PROGRAMMING FOR PROBLEM SOLVING USING C

(201ES2T08)

A.Lakshmanarao
Associate Professor,H&BS-I Dept.
Aditya Engineering College(A)
Mail : a.lakshmanarao@aec.edu.in
Cell: +91-9951060528

UNIT-V:

Structures, Unions, Bit Fields: Introduction, Nested Structures, Arrays of Structures, Structures and Functions, Self-Referential Structures, Unions, Enumerated Data Type–enum variables, Using Typedef keyword, Bit Fields.

Data Files: Introduction to Files, Using Files in C, Reading from Text Files, Writing to Text Files, Random File Access.

C Structure

- Structure is a user-defined datatype in C language.

- **Structure is a group of variables of different data types represented by a single name.**

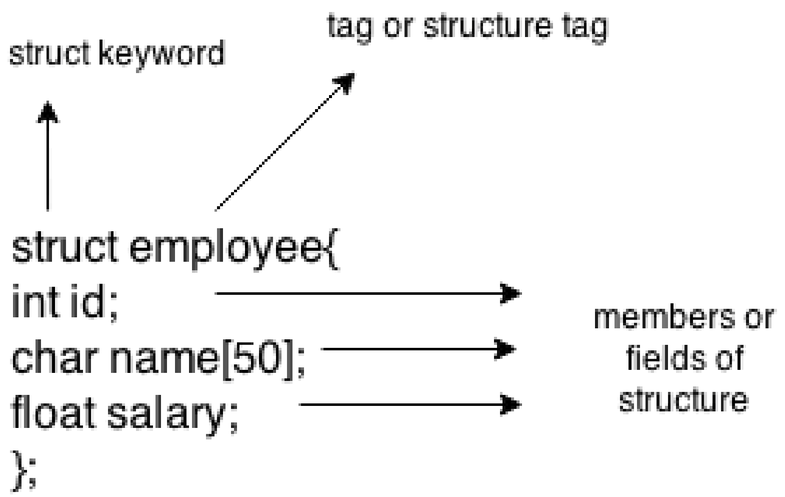
- The **,struct** keyword is used to define the structure.

SYNTAX-structure declaration

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Example1:

```
struct employee
{
 int id;
   char name[50];
   float salary;
};
```



```
struct employee{
int id;
char name[50];
float salary;
};
```

structure declaration

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

We can write structure in two places 1)above main   2)In main

```
struct student
{
 int rollno;
    char name[20];
    float CGPA;
};
int main()
{.....
...
...return 0;}
```

```
int main()
{
struct student
{
 int rollno;
    char name[20];
    float CGPA;
};
.....
...
...return 0;}
```

Accessing members of the structure

- Structure members(data) cannot be accessed directly.

step1)First, we need to create a structure variable with the following syntax.

syntax for creating structure variable:

- **strucut structure_name variable;**

step2)Next,use dot operator to access structure member.

syntax for accessing structure member:

strucuturevariable.structuremember;

Creating structure variable-2 ways

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1;
int main()
{
..
..
.
..}
```

```
struct employee
{   int id;
    char name[50];
    float salary;
};
int main()
{
struct employee e1;
..
..
...}
```

Initialization of structure members

```
struct student
{
 int rollno=20;  //gives error
   char name[20]="ABCD";   //gives error
   float CGPA=9.5;   //gives error
};
int main()
{.....
...
...return 0;}
```

cannot initialize members here

Reason:

when a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.

Initialization can be done with structure variable only.

structure-example

```
#include<stdio.h>
#include <string.h>
struct student
{   int rollno;
    char name[50];
    float cgpa;
}s1;  //declaring s1 variable for structure
```

