C Programming

FILES

Dr. Suresh Mudunuri, SRKR Engineering College

UNIT 5 Syllabus

UNIT-V (10 Hrs)	Functions: Designing, Structured Programs, Function in C, User Defined Functions, Inter Function Communication, Standard Functions, Passing Array to Functions, Passing Pointers to Functions, Recursion Text Input / Output: Files, Streams, Standard Library Input / Output Functions, Formatting Input / Output Functions, Character Input / Output Functions Binary Input / Output: Text versus Binary Streams, Standard Library, Functions for Files, Converting File Type.

typedef – to define new datatype

'typedef' is a keyword,which allows you to specify a new name for a datatype which is already defined in c language program.

Syntax:

typedef <datatype> <newname>

```
/* Re-defining int type as Integer type */
typedef int Integer;
int main( ) {
    Integer a ,b , sub;
    a = 20,b = 10;
    sub = a - b;
    printf("%d - %d = %d", a, b, sub);
}
/* Defining structure with typedef to avoid
   repeated usage of  struct keyword */
struct timing  {
  int hours;
  int minutes;
} ;
typedef struct timing TIME;
int main( ) {
   TIME t1, t2 , *t;
   t = (TIME *) calloc (10, sizeof( TIME ));
}
```

bitfieds

```
struct  playcard  {
  unsigned int king ;
  unsigned int suit ;
};
```

Above structure occupies 4 bytes of memory. But if the member king accepts a value between 1 to 13 and the member suit accepts any value of 0, 1, 2 and 3 .

So we can create a more packed representation of above structure with bitfields.

```
struct  playcard {
  unsigned  int king : 4;
  unsigned  int suit : 2;
};
```

A bitfield is a set of adjacent bits within a single machine word.

4-bit field called king that is capable of storing the 16 values 0 to 15, and a 2-bit field called suit that is capable of storing values 0, 1, 2, and 3. So the entire structure variable occupies only one byte.

Note : arrays of bit fields and a pointer to address a bit field is not permitted.

Enumeration – a set of named integers, makes program more readable

Declaration of enumeration :

enum <enum_name> { member1, member2, ... } ;

Example :

enum option { YES, NO, CANCEL } ;

By default YES has value 0, NO has value 1 and CANCEL has 2.

enum direction { EAST = 1, SOUTH, WEST = 6, NORTH } ;

Now EAST has value 1, SOUTH has value 2, WEST has value 6, and NORTH has value 7.

Enumerated types can be converted implicitly or cast explicitly.

int x = WEST ; /* Valid. x contains 6. */

enum direction y ; y = (enum direction) 2 ; /* Valid. Y contains SOUTH */

```
#include<stdio.h>
int main()  {
    int signal;
    printf ("\t\t\t MENU \n\t1.RED \n");
    printf ("\t2.ORANGE\n\t3.GREEN \n" );
    printf ("\n\t Enter the signal : " );
    scanf ("%d", &signal );
    switch(signal)
    {
       case 1:
         printf("\t Stop and Wait!"); break;
       case 2:
         printf("\t Ready to start!"); break;
       case 3:
         printf("\t Start and go!");  break;
    }
}
```

```
#include<stdio.h>
enum color {RED = 1,ORANGE,GREEN };
int main()  {
    enum color signal;
    printf ("\t\t\t MENU \n\t1.RED \n");
    printf ("\t2.ORANGE\n\t3.GREEN\n");
    printf ("\n\t Enter the signal : ");
    scanf ("%d", &signal);
    switch(signal) {
       case RED:
         printf("\t Stop and Wait!"); break;
       case ORANGE:
         printf("\t Ready to start!"); break;
       case GREEN:
         printf("\t Start and go!");    break;
    }
}
```

Console I / O Vs File I / O

- scanf( ) and printf( ) functions read and write data which always uses the terminal (keyboard and screen) as the target.
- It becomes confusing and time consuming to use large volumes of data through terminals.
- The entire data is lost when either program terminates or computer is turned off.
- Some times it may be necessary to store data in a manner that can be later retrieved and processed.

  This leads to employ the concept of FILES to store data permanently in the system.

