existing data types or rename the existing Variables.

Syntax:

```
typedef datatype identifier;
```

ex:

```
typedef int marks;
```

```
marks sub1,sub2,sub3;
```

```
typedef float a;
```

```
a s1;
```

```
#include<stdio.h> int main()
```

```
{
```

```
typedef int marks;
```

```
marks s1,s2,s3;
```

```
printf("enter marks:");
```

```
scanf("%d%d%d",&s1,&s2,&s3);
```

```
printf("the marks are:");
```

```
printf("%d %d %d",s1,s2,s3);
```

```
}
```



**Enumerated data type:**

**It is represented by “enum”**

**Syntax:**

```
enum typename
```

```
{
```

```
mem1,mem2,mem3;
```

```
};
```

```
#include <stdio.h>
```

```
enum week
```

```
{
```

```
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
```

```
};
```

```
int main()
```

```
{ // creating today variable of enum week type
```

```
enum week today;
```

```
today = Wednesday;
```

```
printf("Day %d",today+1);
```

```
return 0;
```

```
}
```

**Output**

**Day 4**

*Write a C program to store name, roll number, year and marks of three subjects of n students and print the student the name, roll number, average, and grade based on average marks of the student using structures.*

```
#include<stdio.h>
struct student {
int rollno;
char name[30]; int year;
int m1,m2,m3;
};struct student s[20];
int main()
{
int n,i; float avg; char grade;
printf("Enter no of students(<=3) : ");
scanf("%d",&n);
printf("Enter rollno,name,year,m1,m2,m3 : ");
for(i=0; i<n; i++)
scanf("%d%s%d%d%d%d",&s[i].rollno,s[i].name,&s[i].year,&s[i].m1,&s[i].m2, &s[i].m3);
for(i=0; i<n; i++)
{
avg = (s[i].m1+s[i].m2+s[i].m3)/3.0; if(avg>=75)
grade = 'A';
else if(avg>=50) grade = 'B';
else
grade = 'C';
printf("Rollno = %d\nName = %s\nAverage = %f\nGrade = %c\n",s[i].rollno,s[i].name,avg,grade);
}
```

Write a C program to read real and imaginary parts of a complex number using structures and perform the following operations on complex numbers.

Add two complex numbers.

Multiply two complex numbers.

Subtract two complex numbers.

```
#include <stdio.h>

struct complex
{
int real, img;
};

struct complex a, b, c;

void add(struct complex a, struct complex b)
{
c.real = a.real + b.real; c.img = a.img + b.img;
printf("Addition = %d + i %d\n", c.real, c.img);
}

void sub(struct complex a, struct complex b)
{
c.real = a.real - b.real; c.img = a.img - b.img;
printf("Subtraction = %d + i %d\n", c.real, c.img);
}

void mul(struct complex a, struct complex b)
{
c.real = a.real*b.real - a.img*b.img; c.img = a.img*b.real + a.real*b.img;
printf("Multiplication = %d + i %d\n", c.real, c.img);
}

int main()
{
printf("Enter complex1 : ");
scanf("%d%d", &a.real, &a.img); printf("Enter complex2 : ");
scanf("%d%d", &b.real, &b.img); add(a, b);
sub(a, b);
mul(a, b);
return 0;
}
```



## Structures in C

In C programming language, a structure is a collection of elements of the different data type. The structure is used to create user-defined data type in the C programming language. As the structure used to create a user-defined data type, the structure is also said to be “user-defined data type in C”.

In other words, a structure is a collection of non-homogeneous elements. Using structure we can define new data types called user-defined data types that holds multiple values of the different data type.

The formal definition of structure is as follows...

**Structure is a collection of different type of elements under a single name that acts as user defined data type in C.**

Generally, structures are used to define a record in the c programming language. Structures allow us to combine elements of a different data type into a group. The elements that are defined in a structure are called members of structure.

## How to create structure?

To create structure in c, we use the keyword called "struct".

We use the following syntax to create structures in c programming language.

```
struct <structure_name>
{
    data_type member1;
    data_type member2, member3;
    .
    .
};
```

Following is the example of creating a structure called Student which is used to hold student record.

### Creating structure in C

```
struct Student
{
    char stud_name[30];
    int roll_number;
    float percentage;
};
```

### Important Points to be Remembered

Every structure must terminated with semicolon symbol (;).

"struct" is a keyword, it must be used in lowercase letters only.

## Creating and Using structure variables

In a c programming language, there are two ways to create structure variables. We can create structure variable while defining the structure and we can also create after terminating structure using struct keyword.

To access members of a structure using structure variable, we use dot (.) operator.

Consider the following example code...