```
int main( )
{
  s1.rollno=39;
   strcpy(s1.name, "ABCD");//copying string into char array
   s1.cgpa=9.8;
printf( "student1-id : %d\n", s1.rollno);
     printf( "student1-cgpa : %f\n", s1.cgpa);
   printf( "student1-1 name : %s\n", s1.name);
return 0;
}
```

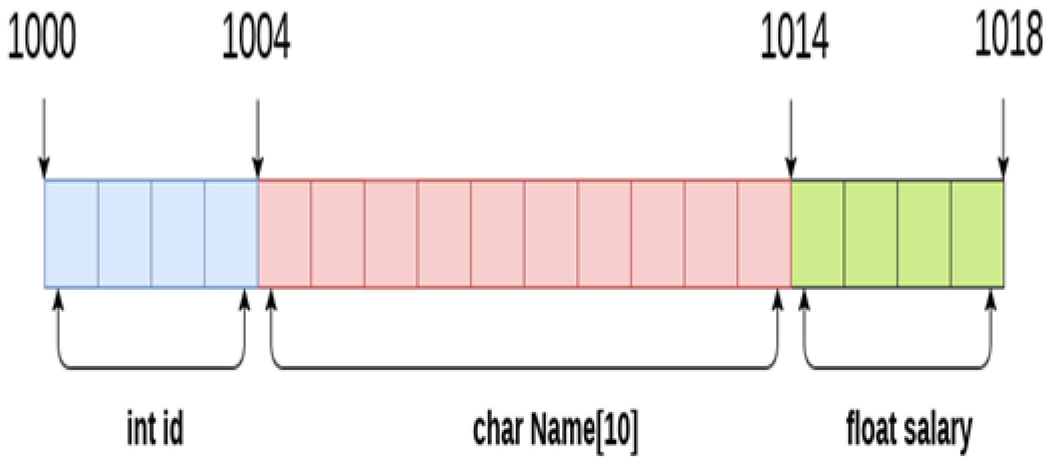
structure-example

```
#include<stdio.h>
#include <string.h>
struct book
{   int pages;
    char *name;
    float price;
};
```

```
int main( )
{     struct book b1;  //declaring s1 variable for structure
      b1.pages=250;
   b1.name="C programming";
   b1.price=345.50;
   //printing first employee information
   printf( "book-pages : %d\n", b1.pages);
     printf( "book-price : %f\n", b1.price);
   printf( "book name : %s\n", b1.name);
return 0;
}
```

Memory allocation of the structure

```
struct Employee
{
 int id;
    char Name[10];
    float salary;
};
```



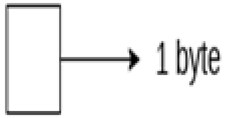
sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;

sizeof (int) = 4 byte

sizeof (char) = 1 byte

sizeof (float) = 4 byte



Nested Structures

- The structure can be nested in the following ways.

1.By separate structure

2.By Embedded structure

Nested Structures-By separate structure

```
#include<stdio.h>
struct DOB
{
   int dd;
   int mm;
   int yyyy;
};
struct Student
{
   int rollno;
   char *name;
   struct DOB d;
}s1;
```

```
int main() {
    s1.rollno=10;
    s1.name="ABCD";
    s1.d.dd=6;
    s1.d.mm=8;
    s1.d.yyyy=1996;
    printf("roll no=%d",s1.rollno);
    printf("\nname is %s",s1.name);
    printf("\n DOB is %d %d %d",s1.d.dd,s1.d.mm,s1.d.yyyy);
    return 0; }
```

Nested Structures-By Embedded structure

```
#include<stdio.h>
struct Student
{
  int rollno;
  char *name;
  struct DOB
   {
    int dd;
    int mm;
    int yyyy;
   }d;
}s1;
```

```
 int main()
{
s1.rollno=10;
  s1.name="ABCD";
  s1.d.dd=6;
  s1.d.mm=8;
  s1.d.yyyy=1996;
 printf("roll no=%d",s1.rollno);
  printf("\nname is %s",s1.name);
  printf("\n DOB is %d %d %d",s1.d.dd,s1.d.mm,s1.d.yyyy);

  return 0;
}
```

Nested Structure-summary

1. By separate structure

```
#include<stdio.h>
struct DOB
{
   int dd;
   int mm;
   int yyyy;
};
struct Student
{
   int rollno;
   char *name;
   struct DOB d;
}s1;
```

2. By Embedded structure

```
#include<stdio.h>
struct Student
{
   int rollno;
   char *name;
   struct DOB
    {
      int dd;
      int mm;
      int yyyy;
    }d;
}s1;
```