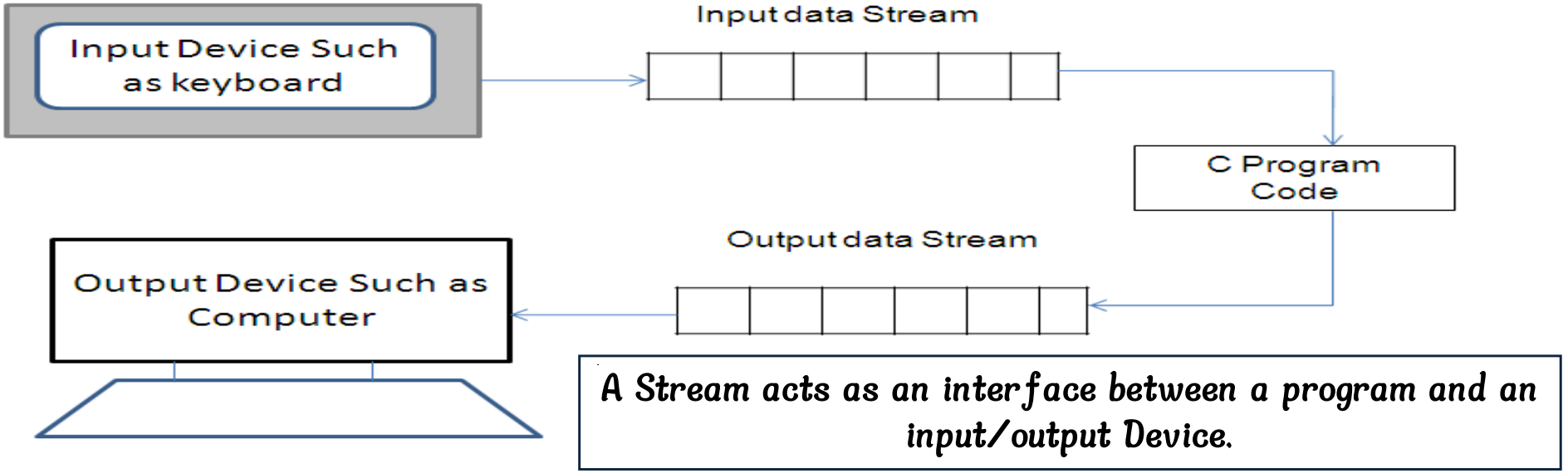
Record is logical group of data fields that comprise a single row of information, which describes the characteristics of an object.

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

A **File** is a place on disk where a group of related data (records) can be stored

File Operations

1. Creating 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 (seek)
6. Closing a file



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**, where as the streams that represent the output data of a program are known as **Output Streams.**

Input streams gets the data from different **input devices** such as keyboard and mouse and provide input data to the program.

Output Streams obtain data from the program and write that on different **Output Devices** such as Memory or print them on the Screen.

Types of Files

1. Text file : It can be thought of as a stream of characters that can be processed sequentially and in forward direction only.
2. Binary file : It is collection of bytes like images.
3. Sequential File: Data stored sequentially, to read the last record of the file, we need to traverse all the previous records before it. Ex: files on magnetic tapes.
4. Random Access File: Data can be accessed and modified randomly. We can read any record directly. Ex : files on disks.

Steps involved using files

```
/*program to write and read data from file*/
#include<stdio.h>
void  main() {
    FILE *fp;
    char ch;
    fp = fopen("data.txt", "w");
    /* writing to file */
    fprintf(fp, "This is fprintf...\n");
    fputs("This is testing for fputs..\n", fp);
    fputc('A', fp);
    fclose(fp);

    /* reading from file */

      fp = fopen("data.txt","r");
      ch=fgetc(fp);
      while(ch != EOF)
            {
               printf("%c", ch);
               ch=fgetc(fp);
            }
      fclose(fp);
}
```

1. Declaring FILE pointer variable :

Syntax :

```
FILE *file_pointer1;
```

2. Open a file using fopen() function :

Syntax :

```
fp= fopen("filename","mode of access");
```

3. fputc() – Used to write a character to the file.

Syntax :

```
fputc(character, file_pointer);
```

4. fgetc() – Used to read a character to the file.

