Console I/O

Console → keyboard + screen.

that is, if we are giving input through keyboard and output is pointing or going to screen. then, that is called Console

Not on files -

Types of I/O Input/output

(1) Console I/O →

(2) File I/O →

(1) Console I/O Functions → Functions to receive input from keyboard and write output to VDU. (Video display unit) (screen)

(2) File I/O Functions → Functions to perform I/O operations on a floppy disk or hard disk means directly receiving input from files and sending output to files.

Console I/O Functions

Formatted Functions

Functions allow us to supply the input in a fixed format and obtain the output in specified format.

Eg printf ("Format string", variable);

scanf ("Format string", list of variable);

In which we can do formatting how to print the output.

Unformatted functions

- Functions that are already formatted by a compiler.
- that can deal with a single character.
- or it can deal with a string of characters.

Eg getch() putch()
getche() putchar()
getchar()
gets() puts()

~~To read and write single character~~

```
scanf ("%c", &a);
printf ("%c", a);
```

```
a = getchar();
putchar(a);
```

For string

```
char str[10];
gets(str);
puts(str);
```

```
char str[10];
scanf("%s", str);
printf("%s", str);
```

↓ This is reverse

Difference between getch(), getche(), getchar()

getch

If we are getting input through scanf(), when we enter the character, you need to hit the Enter key whatever you have typed.

But we often want a function that will read a single character the instant it is typed without waiting for the Enter key to be hit.

getch() and getche() are two functions that serves this purpose.

These functions return the character that has been most recently typed.

'e' in getche() stands for echo.

It echoes (displays) the character that you typed to the screen.

getch() returns the character that you typed without echoing (or displaying) it on the screen.

getchar() → work similarly as getche(), It echoes the character that you typed on the screen, but requires Enter key to be typed after entering the character.

main()
{
char ch;

I/P
getch();
[It will not echo the character, you have just typed.

ch = getche();
[It will echo the character typed.

I/P a O/P a

ch = getchar();
[It will echo character, must be followed by Enter key

I/P A ↵ Enter
O/P A

}

---

putch(), putchar()

They print a character on the screen.

putch(ch);
putchar(ch);

I/P putch('Z'); Z

putchar('Z'); Z

File I/O

Getting Input from a file and sending output to a file through a program is called File I/O.

File operations

1. Creating a 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
6. Closing a file.

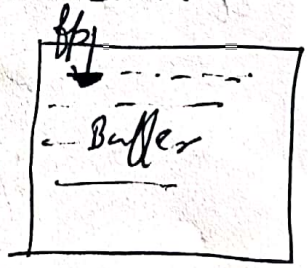
Step 1

1. Create a buffer area.

Buffer means temporary memory.
Means ~~or~~ file which you are opening for reading or writing it needs some memory to be operated upon.

Create a Temporary memory on which file will come for reading or writing.

```
FILE *fp;
```



FILE is a special structure type that creates an empty buffer area for a file.
fp is a pointer that points to the beginning of buffer area.

(2) opening a file

```
fp = fopen("Filename", "Mode-type");
```

eg: `fp = fopen("ABC.txt", "r.");`

6 File opening Modes

"r" → Mode to reading a file.
If file exist, file will be opened.
If file doesn't exist, it will return NULL.

"w" → working to a file
If file exist, contents are overwritten.
If file doesn't exist, new file will be created.
If unable to open or create a file, it will return NULL.

"a" → Adding new contents at the end of file
(means append mode)
If file exist, it sets up a pointer to the last character in a file.
If file doesn't exist, a new file is created.
If unable to open file, it returns NULL.

r → • Reading existing contents.
- Writing new contents.
, Modifying existing contents of the file

w+ → - Writing new contents.
- Reading them back.
= Modifying existing contents of file..

a+ → • Reading existing contents
• Appending new contents to end of file.
• Cannot modify existing contents.

---

(3) closing a file

fclose (file pointer);

fclose (fp);

Reading and writing character to a file

fgetc, fputc (OR fgetc and fputc)
to read and work.

Syntax: ch = fgetc (fp); Reading a single character from a file.

fputc (ch, fp); Writing a single character to a file.

EOF (End of File)

EOF is defined in stdio.h.

It is a last character of any text file. to mark the end of file. and its ASCII code is 26.

eg This is a uiet
It is in Chandigarh.

This is a uiet
It is in Chandigarh
EOF

At the end of Chandigarh, EOF will be placed to mark the end of text file.
It is invisible character.