```
int main() {
    s1.rollno=10;
    s1.name="ABCD";
    s1.d.dd=6;
    s1.d.mm=8;
    s1.d.yyyy=1996;
printf("roll no=%d",s1.rollno);
    printf("\nname is %s",s1.name);
    printf("\n DOB is %d %d %d",s1.d.dd,s1.d.mm,s1.d.yyyy);
    return 0; }
```

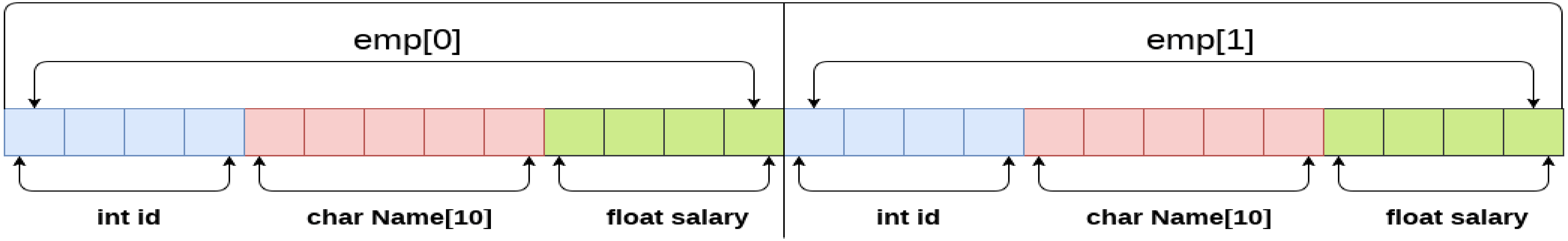
Array of Structures

- An array of structres in C can be defined as the collection of multiple structures variables where each variable contains information about different entities.
- The array of structures in C are used to store information about multiple entities of different data types.
- The array of structures is also known as the collection of structures.

Array of Structures-Example

```
struct employee
{
     int id;
     char name[5];
     float salary;
};
struct employee emp[2];
```

Array of structures



Array of Structures-Example

```
#include<stdio.h>
struct student{
int rollno;
float marks;
}s[5];
int main()
{
int i;

printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&s[i].rollno);
printf("\nEnter Marks:");
scanf("%f",&s[i].marks);
}
printf("Entered Details of 5 studentsare:\n");
for(i=0;i<5;i++){
  printf("%d %f\n",s[i].rollno,s[i].marks);
}
return 0;
}
```

Structure and Function

- We can pass structure members as arguments to a function.

```
#include <stdio.h>
struct student{
  int no;
 int marks;
}s;

void display(int x,int y)
{
printf("%d %d",x,y);
}
```

```
int main()
{
 s.no=10;
  s.marks=20;
  display(s.no,s.marks);
 return 0;
}
```

Structure and Function

- We can pass struct variables as arguments to a function.

```
#include <stdio.h>
struct student {
  char name[50];
  int age;
};
void display(struct student s) {
  printf("\nDisplaying information\n");
  printf("Name: %s", s.name);
  printf("\nAge: %d", s.age);
}
```

```
int main() {
  struct student s1;

  printf("Enter name: ");

  // read string input from the user until \n is entered
  // \n is discarded
  scanf("%[^\n]%*c", s1.name);

  printf("Enter age: ");
  scanf("%d", &s1.age);

  display(s1); // passing struct as an argument

  return 0;
}
```

Union

- **union is a group of variables of different data types represented by a single name.**

- Like Structures, union is a user defined data type.

Union

```
union union_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
}variable;
```

Example1:

```
union employee
{
 int id;
   char name[50];
   float salary;
}e;
```

union-example

```
#include<stdio.h>
#include <string.h>
union student
{   int rollno;
   char name[50];
   float cgpa;
}s1;  //declaring s1 variable for structure
```

```
int main( )
{
  s1.rollno=39;
printf( "student1-id : %d\n", s1.rollno);
   strcpy(s1.name, "ABCD");//copying string into char array
printf( "student1-1 name : %s\n", s1.name);
s1.cgpa=9.8;
printf( "student1-cgpa : %f\n", s1.cgpa);
return 0;
}
```

Bitfields

a bit field is a data structure that allows the programmer to allocate memory to structures and unions in bits in order to utilize computer memory in an efficient manner.

Need for Bit Fields in C

Bit fields are of great significance in C programming, because of the following reasons:

- Used to reduce memory consumption.
- Easy to implement.
- Provides flexibility to the code.

Declaration

```
struct structname
{
data_type variable_name : size_in_bits;
};
```

Bitfields-example

```
#include <stdio.h>
// A structure with forced alignment
struct test {
int x :10 ;
int y: 4;
  }s;
```