### Creating and Using structure variables in C

```
struct Student
```

```
{  
    char stud_name[30];  
    int roll_number;  
    float percentage;  
} stud_1 ; // while defining structure  
  
void main()  
{ struct Student stud_2; // using struct keyword  
    printf("Enter details of stud_1 : \n");  
    printf("Name : ");  
    scanf("%s", stud_1.stud_name);  
    printf("Roll Number : ");  
    scanf("%d", &stud_1.roll_number);  
    printf("Percentage : ");  
    scanf("%f", &stud_1.percentage);  
    printf("***** Student 1 Details *****\n");  
    printf("Name of the Student : %s\n", stud_1.stud_name);  
    printf("Roll Number of the Student : %i\n", stud_1.roll_number); printf("Percentage of the Student : %f\n",  
stud_1.percentage);  
}
```

In the above example program, the structure variable "**stud\_1**" is created while defining the structure and the variable "**stud\_2**" is created using struct keyword. Whenever we access the members of a structure we use the dot (.) operator.

## Memory allocation of Structure

When the structures are used in the c programming language, the memory does not allocate on defining a structure. The memory is allocated when we create the variable of a particular structure. As long as the variable of a structure is created no memory is allocated. The size of memory allocated is equal to the sum of memory required by individual members of that structure.

In the above example program, the variables stud\_1 and stud\_2 are allocated with 36 bytes of memory each.

```
struct Student{  
    char stud_name[30]; ————— 30 bytes  
    int roll_number; ————— 02 bytes  
    float percentage; ————— 04 bytes  
};  
                                sum = 36 bytes
```

Here the variable of **Student** structure is allocated with 36 bytes of memory.

### Important Points to be Remembered

All the members of a structure can be used simultaneously.

Until variable of a structure is created no memory is allocated.

The memory required by a structure variable is sum of the memory required by individual members of that structure.



## Unions in C

In C programming language, the union is a collection of elements of the different data type. The union is used to create user-defined data type in the C programming language. As the union used to create a user-defined data type, the union is also said to be “user-defined data type in C”.

In other words, the union is a collection of non-homogeneous elements. Using union we can define new data types called user-defined data types that holds multiple values of the different data type.

The formal definition of a union is as follows...

**Union is a collection of different type of elements under a single name that acts as user defined data type in C.**

Generally, unions are used to define a record in the c programming language. Unions allow us to combine elements of a different data type into a group. The elements that are defined in a union are called members of union.

## How to create union?

To create union in c, we use the keyword called "union".

We use the following syntax to create unions in c programming language.

**union <structure\_name>**

**{**

**data\_type member1;**

**data\_type member2, member3;**

**•**

**•**

**} ;**

Following is the example of creating a union called Student which is used to hold student record.

### Creating union in C

```
union Student { char stud_name[30]; int roll_number; float percentage; } ;
```

## Important Points to be Remembered

Every union must be terminated with semicolon symbol (;).

"union" is a keyword, it must be used in lowercase letters only.

## Creating and Using union variables

In a c programming language, there are two ways to create union variables. We can create union variable while the union is defined and we can also create after terminating union using union keyword.

TO access members of a union using union variable, we use dot (.) operator.

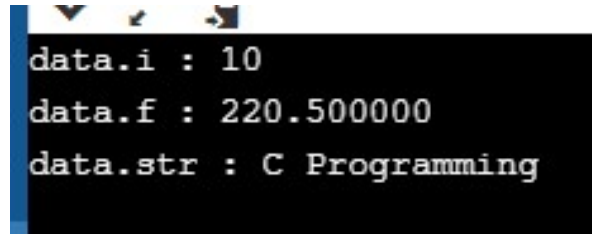
Consider the following example code...

### Creating and Using union variables in C

```
union Student {  
char stud_name[30];  
int roll_number;  
float percentage;  
} stud_1 ; // while defining union  
  
void main()  
{  
union Student stud_2; // using union keyword  
printf("Enter details of stud_1 : \n");  
printf("Name : ");  
scanf("%s", stud_1.stud_name);  
printf("Roll Number : ");  
scanf("%d", &stud_1.roll_number);  
printf("Percentage : ");  
scanf("%f", &stud_1.percentage);  
printf("***** Student 1 Details *****\n");  
printf("Name of the Student : %s\n", stud_1.stud_name);  
printf("Roll Number of the Student : %i\n", stud_1.roll_number);  
printf("Percentage of the Student : %f\n", stud_1.percentage);  
}
```

In the above example program, the union variable "**stud\_1**" is created while defining the union and the variable "**stud\_2**" is created using union keyword. Whenever we access the members of a union we use the dot (.) operator.

```
#include <stdio.h>
#include <string.h>
union Data
{
    int i;
    float f;
    char str[20];
};
int main( )
{
    union Data data;
    data.i = 10
; printf( "data.i : %d\n", data.i);
    data.f = 220.5;
    printf( "data.f : %f\n", data.f);
    strcpy( data.str, "C Programming");
    printf( "data.str : %s\n", data.str);
    return 0;
}
```

A screenshot of a terminal window with a black background and white text. It displays the output of the C program: 'data.i : 10', 'data.f : 220.500000', and 'data.str : C Programming'.

