B.C.A. (Sem – II)

B.C.A. - 201

Advanced Programming Language 'C'

Purushottam Singh

Unit:- 4

* What is File?

A File is a Place on the disk where A group of related data is stored.

Operation of File

1) Naming of File
2) Opening a File
3) Reading data from a File
4) writing data to a File
5) closing a file.

* How to open a File or declare a File

To create a file on disk we required 3 thinks.

1) File name
2) mode of file
3) Related file.

Date __/__/__
Page _____

→ 1) File name :-
It is a string of characters that makeup a valid file name.

→ 2) Mode of file :-
There are 3 types.
1) Read :-
Read data from a file.

2) write :-
write a data to a file.
3) Read & write.

→ 3). Related data :-
we can store related data to a file.

* Syntax of opening the file or declaring the file :-

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

file is a data structure that' defined in input output library.
fp is pointer variable and it is an identifier to the file type *fp.
Mode is a type which operation are performed on file.

Date __/__/__
Page _____

r :- open the file for Reading only.

w :- open the file for writing only.

a :- open the file for appending or adding the data to file.

r+ :- The existing file is open to the beginning for both.

w+ :- The existing file is open to the beginning for both.

a+ :- open the file for appending or adding the data to both file.

Example

```
FILE *FP1, *FP2;
FP1=Fopen("File1.txt", "r");
FP2 = Fopen("File2.txt", "w");
```

we can open and use a number of files at a time. this number depends on the system.

Closing a File

A file must be closed as soon as all operations on it have been completed.

Date ____/____/____
Page _____

Syntax

fclose(File Pointer)

ex

```
FILE *FP;
FP = fopen("D:\data", "r");
fclose(FP);
```

=) Input output operation of file :-

1) getc
2) Putc

The simplace file input output function are getc(), Putc()e

Syntax

1) getc :-

```
c = getc(File Pointer);
```

2) Putc :-

```
Putc(c, fp);
```

fprintf() and fscanf()

-) fprintf() :-

use same as Printf and scanf.

Date ___/___/___
Page _____

Syntax

```
FPrintF(FP, "control string", arg list);
FScanF(FP, "control string", &arg list);
```

Example

```
FPrintF(FP, "%d", i);
FScanF(FP, "%d", &i);
```

* How to handling error in a file.

Error means some mistake done by user as a logical or others.

1) To Read Beyond end of file.
2) device overflow
3) To use a file that has not been open.
4) To Perform operation on different mode.

Date ____/____/____
Page _____

5) To open a file with invalid file name.
6) To try a write protected file.

* File Function

* Random access to file

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

1) ftell();
- given the current position in file.
- It takes file pointer and return a number of type is long, that corresp. to a current position.
- when the operation is successful ftell returns a zero.
- when the operation is not successful ftell returns a minus one (-1).

Syntax

number = ftell(fp);

2) fseek();
- It is used to move the file position to a desired location within the file.

- when the operation is successful fseek returns a zero.
- when the operation is not successful fseek returns minus one (-1).

Syntax

fseek (FP, offset, position);

where FP = file pointer

offset :- offset is a number or variable of type long.

position :- position is an integer number.

value	meaning
0	beginning of the file
1	current position
2	end of file

example

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *FP;
        long int n;
        char c;
        FP = FOpen("RANDOM","w");
        while((c=getchar()) != EOF)
        {
                put(c,FP);
        }
```

```
        printf("No. of characters entered = %ld\n"
                                ,ftell(fp));
        fclose(fp);
        fp = fopen("RANDOM","r");
        n = 0L;
        while(feof(fp) == 0)
        {
            fseek(fp,n,0);
            printf("Position of %c is %ld\n",
                        getc(fp),ftell(fp));
            n = n+5L;
        }
        putchar('\n');
        fseek(fp,-1L,2);
        do
        {
            putchar(getc(fp));
        }
        while(!fseek(fp,-2L,1));
        fclose(fp);
}
```

3) Rewind();

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

When the operation is successful rewind returns a zero value.

When the operation is not successful rewind returns a minus one (-1).

Date ___/___/___
Page _____

syntax

```
rewind(File Pointer);
```

example

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *p;
    char s;
    clrscr();
    p=fopen("input.txt" "w+");
    fprintf(p,"Ami Patel");
    rewind(p);
    fscanf(p,"%c",s);
    printf("The first character is %c",s);
    fclose(p);
    getch();
}
```

=) Command line Argument [CLA]

It is a Program but it requires a Parameter whenever you execute a Program.

A Parameter may be File, variable or any other item.

Date __/__/__
Page _____

syntax

```
void main(int argc, char *argv[])
{
    ..........
    ..........
}
```

- where argc is an argument counter that counts the number of arguments on command line.
- argv[] is an argument vector and represents an array of character pointer that points to the command line argument.
- The size of this array will be equal to the value of argument counter.

example.

```
#include<stdio.h>
#include<conio.h>
void main(int argc, char *argv[])
{
    FILE *fp;
    int i;
    char word[15];
    fp = fopen(argv[1], "w");
    printf("\n No. of arguments in comm
                line = %d\n\n", argc);
    for(i=2; i<argc; i++)
    {
```

```
        fprintf(fp,"%s",argv[i]);
    }
    fclose(fp);
    printf("contents of %s file\n\n",argv[1]);
    fp = fopen(argv[1],"r");
    for(i=2;i<argc;i++)
    {
        fscanf(fp,"%s",word);
        printf("%s",word);
    }
    fclose(fp);
    printf("\n\n");
    for(i=0;i<argc;i++)
    {
        printf("%*s\n",i*5,argv[i]);
    }
    getch();
}
```

Date __/__/__
Page _____

Dynamic memory allocation

Introduction

Dynamic data structure Provide flexibility in adding, deleting or Rearranging the data items at run time.