```
int main()
{
   s.x=20;
   s.y=3;
   printf("\n%d",s.x);
    printf("\n%d",s.y);
   printf("\nSize of test is %lu
bytes\n",sizeof(s));

   return 0;
}
```

Enumeration (or enum) in C

- Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

- The enum in C is also known as the enumerated type.

syntax:

```
enum flag{integer_const1, integer_const2,.....integter_constN};
```

Enumeration (or enum) in C-example

```
#include <stdio.h>
int main()
{
 enum day{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
printf("\n%d", Sunday);
printf("\n%d", Monday);
printf("\n%d", Tuesday);
printf("\n%d", Wednesday);
printf("\n%d", Thursday);
printf("\n%d", Friday);
printf("\n%d", Saturday);

    return 0;
}
```

typedef

- C programming language provides a keyword called **typedef**, which you can use to give a type a new name.
- **typedef** is a keyword used in C programming to provide some meaningful names to the already existing variable in the C program.

Syntax:

typedef originaldatatypename newname;

Example:

typedef int Integer;

Then,we can use Integer in the place of int.

typedef-example

```
#include <stdio.h>
int main()
{
typedef unsigned int unit;
unit i,j;
i=10;
j=20;
printf("Value of i is :%d",i);
printf("\nValue of j is :%d",j);
return 0;
}
```

Using typedef with structures

```
typedef struct student
{
char name[20];
int age;
}stud;
// we can use ‘stud’ for structure student from this point onwards….
int main()
{
stud s1, s2;
…..
}
```

FILES

Stream is a sequence of data bytes, which is used to read and write data to a file.
The streams that represent the input data of a program are known as **input streams**.
The streams that represent the output data of a program are known as **output streams**.
A stream acts as an interface between a program and an input/output device.

Relationship between streams and I/O devices

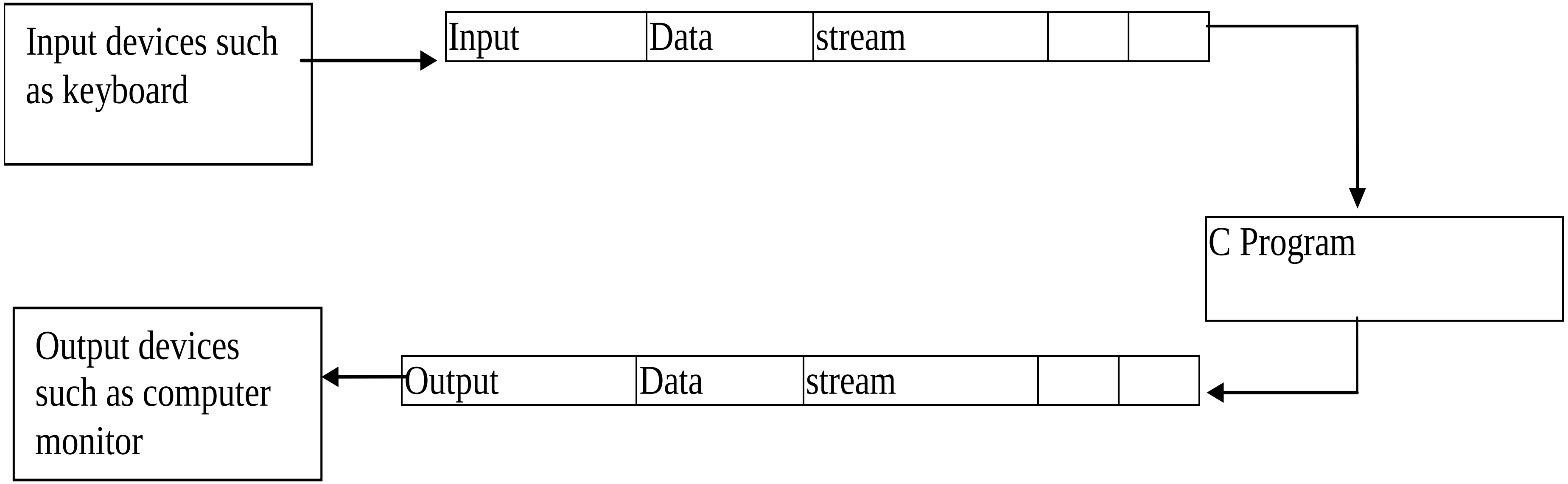

Input devices such as keyboard
Input
Data
stream
C Program
Output devices such as computer monitor
Output
Data
stream

Fig: Relationship between streams and I/O devices

FILES

- **A file is a collection of bytes stored on a secondary storage device (generally a disk).**
- File is a collection of records.
- File is a place on the disk where a group of related data is stored.
- Files are used to store data.
- Essentially there are two kinds of files

1) text files 2)binary files

FILES

1)Text files:

- A text file can be a stream of characters that a computer can process sequentially in forward direction.
- A text file contains only textual information like alphabets, digits and special symbols .

2)Binary files:

- It is also collection of bytes. Binary files can be either processed sequentially or, depending on the needs of the application.
- The binary file is generally in a form, which can be interpreted and understood by a computer system.

File Operations:

There are different operations that can be carried out on a file. These are:

- **1. Creation of a new file**
- **2. Opening an existing file**
- **3. Reading from a file**
- **4. Writing to a file**
- **5. Moving to a specific location in a file (seeking)**
- **6. Closing a file**

The File Pointer

- C communicates with files using a new datatype called a file pointer.

- FILE pointer is struct data type which has been defined in standard library stdio.h. This data type points to a stream or a null value. It has been defined in stdio.h.

- A *file pointer* is a pointer to a structure of type FILE.
- To obtain a file pointer variable, use a statement like this: **FILE * fp;**

Opening a file :

- Usethe following declaration before opening a file or creating a new file