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

When the above code is compiled and executed, it produces the following result –

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

## Memory allocation of Union

When the unions are used in the c programming language, the memory does not allocate on defining union. The memory is allocated when we create the variable of a particular union. As long as the variable of a union is created no memory is allocated. The size of memory allocated is equal to the maximum memory required by an individual member among all members of that union.

In the above example program, the variables stud\_1 and stud\_2 are allocated with 30 bytes of memory each.

```
union Student{  
    char stud_name[30]; ————— 30 bytes  
    int roll_number; ————— 02 bytes  
    float percentage; ————— 04 bytes  
};  
                                max = 30 bytes
```

Here the variable of **Student** union is allocated with 30 bytes of memory and it is shared by all the members of that union.

## **Files in C**

Generally, a file is used to store user data in a computer. In other words, computer stores the data using files.

we can define a file as follows...

**File is a collection of data that stored on secondary memory like harddisk of a computer.**

C programming language supports two types of files and they are as follows...

- **Text Files (or) ASCII Files**
- **Binary Files**

***Text File (or) ASCII File*** - The file that contains ASCII codes of data like digits, alphabets and symbols is called text file (or) ASCII file.

***Binary File*** - The file that contains data in the form of bytes (0's and 1's) is called as binary file. Generally, the binary files are compiled version of text files.

## **File Operations in C**

The following are the operations performed on files in c programming language...

- **Creating (or) Opening a file**
- **Reading data from a file**
- **Writing data into a file**
- **Closing a file**

All the above operations are performed using file handling functions available in C. We discuss file handling functions in the next topic.

## **File Handling Functions in C**

File is a collection of data that stored on secondary memory like hard disk of a computer.

The following are the operations performed on files in the c programming language...

**Creating (or) Opening a file**

**Reading data from a file**

**Writing data into a file**

**Closing a file**

All the above operations are performed using file handling functions available in C.





## Creating (or) Opening a file

To create a new file or open an existing file, we need to create a file pointer of FILE type.

Following is the sample code for creating file pointer.

```
File *f_ptr ;
```

We use the pre-defined method **fopen()** to create a new file or to open an existing file. There are different modes in which a file can be opened.

Consider the following code...

```
File *f_ptr ; *f_ptr = fopen("abc.txt", "w") ;
```

The above example code creates a new file called **abc.txt** if it does not exist otherwise it is opened in writing mode.

In C programming language, there different modes are available to open a file and they are shown in the following table.

S. No.	Mode	Description
1	r	Opens a text file in <b>reading</b> mode.
2	w	Opens a text file in <b>wirting</b> mode.
3	a	Opens a text file in <b>append</b> mode.
4	r+	Opens a text file in both <b>reading and writing</b> mode.
5	w+	Opens a text file in both <b>reading and writing</b> mode. It set the cursor position to the begining of the file if it exists.
6	a+	Opens a text file in both <b>reading and writing</b> mode. The reading operation is performed from begining and writing operation is performed at the end of the file.



**Note** - The above modes are used with text files only. If we want to work with binary files we use

**rb, wb, ab, rb+, wb+ and ab+.**

## Reading from a file

The reading from a file operation is performed using the following pre-defined file handling methods.