Syntax :

```
Ch = fgetc(file_pointer);
```

5. Close a file using fclose() function :

Syntax :

```
fclose(file_pointer);
```

```
/* creating a new file */
int main(){

    char ch;FILE *fp;

    printf("\nEnter the text\n");
    fp=fopen("str.txt","w");
    while((ch=getchar())!=EOF)
          fputc(ch,fp);
    fclose(fp);
}
```

file pointer used to handle files

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

fputc(character,filepointer);

Also, you can use **putc()**.
putc(character,filepointer);

fclose(filepointer);

```
/* Reading the contents of existing file */
#include<stdio.h>
int main()  {
    FILE *fp;
    char ch;
    fp=fopen("str.txt","r");
    while((ch=fgetc(fp))!=EOF)
        printf("%c",ch);
    fclose(fp);
}
```

```
/* appending data to an existing file */
int main()  {
    FILE *fp; char ch;
    printf("\nEnter the text\n");
    printf("\n\t(Press ctrl+Z after
                completing text)\n");
    fp=fopen("str.txt","a");
    while((ch=getchar())!=EOF)
            fputc(ch,fp);
    fclose(fp);
}
```

r -- open a file in read mode -- if file exits, the marker is positioned at beginning. -- if file does not exist, error returned.	r+ -- open a file in read and write mode -- if file exits, the marker is positioned at beginning. -- if file does not exist, NULL returned.
w -- open a file in write mode -- if file exits, all its data is erased. -- if file does not exist, it is created.	w+ -- open a file in read and write mode -- if file exits, all its data is erased. -- if file does not exist, it is created.
a -- open a file in append mode -- if file exits, the marker is positioned at end. -- if file does not exist, it is created.	a+ -- open a file in read and append mode -- if file exits, the marker is positioned at end. -- if file does not exist, it is created.

rb , wb , ab, rb+ , wb+ , ab+ are modes to operate a file as binary file.

Mode	Read	Write	Crate New File*	Truncate
r	1	0	0	0
w	0	1	1	1
a	0	1	1	0
r+	1	1	0	0
w+	1	1	1	1
a+	1	1	1	0
*Creates a new file if it doesn't exist				

File Input / Output Functions	
fopen(filename, mode)	Open existing file / Create new file
fclose(fp)	Closes a file associated with file pointer.
closeall ( )	Closes all opened files with fopen()
fgetc(fp)	Reads character from current position and advances the pointer to next character.
fprintf( )	Writes all types of data values to the file.
fscanf()	Reads all types of data values from a file.
fgets()	Reads string from a file.
fputs()	Writes string to a file.
getw()	Reads integer from a file.
putw()	Writes integer to a file.
fread()	Reads structured data written by fwrite() function
fwrite()	Writes block of structured data to the file.
fseek()	Sets the pointer position anywhere in the file
feof()	Detects the end of file.
rewind()	Sets the record pointer at the beginning of the file.
ferror()	Reports error occurred while read/write operations
fflush()	Clears buffer of input stream and writes buffer of output stream.
ftell()	Returns the current pointer position.

Text files Vs Binary Files

```
/* Copying one binary file to other */

#include<stdio.h>
int main( )
{
   FILE *fs,*ft;
   char ch;
   fs=fopen("pr1.exe","rb");
   if(fs==NULL){
      printf("\nCannot Open the file");
      exit(0);
   }
   ft=fopen("newpr1.exe","wb");
   if(ft==NULL) {
       printf("\nCannot open the file");
       fclose(fs);
       exit( 0);
    }
    while((ch=fgetc(fs))!=EOF)
              fputc(ch,ft);
    fclose(fs);
    fclose(ft);
}
```

"rb"	→	open a file in read mode
"wb"	→	open a file in write mode
"ab"	→	open a file in append mode
"rb+"	→	open a pre-existing file in read and write mode
"wb+"	→	open a file in read and write mode
"ab+"	→	open a file in read and append mode

Text File :

i) Data are human readable characters.
ii) Each line ends with a newline character.
iii) Ctrl+z or Ctrl+d is end of file character.
iv) Data is read in forward direction only.
v) Data is converted into the internal format before being stored in memory.