```
FILE * fp;
```

Syntax for opening a file:

```
FILE *fp;
fp = fopen (“filename”, ”mode”);
```

- fp is a pointer variable which contains address of the structure FILE which has been defined in the header file “stdio.h”.
- **fopen( ):** open a file in specified mode.

- **File Opening Modes** (mode may be anyone of the following):((r, w, a, r+, w+, a+))

a)TEXT FILES MODES:

(i) r : Open a text file for reading:

"r" Searches the file. If the file exists, loads it in to memory and sets up a pointer which points to the first character in it. If the file doesn't exist it returns NULL. **fp=fopen("filename",r);**

(ii) w : Create a text file for writing:

"w" Searches file if the file exists it contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file. **fp=fopen("filename",w);**

Operations possible - writing to the file.

(iii) a : Append to a text file:

"a" Searches file. If the file exists, loads it in to memory and sets up a pointer which points to the last character in it. If the file doesn't exist a new file is created. Returns NULL, if unable to open file. Operations possible - Appending new contents at the end of file.

(iv) r+ open a text file for read/write :

“r+” Searches file. If it exists, loads it in to memory and sets up a pointer which **points to the first character in it**. If file doesn’t exist it returns NULL.

Operations possible - reading existing contents, writing new contents, modifying existing contents of the file.

(v) w+ Create a text file for read/write :

“w+” Searches file. If the file exists, **it contents are destroyed**. It the file doesn’t exist a new file is created. Returns NULL if unable to open file.

Operations possible – writing new contents, reading them back and modifying existing contents of the file.

(vi) a+ Append or create a text file for read/write :

”**a+”:**Searches file. If the file is opened successfully **fopen( )** loads it into memory and sets up a pointer which points to the last character in it. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open file.

Operations possible - reading existing contents, appending new contents to end of file. Cannot modify existing contents.

b)BINARY FILES MODES:((rb, wb ,ab, r+b, w+b, a+b))

- rb : Open a binary file for reading.
- wb: Create a binary file for writing.
- ab : Append to a binary file.
- 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.

Closing the file:

- A file must be closed after performing any operation (either reading or writing) on it.
- A file can be closed using **fclose** function. It takes the following form.