**getc()**

**getw()**

**fscanf()**

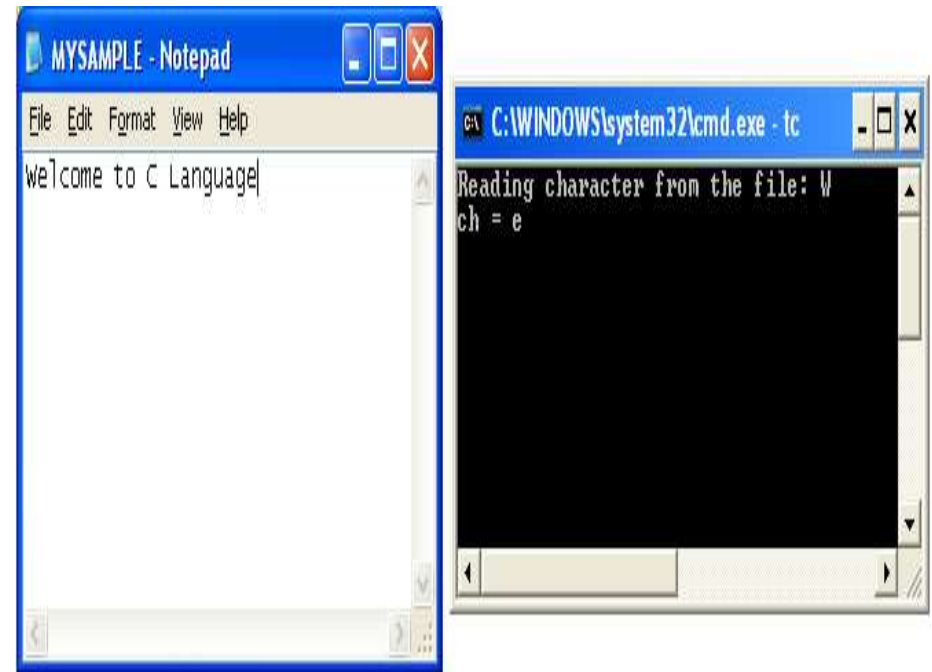
**fgets()**

**fread()**

**getc( *\*file\_pointer* )** - This function is used to read a character from specified file which is opened in reading mode. It reads from the current position of the cursor. After reading the character the cursor will be at next character.

### Example Program to illustrate getc() in C.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    char ch;
    clrscr();
    fp = fopen("MySample.txt","r");
    printf("Reading character from the file: %c\n",getc(fp));
    ch = getc(fp);
    printf("ch = %c", ch);
    fclose(fp);
    getch();
    return 0;
}
```

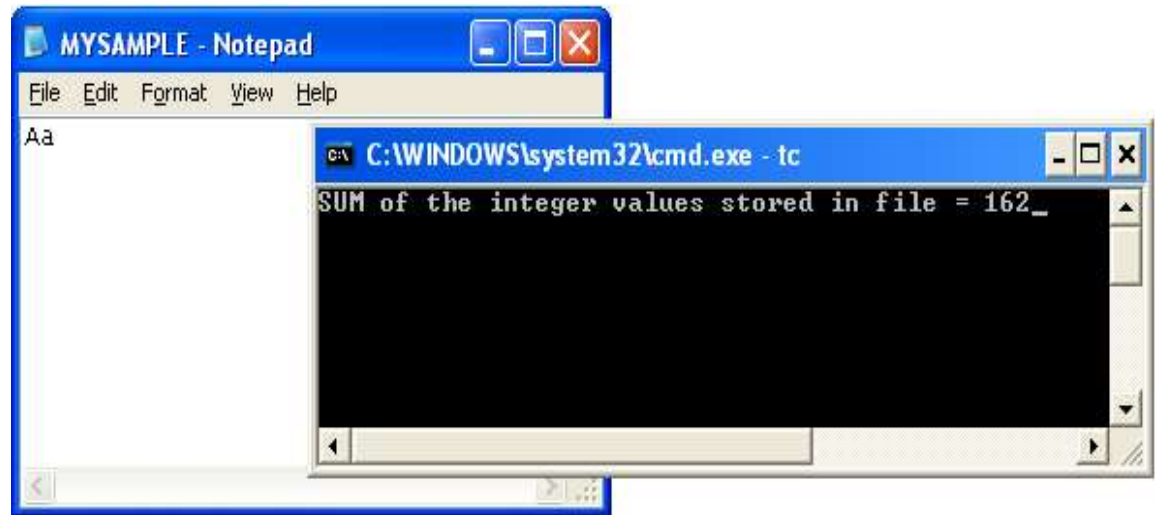


***getw( \*file\_pointer )*** - This function is used to read an integer value form the specified file which is opened in reading mode. If the data in file is set of characters then it reads ASCII values of those characters.

### Example Program to illustrate getw() in C.

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
int i,j;
clrscr();
fp = fopen("MySample.txt","w");
putw(65,fp); // inserts A
putw(97,fp); // inserts a
fclose(fp); fp = fopen("MySample.txt","r");
i = getw(fp); // reads 65 - ASCII value of A
j = getw(fp); // reads 97 - ASCII value of a
printf("SUM of the integer values stored in file = %d", i+j); // 65 + 97 = 162
fclose(fp);
getch();
return 0;
}
```

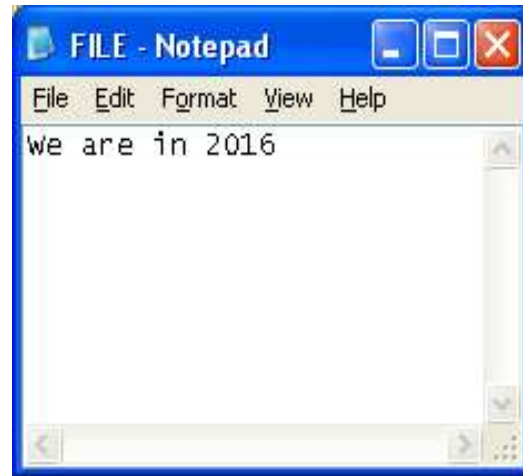
**Output**



***fscanf( \*file\_pointer, typeSpecifier, &variableName )*** - This function is used to read multiple datatype values from specified file which is opened in reading mode.