Q1 WAP to read a text file

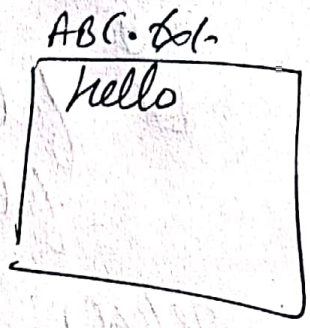
```
void main()
{
    FILE *fp;
    Char ch;
    fp = fopen("ABC.txt", "r");
    if (fp == NULL)
    printf("Cannot open file"); exit(0);
    while ((ch = fgetc(fp)) != EOF)
    {
        putchar(ch);
    }
    getch();
    fclose(fp);
}
```

① Make a file in Notepad

② store in that location where your programs reside.

Prgm:- Reading a Characters from ABC.txt one by one and storing it in Ch variable till EOF.
and Printing Ch one by one.

Eg Say file contents are "Hello"

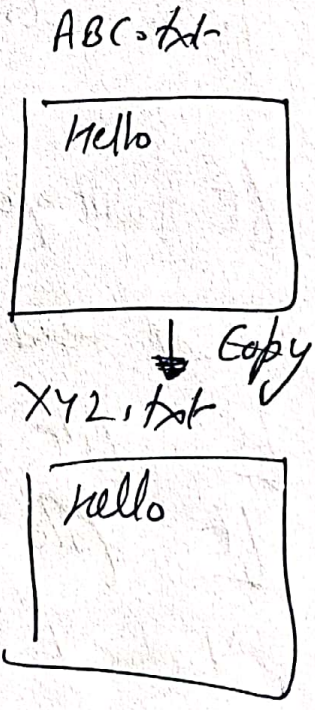


Loop = ① Ch = 'H'
② Ch = 'e'
③ Ch = 'l'
④ Ch = 'l'
⑤ Ch = 'o'
⑥ Ch = EOF

First of all, it will read 'H' then 'e', then 'l', ---- till EOF.

Q2 WAP to Copy a text file

Eg




```
void main()
{
    FILE *fs, *ft;
    Char ch;
    fs = fopen("ABC.txt", "r");
    ft = fopen("XYZ.txt", "w");
    if (fs == NULL)
    {
        printf("Cannot open file");
        exit(0);
    }
    if (ft == NULL)
    {
        printf("Cannot open file");
        exit(0);
    }
```

```
while (ch = fgetc(fs) != EOF)
{
    fputc(ch, ft);
}
    fclose(fs);
    fclose(ft);
getch();
}
```

Explanation:- 2 file pointers are required, 2 buffer areas.

fs, → source file.

ft, → target file

→ fs is already existing file, so it will open in 'r' read only mode

ft will open in 'w' → write mode.

→ It will read one by one character (ch) from fs file and writing character one by one to ft file.

→ You're to close the files using fclose().

WAP to count the no. of spaces, tabs, newlines and characters in a file.

```
main()
{
FILE *fp;
char ch;
int nol=0, not=0, nob=0, noc=0;
fp=fopen("PR1.C","r");
while((ch=fgetc(fp))!=EOF)
{
noc++;
if(ch==' ')
nob++;
if(ch=='\n')
nol++;
if(ch=='\t')
not++;
}
fclose(fp);
printf("No. of Characters including spaces=%d",noc);
printf("No of blanks =%d",nob);
printf("No of tabs=%d",not);
printf("No of lines=%d",nol);
printf("No of characters excluding spaces=%d",noc-nob);
}
```

I/P

```
Hello_friends
How_are_you
all ?
```

O/P

```
No of spaces= 3
No of newlines= 2
No of tabs= 0
No of Characters=28
(including spaces)
```

Q4 WAP to append a file

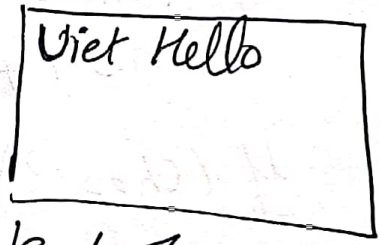
```
main()
{
  FILE *fs, *ft;
  char ch;
  fs = fopen("ABC.txt", "r");
  ft = fopen("XYZ.txt", "a");
  while (ch = fgetc(fs)) != EOF)
  {
    fputc(ch, ft);
  }
  getch();
}
```

I/P

ABC.txt

XYZ.txt

O/P



Explanation → Same as Copy program but instead of opening a ft file in 'w' mode, we opened in "a" mode. So pointer will be at the end of XYZ file. and ABC file contents will be added or appended to the end of XYZ contents.