```
fclose(fp);
```

Unformatted I/O:

Unformatted I/O functions works without any format specifier(control string).

Reading from a File :

fgetc(): To read the file's contents from memory(read a single character from a file).

Ex:file1.c

getc(): it is same as fgetc().

getw(): It is used to read a single integer from a file.

syntax: **i=getw(fp);**

fgets( ) :read a string from a file.

fgets(*str, numberof bytes, fp*);

reads numberofbytes from file and stores into str .

Writing to a File :

fputc(): To write data into a file (character a ta time).

Ex:file2.c,file.c

putc() :it is same as fputc().

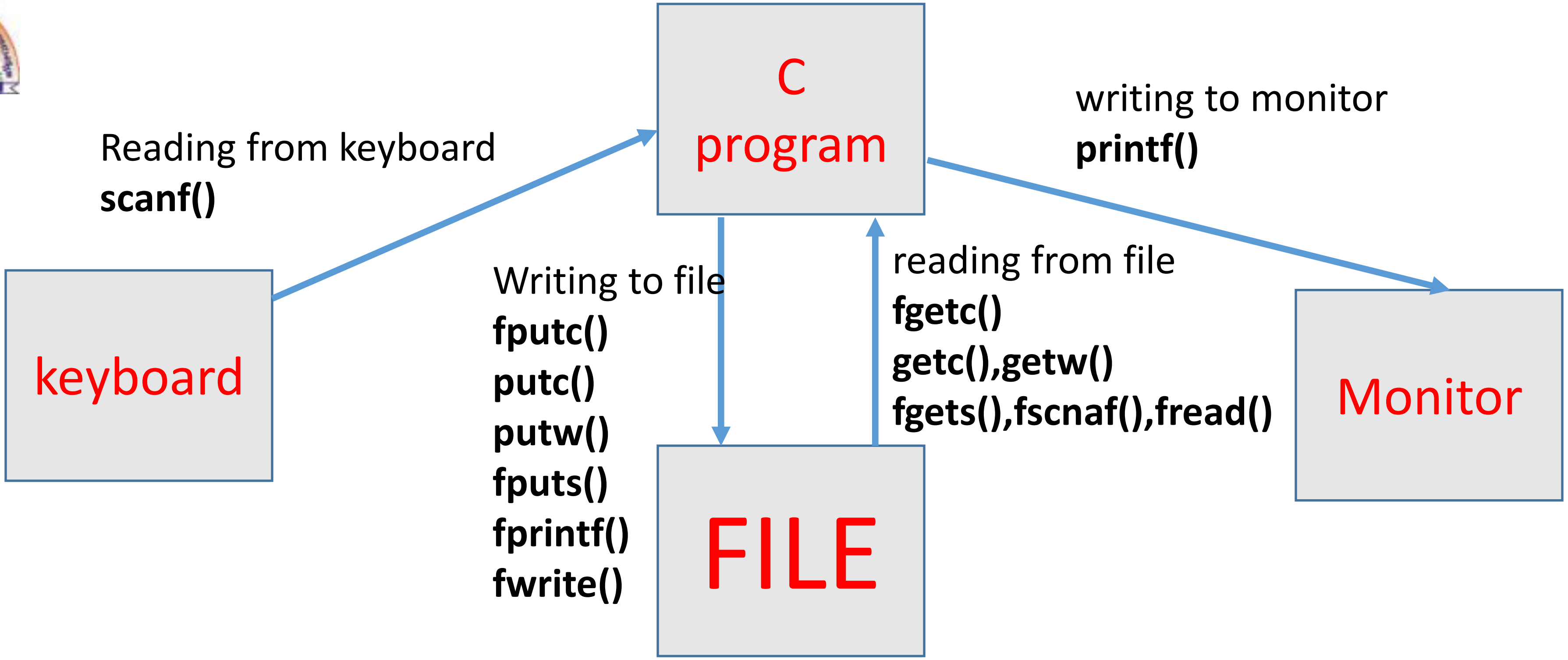
putw(): It is used to write a single integer to a file.(ex: file3.c)

Syntax: **putw(i,fp);**

fputs( ) :write a string to a file.

fputs(*str, fp*);

writes data in str into filepointed by fp. (file4.c)



C program

Reading from keyboard
scanf()

keyboard

Writing to file
fputc()
putc()
putw()
fputs()
fprintf()
fwrite()

FILE

reading from file
fgetc()
getc(),getw()
fgets(),fscnaf(),fread()

writing to monitor
printf()

Monitor

fgetc() and fputc()-example

fgetc()

12.1:Open a file and to print the contents of the file on screen

```
#include<stdio.h>
#include<conio.h>
int main() {
   FILE *f;
   char s;
f=fopen(“test.txt","r");
while((s=fgetc(f))!=EOF)
{
 printf("%c",s);
   }
   fclose(f);
return 0;
}
```

fputc()

```
int main( )
{
FILE *fp ;
char ch=' ';
fp = fopen ( "newfile.txt", "w" ) ;
printf("enter text");
while(ch!='$')
{
scanf("%c",&ch);
if(ch!='$')
fputc(ch,fp);
}
fclose(fp);
return 0;}
```

getw and putw -example