Binary File :

i) Data is in the form of sequence of bytes.
ii) There are no lines or newline character.
iii) An EOF marker is used.
iv) Data may be read in any direction.
v) Data stored in file are in same format that they are stored in memory.

Random Access File

```
int main() {
  int n,i;
  char *str="abcdefghijklmnopqrstuvwxyz";
  FILE *fp = fopen("one.txt","w");
  fprintf(fp,"%s",str);
  fclose(fp);

  fp = fopen("one.txt","r");
  fseek(fp, 3 ,SEEK_SET);
  printf("Text from position %ld : \n\t", ftell(fp));

  for(i=0; i < 5; i++)  putchar(getc(fp));
  fseek(fp, 4 ,SEEK_CUR);
  printf("\nText from position %ld : \n\t",ftell(fp));

  for(i=0; i < 6; i++)  putchar(getc(fp));
  fseek(fp,  - 10 , SEEK_END);
  printf("\nText from position %ld : \n\t",ftell(fp));

  for(i=0; i < 5; i++)  putchar(getc(fp));
  printf("\nCurrent position : %ld",ftell(fp));

  rewind(fp);
  printf("\nText from starting : \n\t");

  for(i=0;i < 8 ; i++) putchar(getc(fp));
  fclose(fp);
  return 0;
}
```


ftell(file_pointer)

-- returns the current position of file pointer in terms of bytes from the beginning.

rewind(file-pointer)

-- moves the file pointer to the starting of the file, and reset it.

fseek(fileptr, offset, position)

- moves the file pointer to the location (position + offset)

position :

- SEEK_SET – beginning of file
- SEEK_CUR – current position
- SEEK_END – end of the file

output :

```
Text from position 3 :
        defgh
Text from position 12 :
        mnopqr
Text from position 16 :
        qrstu
Current position : 21
Text from starting :
        abcdefgh
```

Formatted I / O

```
/* using fscanf() and fprintf() functions */
#include<stdio.h>
int main( ) {
   FILE *fp;
   int rno , i;
   float avg;
   char  name[20] ,  filename[15];
   printf("\nEnter the filename\n");
   scanf("%s",filename);
   fp=fopen(filename,"w");
   for(i=1;i<=3;i++)  {
       printf("Enter rno,name,average
           of student no:%d",i);
       scanf("%d %s %f",&rno,name,&avg);
       fprintf(fp,"%d %s %f\n",rno,name,avg);
   }
   fclose(fp);
   fp=fopen ( filename, "r" );
   for(i=1;i<=3;i++)  {
       fscanf(fp,"%d %s %f",&rno,name,&avg);
       printf("\n%d %s %f",rno,name,avg);
   }
   fclose(fp);
}
```

```
/*Receives strings from keyboard
   and writes them to file
   and prints on screen*/
#include<stdio.h>
int main( )  {
   FILE *fp;
   char s[80];
   fp=fopen("poem.txt","w");
   if(fp==NULL)  {
       puts("Cannot open file");exit(0);
    }
    printf("\nEnter a few lines of text:\n");
    while(strlen(gets(s))>0){
        fputs(s,fp);
        fputs("\n",fp);
    }
    fclose(fp);
    fp=fopen("poem.txt","r");
    if(fp==NULL){
       puts("Cannot open file"); exit(0);
   }
   printf("\nContents of file:\n");
   while(fgets(s,79,fp)!=NULL)
       printf("%s",s);
   fclose(fp);
}
```

```c
/* using putw() and getw() functions */
#include<stdio.h>
int main( )  {
   FILE *fp1,*fp2; int i,n;
   char *filename;
   clrscr();
   fp1=fopen("test.txt","w");
   for(i=10;i<=50;i+=10)
     putw(i,fp1);
   fclose(fp1);
   do {
     printf("\nEnter the filename : \n");
     scanf("%s",filename);
     fp2=fopen(filename,"r");
     if(fp2==NULL)
      printf("\nCannot open the file");
   } while(fp2==NULL);
   while(!feof(fp2)) {
     n=getw(fp2);
     if(n==-1) printf("\nRan out of data");
     else printf("\n%d",n);
   }
   fclose(fp2);
   getch();
}
```