### Example Program to illustrate fscanf() in C.

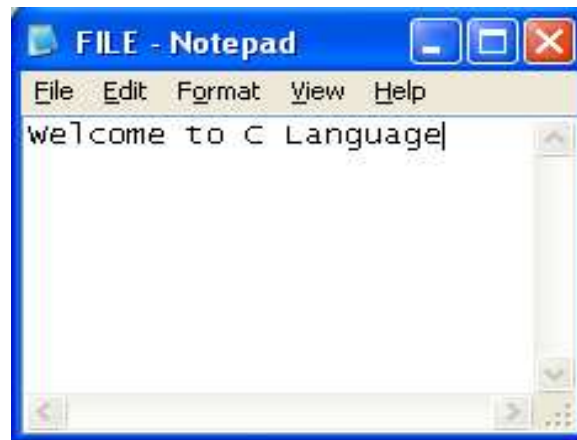
```
#include<stdio.h>
#include<conio.h>
int main()
{
    char str1[10], str2[10], str3[10];
    int year;
    FILE * fp;
    clrscr();
    fp = fopen ("file.txt", "w+");
    fputs("We are in 2016", fp);
    rewind(fp); // moves the cursor to beginning of the file
    fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
    printf("Read String1 - %s\n", str1 );
    printf("Read String2 - %s\n", str2 );
    printf("Read String3 - %s\n", str3 );
    printf("Read Integer - %d", year );
    fclose(fp);
    getch();
    return 0;
}
```



***fgets( variableName, numberOfCharacters, \*file\_pointer )*** - This method is used for reading a set of characters from a file which is opened in reading mode starting from the current cursor position. The fgets() function reading terminates with reading NULL character.

### **Example Program to illustrate fgets() in C.**

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char *str;
clrscr();
fp = fopen ("file.txt", "r");
fgets(str,6,fp);
printf("str = %s", str);
fclose(fp);
getch();
return 0;
}
```



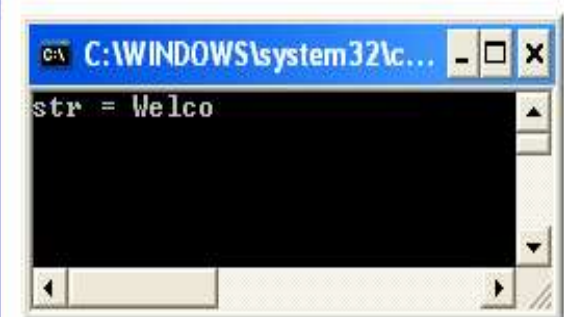
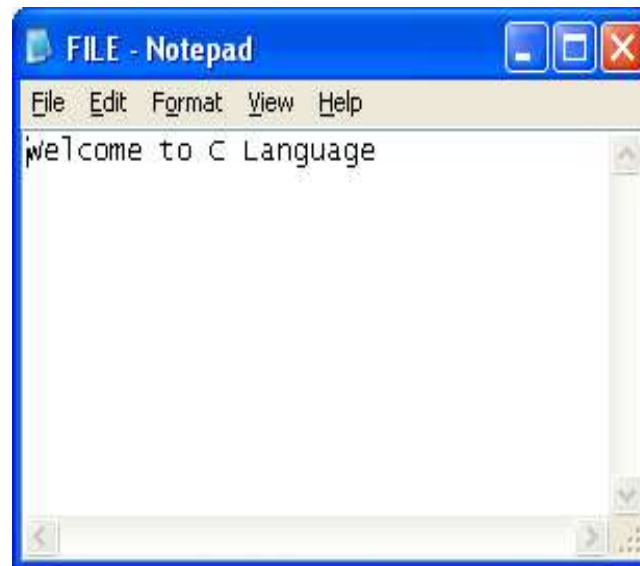


*fread( source, sizeofReadingElement, numberOfCharacters, FILE \*pointer ) –*

This function is used to read specific number of sequence of characters from the specified file which is opened in reading mode.

Example Program to illustrate fgets() in C.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    char *str;
    clrscr();
    fp = fopen ("file.txt", "r");
    fread(str,sizeof(char),5,fp);
    str[strlen(str)+1] = 0;
    printf("str = %s", str);
    fclose(fp);
    getch();
    return 0;
}
```



## Writing into a file

The writing into a file operation is performed using the following pre-defined file handling methods.