```
#include<stdio.h>
int main()
{
FILE *fp;
int no,i;
fp=fopen("f11.txt","w");
printf("\n enter 10 numbers");
for(i=1;i<=10;i++)
{
printf("enter no");
scanf("%d",&no);
putw(no,fp);
}
fclose(fp);
fp=fopen("f11.txt","r");
printf(" numbers in the
file are\n");
for(i=1;i<=10;i++)
{
no=getw(fp);
printf(" %d",no);
}
return 0;}
```

program on fgets and fputs

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int choice;
char str[80];
FILE *fp;
fp = fopen("test.txt", "w");
do
{
printf("\n Enter a string :\n");
scanf("%s",str);
fputs(str, fp);
printf("\n1. read 2. stop enter 1 or 2:\n");
scanf("%d",&choice);
}while(choice!=2);
fclose(fp);
//open file and print integers on screen
fp=fopen("test.txt","r");
printf("\n data in the file is\n ");
while(!feof(fp))
{
fgets(str,120,fp);
printf(" %s",str);
}
return 0;
}
```

FORMATTED I/O

(Formatted I/O functions works with format specifier(control string).

fscanf():(Reading from a file)

It is used to read data from a file.

syntax:

```
  fscanf (filepointer,”control string”, listofvariables);
```

[to read data from file]

```
  fscanf (stdin,”control string”, listofvariables);
```

[to read data from keyboard]

fprintf():(writing to a file)

It is used to write data to a file.

syntax :

```
  fprintf (filepointer,”control string”, listofvariables);
```

[to write data into file]

```
  fprintf (stdout,”control string”, listofvariables);
```

[to write data in to monitor]

ex:file5.c

fscanf() - fprintf() example

```
#include <stdio.h>
#include <stdlib.h>
main()
{
FILE *fp;
char s[80];
int t;
fp=fopen(“sample.txt", "w");
printf("Enter a string and a number: ");
fscanf(stdin, "%s%d", &s, &t); /* read from keyboard */
fprintf(fp, "%s %d", s, t); /* write to file */
fclose(fp);
fp=fopen(“sample.txt","r");
fscanf(fp, "%s%d", s, &t); /* read from file */
fprintf(stdout, "%s %d", s, t); /* print on screen */
return 0;}
```

fread( ) and fwrite( ):

To read and write data types that are longer than 1 byte, the C file system provides two functions:

*These functions allow the reading and writing of blocks of any type of data.

Syntax :

fread(*buffer*, *numberofbytes*, *count*,filepointer);

fwrite(*buffer*, *numberofbytes*,*count*, filepointer);

buffer : Pointer to a block of memory(generally buffer is a character array)

numberofbytes : Size in bytes of each element to be read.

count: Number of elements, each one with a size of ***numberofbytes*** .