So output will be **Viet Hello**

String I/O

① fgets() ② fputs.

```
fputs ("Hello", fp);
```

Putting string into a file (pointed by fp)

② fgets() → To read a string from a file.

```
Str = fgets (S, 79, fp);
fputs (str, ft);
```

```
Char S[80];
Char str[20];
```

It will read the string from file (fp) and put string (S) into Str.

i.e. It will read word by word from a file not the character

Imp:- Difference b/w Text file & Binary file

There are 2 modes in which we can open a file. text and binary.

① Handling of new lines

Text file	Binary file
In text mode, a newline character is converted into carriage-return + line feed combination on the disk.	In binary mode, these conversions will not take place.

\n \r\t\b	\n \n
② Representation of End-of-File (EOF) In text mode, a special character, whose ASCII value is 26, is inserted after the last character in the file to mark the end of file.	In binary mode, there is no such special character to mark the end of file. The binary mode files keep track of the end of file from the number of characters present in the directory entry of the file
③ Storage of Integers In text mode, numbers are stored as strings of characters. Thus, 1234, even if it occupies 2 bytes in memory, but when it will be transferred to disk, it will occupy 4 bytes, one byte per character. Eg 1234.56 is floating no, it will occupy 7 bytes on disk but 4 bytes in memory. This will require more disk space (Using Data will be Input or O/P [fprintf(), fscanf()	In binary mode, it stores the numbers in binary format. Means each number would occupy same number of bytes on disk as it occupies in memory. [Functions used to open a file in binary mode fread() & fwrite()

Record I/O in Text files.

⟹ Using fprintf() and fscanf()

Syntax :- `fprintf (fp "%d %s %f", x, n, m);`

3 parameters

- fp → file to write
- "%d %s %f" → Individual Memory Format String
- x, n, m → list of variables.

`fscanf (fp, "%d %s %f", &x, &n, &m);`

Example:

```
main()
{
    FILE *fp;
    char another = 'y';
    struct emp
    {
        char name[40];
        int age;
        float bs;
    };
    struct emp e;
    fp = fopen("Emp.dat", "w");
    if (fp == NULL)
    {
        puts("Cannot open file");
        exit(0);
    }
```

definition of structure (employee) Record.

⟹ opening file in "w" mode because Record is going to write in a file. fp -

```
while (another == 'y')
{
printf ("Enter name, age and basic salary");
scanf ("%s %d %f", e.name, &e.age, &e.bs);
fprintf (fp, "%s %d %f", e.name, e.age, e.bs);
printf ("Add another record (Y/N)");
fflush (stdin);
another = getche();
}
fclose (fp);
}
// Reading record from a file using fscanf()
fp = fopen ("Emp.data", "r");
while (fscanf (fp, "%s %d %f", e.name, &e.age, &e.bs) != EOF)
{
printf ("%s %d %f \n", e.name, e.age, e.bs);
}
fclose (fp);
}
```

Till you will press 'y', it will take input using printf and scanf() and write that record into file using fprintf()

Output:

```
Enter name, age and basic salary: Sunil 34 1250.50
Add another record (Y/N): Y
Enter name, age and basic salary: Sameer 21 1300.50
Add another record (Y/N): Y
Enter name, age and basic salary: Rahul 34 1400.55
Add another record (Y/N): N
```

Sunil	34	1250.50000
Sameer	21	1300.50000
Rahul	34	1400.50000

Record I/O in binary file

Modes will be now "wb" & "rb" "ab"

```
main()
{
FILE *fp;
char another='y';
struct emp
{
char name[40];
int age;
float bs;
};
struct emp e;
fp=fopen("emp.data","wb");
while(another=='y')
{
printf("Enter name, age and basic salary");
scanf("%s %d %f",e.name,&e.age,&e.bs);
fwrite(&e,sizeof(e),1,fp);
printf("Add another record Y/N");
another=getche();
}
fclose(fp);
```

Writing a record to a binary file using fwrite() function.

```
fp = fopen ("Emp.data", "rb");
while (fread (&e, sizeof (e), 1, fp) == 1)
{
    printf ("%-5s %d %f \n", e.name, e.age, e.bs);
}
fclose (fp);
}
```

Read record using fread

O/P will be same as earlier program.