**putc()**

**putw()**

**fprintf()**

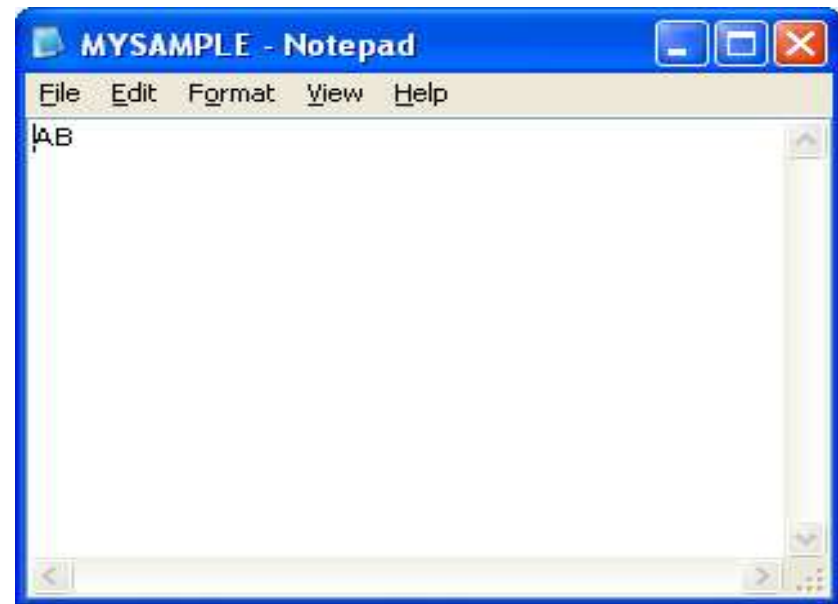
**fputs()**

**fwrite()**

***putc(char, \*file\_pointer)*** - This function is used to write/insert a character to the specified file when the file is opened in writing mode.

### Example Program to illustrate putc() in C.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    char ch;
    clrscr();
    fp = fopen("C:/TC/EXAMPLES/MySample.txt","w");
    putc('A',fp);
    ch = 'B';
    putc(ch,fp);
    fclose(fp);
    getch();
    return 0;
}
```

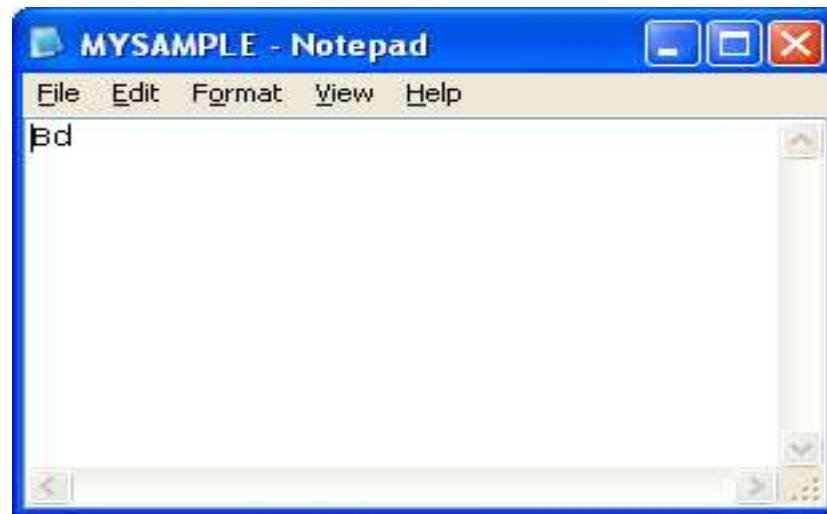




***putw( int, \*file\_pointer )*** - This function is used to writes/inserts an integer value to the specified file when the file is opened in writing mode.

**Example Program to illustrate putw() in C.**

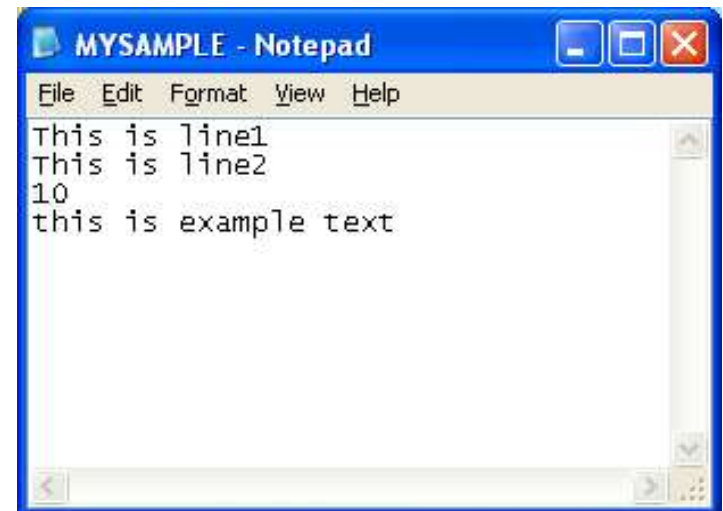
```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    int i;
    clrscr();
    fp = fopen("MySample.txt","w");
    putw(66,fp);
    i = 100;
    putw(i,fp);
    fclose(fp);
    getch();
    return 0;
}
```



***fprintf( \*file\_pointer, "text" )*** - This function is used to writes/inserts multiple lines of text with mixed data types (char, int, float, double) into specified file which is opened in writing mode.

### **Example Program to illustrate "fprintf()" in C.**

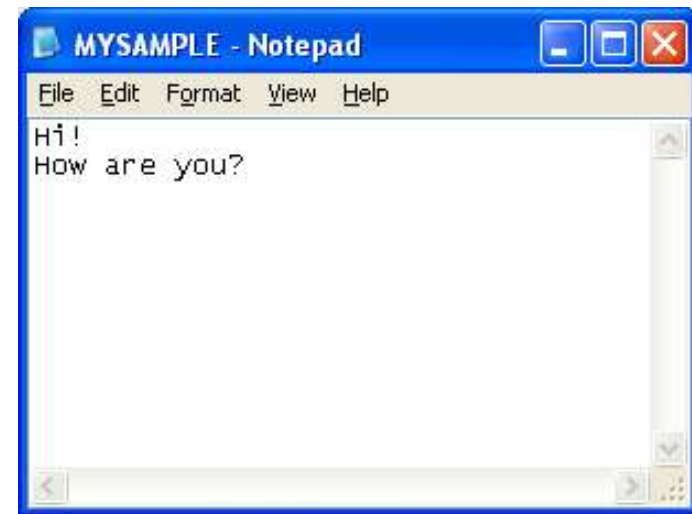
```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    char *text = "\nthis is example text"; int i = 10;
    clrscr();
    fp = fopen("MySample.txt","w");
    fprintf(fp,"This is line1\nThis is line2\n%d", i);
    fprintf(fp,text);
    fclose(fp);
    getch();
    return 0;
}
```



***fputs( "string", \*file\_pointer )*** - This method is used to insert string data into specified file which is opened in writing mode.

### **Example Program to illustrate fputs() in C.**

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char *text = "\nthis is example text";
clrscr();
fp = fopen("MySample.txt","w");
fputs("Hi!\nHow are you?",fp);
fclose(fp);
getch();
return 0;
}
```