Standard I / O

fputc()	fgetc()	Individual characters
fputs()	fgets()	Character Strings
fprintf()	fscanf()	Formatted ASCII
fwrite()	fread()	Binary files
write()	read()	Low-level binary

Predefined Streams

NAME	MEANING
stdin	Standard input (from keyboard)
stdout	Standard output (to monitor)
stderr	Standard error output (to monitor)
stdaux	Standard auxiliary (both input and output)
stdprn	Standard printer output(to printer)

Handling Records (structures) in a File

```
struct player {
  char name[40]; int age; int runs;
} p1,p2;
void main() {
     int i ; FILE *fp = fopen ( "player.txt", "w");
     if(fp == NULL)  {
         printf ("\nCannot open file."); exit(0);
     }
     for(i=0;i<3;i++) {
       printf("Enter name, age, runs of a player : ");
       scanf("%s %d %d",p1.name, &p1.age,&p1.runs);
       fwrite(&p1,sizeof(p1),1,fp);
     }
    fclose(fp);
    fp = fopen("player.txt","r");
    printf("\nRecords Entered : \n");
    for(i=0;i<3;i++) {
       fread(&p2,sizeof(p2),1,fp);
       printf("\nName : %s\nAge : %d\nRuns : %d",p2.name,p2.age,p2.runs);
   }
   fclose(fp);
}
```

Error Handling:

While operating on files, there may be a chance of having certain errors which will cause abnormal behavior in our programs.

1)Opening an file that was not present in the system.
2)Trying to read beyond the end of file mark.
3)Device overflow.
4)Trying to use a file that has not been opened.
5)Trying to perform an operation on a file when the file is opened for another type of operation.
6)Attempting to write to a write-protected file.

feof(fp) → returns non-zero integer value if we reach end of the file otherwise zero.
ferror(fp) → returns non-zero integer value if an error has been detected otherwise zero
perror(string) → prints the string, a colon and an error message specified by the compiler
file pointer (fp) will return NULL if it cannot open the specified file.

```
/* program on ferror( ) and perror ( ) */
#include<stdio.h>
int main(){
    FILE *fp;
    char ch;
    fp=fopen("str.txt","w");
    ch=getc(fp);
    if(ferror(fp))
          perror("Error Raised : ");
    else
        printf("%c",ch);
    fclose(fp);
}
```

```
#include<stdio.h>
main(){
FILE *fp1,*fp2;
int i,number;
char *filename;
fp1=fopen("TEST.txt","w");
for(i=10;i<=50;i+=10)
            putw(i,fp1);
fclose(fp1);
file:
printf("\nEnter the filename\n");
scanf("%s",filename);
fp2=fopen(filename,"r");
if(fp2==NULL){
     printf("\nCannot open the file");
     printf("\nType File name again");
      goto file;}
else{
        for(i=1;i<=10;i++){
            number=getw(fp2);
                    if(feof(fp2)){
                printf("\nRan out of data");
                break;}
else
    printf("\n%d",number); }  }
fclose(fp2);}
```

fp will return NULL if unable to open the file

feof(fp) returns 1 if it reaches end of file otherwise 0.

Output:

```
Enter the filename
TETS.txt
Cannot open the file
Type the File name again
Enter the filename
TEST.txt
10
20
30
40
50
Ran out of data.
```

Structure of FILE pointer

Type: FILE
File control structure for streams.

```
typedef struct {
    short    level;
    unsigned      flags;
    char     fd;
    unsigned char  hold;
    short         bsize;
    unsigned char *buffer, *curp;
    unsigned      istemp;
    short         token;
} FILE;
```

File status functions

feof(file_pointer)
-- to check if end of file has been reached.

ferror(file_pointer)
-- to check the error status of the file

clearerr(file_pointer)
-- to reset the error status of the file

File Management Functions

rename("old-filename","new-filename");
-- It renames the file with the new name

remove("filename")
-- It removes the file specified (macro)

unlink("filename");
-- It also removes the file name

fcloseall();
-- Closes all the opened streams in the program except standard streams.

fflush(file_pointer)
-- Bytes in the buffer transferred to file.

tmpfile ()
-- creates a temporary file, to be deleted when program is completed.

tmpnam("filename")
-- creates a unique file name