Explanation

fread (&e, sizeof (e), 1, fp);

- &e → Address of structure record
- sizeof (e) → Size of structure/record
- 1 → No. of structure record (1)
- fp → file to work

fread (&e, sizeof (e), 1, fp);

- &e → do
- sizeof (e) → do
- 1 → do
- fp → file to read

Till then this function will return 1 i.e. No of record It will print that record on screen.

Moving a pointer / cursor in a file

Read

① **fseek()** → To set the cursor position in a file

⇒ `fseek (fp, offset bytes, byte-position);`

- fp → file where cursor to be set
- offset bytes → no. of bytes
- byte-position → 0 → from current, 1 → from beginning, 2 → from last.

Eg ① `fseek (fp, 0, SEEK_CUR);`

It will move the pointer 0 bytes from current position.

SEEK_CUR → 0
SEEK_SET → 1
SEEK_END → 2

② `fseek (fp, 2, SEEK_CUR);`

It will move the pointer 2 bytes from current position.

③ `fseek (fp, 0, SEEK_SET);`

It will set the pointer to beginning of file.

④ `fseek (fp, 0, SEEK_END);`

It will set the pointer to end of file.

⑤ `fseek (fp, -2, SEEK_END);`

It will move 2 bytes back from the end of file.

(2) ftell()

It will tell the position of a cursor

eg `long int position = ftell(fp);`

(3) rewind(fp);

This function will rewind the file ie.
It will take the cursor to the beginning of file.

Stdin, Stdout, Stderr (Terms).

Whenever a program is executed, three files are automatically opened by a compiler.

These files are stdin, stdout, stderr.

Stdin → This file stores whatever you read from a keyboard.

eg `fscanf(stdin, "%d", &a);`
⇓ is equivalent to
`scanf("%d", &a);`

Stdout → `fprintf(stdout, "%d", a);` ⇓ `printf("%d", a);` — all outputs will go to this file by default.

Stderr → All errors will go to this file.

`fprintf(stderr, "Cannot open file");`

Command Line Arguments

Command Line → Like DOS Prompt

C:\> _ _ _ _ _ _ _ _

D:\> _ _ _ _ _ _ _ _

C:\> Documents and Settings\ Desktop> _ _ _ _ _

This is Command line window, where you can type the command.

- These arguments helps you to create your own commands, own Operating System. (Text-based).
- These arguments will be given in main() function.
- 2 types of arguments

argc argv

argc → It counts the no. of parameters passed from command line.

argv[] → It is a pointer to array of strings.

i.e. All the arguments will be treated as strings. and argv[] is collection of that strings

Eg: C:\> copy (1) a.txt (2) b.txt (3)

argc = 3 (No of parameters)

argv[0] "copy" argv[2] "b.txt"
argv[1] "a.txt"

Need of argc and argv

To execute the file copy program, we are required to first type the program, compile it and then execute it. But with these arguments:

(1) There is no need to compile the program every time to copy a file. Means programs must be executable at command prompt.

(2) ~~[illegible] program~~ In file copy program, we have to change source and target file names everytime we want to copy different files and we have to recompile the program after changing. In this, we can supply file names at command prompt.

Eg

```
void main()
{
  FILE *fs, *ft;
  char ch;
  fs = fopen("a.txt", "r");
  ft = fopen("b.txt", "w");
  while ((ch = fgetc(fs)) != EOF)
  {
    fputc(ch, ft);
  }
  fclose(fs);
  fclose(ft);
}
```

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

```
main (int argc, char *argv[])
FILE *fs, *ft;
char ch;
fs= fopen (argv[1], "r");
ft= fopen (argv[2], "w");
if (argc != 3)
{
printf ("Too many or few arguments");
exit (0);
}
while ((ch= fgetc (fs)) != EOF)
{
fputc (ch, ft);
}
fclose (fs);
fclose (ft);
}
```

Save this program say filecopy.c
This filename will become your command.

Explanation:- open Dos prompt (goto that folder where your programs reside)

C:\> cd tc>
C:\> cd tc> cd bin\
C:\tc\bin> ... Here you type the command

code: C:\tc\bin> filecopy a.txt b.txt

argv[0] will be command, will call that main() function.
argv[1] = "a.txt", argv[2] = "b.txt"

Next time, when you want to copy other 2 files, you don't need to change the program. Simply give the command

C:\tc\bin> filecopy c.txt d.txt

Automatically file will be copied from c.txt to d.txt at the same location (2 copies will be there.)

Assignment

Q1 WAP to count number of vowels in a text file.

Q2 WAP to append a binary file using command line arguments.