ex:file6.c

fread( ) and fwrite( )-example

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fp;
float bal[5] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
int i;
/* write the values to the file  */
fp=fopen("test1.txt", "wb");
fwrite(bal, sizeof(float), 5, fp);
fclose(fp);
/* read the values  from file*/
fp=fopen("test1.txt", "rb");
fread(bal, sizeof(float), 5,fp)  ;
fclose(fp);
printf("\n printing data\n");
for(i=0; i<5; i++)
{ printf("\n%f ", bal[i]);   }
return 0;
}
```

Random File Access

C supports following functions for random access file processing.

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

File handling functions(Random File Access):

a)fseek function:

fseek function is used to move the file position to a desired location within the file.

Syntax: **fseek(fileptr, offset, position);**

Fileptr is a pointer to the file concerned,

offset is a number variable of type long and position is an integer number. The offset specifics the number of positions(bytes) to the moved from the location specified by position.

position can take one of the following three values

Values Meaning

Beginning of file: 0 (or) SEEK_SET

Current position : 1 (or) SEEK_CUR

End of file : 2 (or) SEEK_END

File handling functions:

fseek function:

offset may be positive meaning move forwards or negative meaning move backwards. The following examples illustrate the operation of the fseek function:

statement	**Meaning**
fseek(fp,0L,0)	Go to beginning
fseek(fp, 0L, 1)	Stays at current position
fseek(fp, 0L, 2)	Go to end of the file, past the last character of the file
fseek(fp, m, 0)	Move to (m+1)th byte in the file
fseek(fp, m, 1)	Go forward by m bytes
fseek(fp, -m, 1)	Go backward by m bytes from the current position
fseek(fp, - m, 2)	Go backward by m bytes from the end

File handling functions:

b)ftell():

ftell takes a file pointer and returns a number of type long that corresponds to the current position. This function is useful in saving the current position of a file, which can be used later in the program.

It takes the following form

n = ftell(fp);

n would give the relative offset(in bytes) of the current position. This means that n bytes have already been read (or written).

c)rewind():

rewind takes a file pointer and resets the position to the start of the file.

rewind(fp);

n = ftell(fp);

n would return 0

Ex:file7.c

fseek( ),ftell() and rewind( )-example

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
char ch=' ';
clrscr();
fp=fopen("test.txt","r");
printf("\n data in the file is : \n");
while(!feof(fp))
{
ch=fgetc(fp);
printf("%c",ch);
}
printf("\n filepointer is at %d \n ",ftell(fp));
rewind(fp);
```

```
printf(" Now after rewind,filepointer is at %d
\n",ftell(fp));
fseek(fp,8,0);
printf("\n Now  after fseek,pointer is at %d
\n",ftell(fp));
while(!feof(fp))
{
ch=fgetc(fp);
printf("%c",ch);
}
fseek(fp,-7,1);printf("\nNow  after fseek,pointer
 is at %d \n",ftell(fp));
while(!feof(fp))
{
ch=fgetc(fp);
printf("%c",ch);
}
}
```

Error Handling During I/O Operations(feof(),ferror()):

feof(): ***feof() function is used to detect the end of file(EOF).***

int feof(FILE *fp); or feof(fp);

(It takes file pointer as its argument and returns -1 when EOF is reached, otherwise it returns 0.)

```
# include <stdio.h>
main( )
{
FILE *fp ;
char ch ;
fp = fopen ( "PR1.C", "r" ) ;
while (!feof(fp) )
{
ch = fgetc ( fp ) ;
printf ( "%c", ch ) ;
}
fclose ( fp ) ;
}
```

feof( ) returns true if the end of the file has been reached; otherwise, it returns zero. Therefore, the following routine reads a file until the end of the file is encountered:

while(!feof(fp)) ch = getc(fp);

Of course, you can apply this method to text files as well as binary files.

Ex:file8.c

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

- Its syntax is: **ferror(fp);**{It takes file pointer as its argument and returns -1 when error has occurred, otherwise it returns 0.}

```
main( )
{
FILE *fp ;
char ch ;
fp = fopen ( "TRIAL", "w");
while ( !feof ( fp ) )
 {   ch = fgetc ( fp ) ;
if ( ferror(fp ) )
{ printf ( "Error in reading file" ) ;
break ; }
}
else
printf ( "%c", ch ) ;
}
```

In this program the **fgetc( )** function would fails first time because file has been opened for writing,and we are trying to reading the data. **fgetc( )** is used to read from the file. The moment the error occurs **ferror( )** returns a non-zero value and the **if** block gets executed.

Ex:file9.c

Copy content of one file to another file

```
# include <stdio.h>
void main()
{
FILE *f1, *f2;
char ch;
f1 = fopen("file1.txt", "r");
f2 = fopen ("filecopied.txt", "w");
while (!(feof(f1)))
{
ch = fgetc(f1);
fputc(ch,f2);
}
printf("\nfile copied
successfully");
fclose(f1);
fclose(f2);
}
```

Merge two files and store content in another file

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
FILE *fp1 = fopen("file1.txt", "r");
FILE *fp2 = fopen("file2.txt", "r");
FILE *fp3 = fopen("file3.txt", "w");
 char c;
   if (fp1 == NULL || fp2 == NULL
|| fp3 == NULL)
  {
       puts("Could not open files");
       exit(0);
  }
```

```
// Copy contents of first file to file3.txt
  while ((c = fgetc(fp1)) != EOF)
    fputc(c, fp3);
    // Copy contents of second file to file3.txt
  while ((c = fgetc(fp2)) != EOF)
    fputc(c, fp3);

  printf("Merged file1.txt and
file2.txt into file3.txt");

  fclose(fp1);
  fclose(fp2);
  fclose(fp3);
  return 0;
}
```