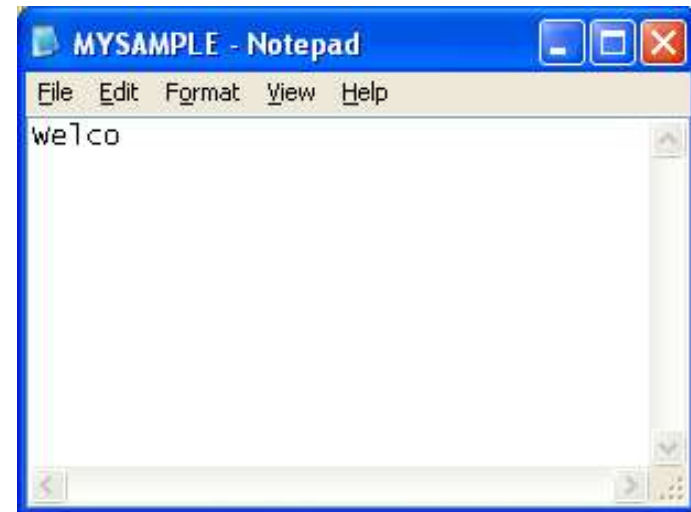


***fwrite( "StringData", sizeof(char), numberOfCharacters, FILE \*pointer ) -***

This function is used to insert specified number of characters into a binary file which is opened in writing mode.

**Example Program to illustrate fwrite() in C.**

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    char *text = "Welcome to C Language";
    clrscr();
    fp = fopen("MySample.txt","wb");
    fwrite(text,sizeof(char),5,fp);
    fclose(fp);
    getch();
    return 0;
}
```



## **Closing a file**

Closing a file is performed using a pre-defined method `fclose()`.

`fclose( *f_ptr )`

The method `fclose()` returns '0' on success of file close otherwise it returns EOF (End Of File).

## **Cursor Positioning Functions in Files**

C programming language provides various pre-defined methods to set the cursor position in files.

The following are the methods available in c, to position cursor in a file.

**`ftell()`**

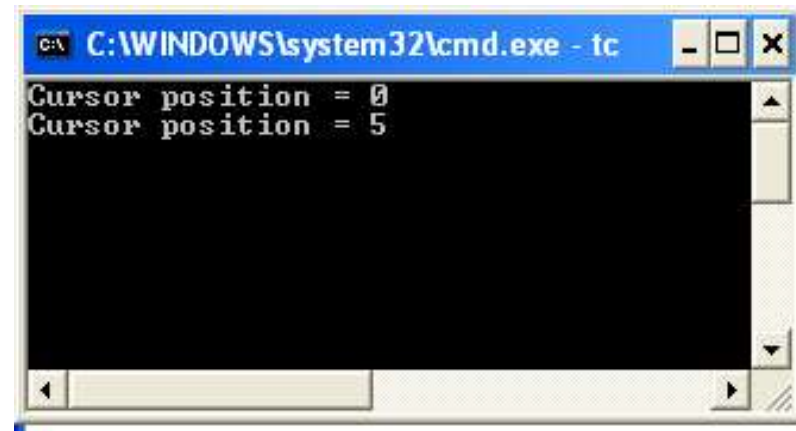
**`rewind()`**

**`fseek()`**

***ftell( \*file\_pointer )*** - This function returns the current position of the cursor in the file.

### Example Program to illustrate ftell() in C.

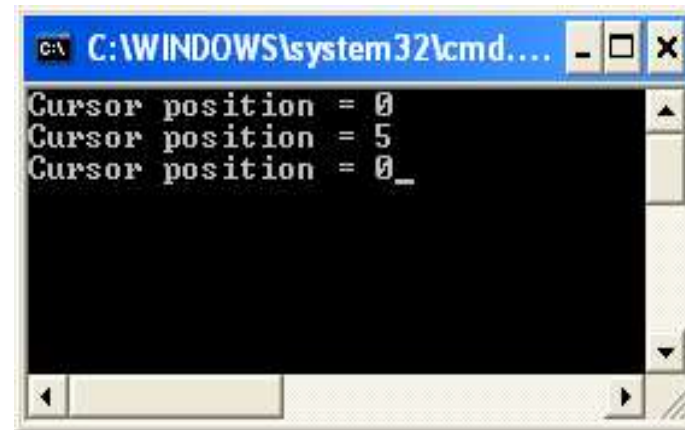
```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    int position;
    clrscr();
    fp = fopen ("file.txt", "r");
    position = ftell(fp);
    printf("Cursor position = %d\n",position);
    fseek(fp,5,0);
    position = ftell(fp);
    printf("Cursor position = %d", position);
    fclose(fp);
    getch();
    return 0;
}
```



***rewind( \*file\_pointer )*** - This function is used reset the cursor position to the beginning of the file.

### Example Program to illustrate rewind() in C.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *fp;
    int position;
    clrscr();
    fp = fopen ("file.txt", "r");
    position = ftell(fp);
    printf("Cursor position = %d\n",position);
    fseek(fp,5,0);
    position = ftell(fp);
    printf("Cursor position = %d\n", position);
    rewind(fp);
    position = ftell(fp);
    printf("Cursor position = %d", position);
    fclose(fp);
    getch();
    return 0;
}
```

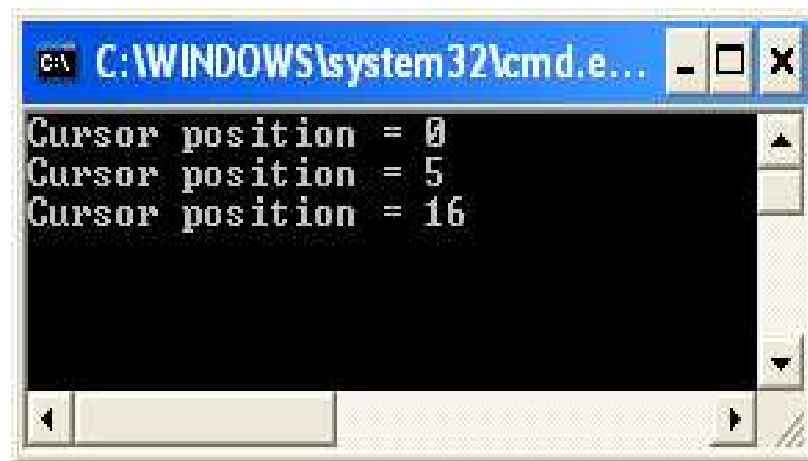
A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd...". The window contains three lines of text: "Cursor position = 0", "Cursor position = 5", and "Cursor position = 0\_". The cursor is positioned at the end of the third line. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

***fseek( \*file\_pointer, numberOfCharacters, fromPosition )*** - This function is used to set the cursor position to the specific position.

Using this function we can set the cursor position from three different position they are as follows.  
from beginning of the file (indicated with 0)  
from current cursor position (indicated with 1)  
from ending of the file (indicated with 2)

### **Example Program to illustrate fseek() in C.**

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
int position;
clrscr();
fp = fopen ("file.txt", "r");
position = ftell(fp);
printf("Cursor position = %d\n",position);
fseek(fp,5,0);
position = ftell(fp);
printf("Cursor position = %d\n", position);
fseek(fp, -5, 2);
position = ftell(fp);
printf("Cursor position = %d", position);
fclose(fp);
getch();
return 0;
}
```



```
C:\WINDOWS\system32\cmd.e...
Cursor position = 0
Cursor position = 5
Cursor position = 16
```