Dynamic memory management techniques permit us to allocate additional memory space or to release unwanted space at run time thus optimising the use of storage space.

Example

linked list

C language requires the number of elements in an array to be specified at compile time but we may not to be able to do so always.

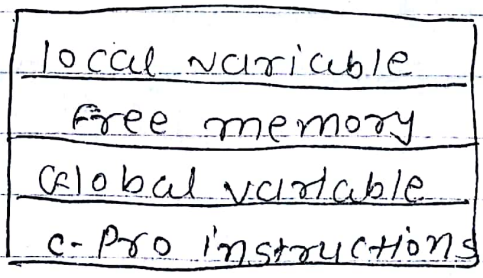
our initial judgement of size if it is wrong, may cause failure of the Program or wastage of memory space.

Many languages permite a Programmer to specify and array size at run time. such languages have the ability to calculate and assign during execution the memory space required by the variables in a Program the process of allocating memory at run time is known as dynamic memory allocation.

Function	Meaning
malloc.	Allocate request size of bytes and returns pointer to the first byte of the allocated space.
calloc	Allocates space for an array of element initializes them to zero and then returns a pointer to the memory.
free	Free previously allocated space.
realloc	Modifies the size of previously allocated space.

* Memory allocation Process

Memory allocation process associated with C Program. Figure shows the conceptual view of storage of a program in memory.



Storage of a C Program

The program instructions and global and static variables are stored in a region known as Permant storage area and the local variables are stored in another area ; is called stack.

The memory space that is located between this two reason is available for dynamic allocation during execution of the program this free memory region is called Heap

The size of the Heap keeps changing when program is executed due to creation and dute of variables that are local to function.

when memory allocation Process fail to locate the not enough memory space at that time return a null Pointer.

* allocating a block of memory

malloc();

A block of memory may be allocated using the function malloc. The malloc() reserve a block of memory of specify size and return a Pointer of type void. This means that we can assign it to any type of Pointer.

ptr = realloc (Ptr, new size);

This function allocates a new memory space of size new size. to the pointer variable ptr and returns a pointer to the first byte of the new memory block the new size may be larger or smaller than previously defined size.

remember that the new memory block may or may not be at the same place as the old one.

If the function is unsuccessful in allocating the space it returns null pointer and the original block is free (lost).

Date __/__/__
Page _____

The Preprocessor

Introduction

C is a unix language we have already seen features such as Structure and Pointers and another unix features of the C language is the Preprocessor.

The C Preprocessor Provides several tools that are unavailable in other high level language.

The programmer can use this tools to make is program easy to read easy to modify portable and more efficient.

The preProcessor as its name implies is a Program that Processes the source code before it passes through the compiler.

It operates under the control of what is known as PreProcessor command lines or directives.

Preprocessor directive are place in the source program before the main line.

directive	meaning
- #define	define a macro substitution.
- #undef	undefine a macro

Date __/__/__
Page _____

- #include	Specify the files to be included
- #if def	Test of a macro definition
- #endif	Specifies the end of #if
- # ifndef	Test whether macro is not defined
- #if	Test a compile time condition
- #else	Specify alternative when #if test fail.

=) The directive can be define in three categories :

1) macro substitution directives
2) file inclusion directives
3) compiler control directives

1) Macro substitution directives :-
macro substitution is process when identifier in a program is replaced by a predefined string composed of one or more Tokens.

The preprocessor accomplishes this task under the direction of #define statement and this statement known as macro definition.

Syntax:-

#define identifier string.

The keyword #define is written and followed by the identifier and a string with atlist one blank space between them.

The macro definition is not terminated by a semicolon.

There are 3 types of macro substitution.

1) simple macro substitution
2) Argumented macro substitution
3) Nested macro substitution

1) simple macro substitution :-

simple string replacement is commanly used to define constant.

Ex

```
#define PI 3.14

#define MS
TOTal = M*value;
PrintF("M = %d", M);
```

This two lines would be change during preprocessor as follow

2) Argumented macro substitution :-

The preprocessor permites us to define more complex and more useful form of replacements.

Syntax

```
#define identifier (f1, f2, ....fn)
                                String ;
```

The identifier f1, f2-- fn are the formal macro arguments.

* the basic difference between simple macro replacement and replacement of macro with Argument.

- subsequent occurence macro with Argument is known as macro call.
- when a macro is call the processor of the string replacing the formal parameters with the actual parameters.

Ex `#define cube (x * x * x)`

Date ___/___/___
Page _____

3) Nesting of macro :-

We can also use the one macro in definition another macro.

```
#define M 5
#define N (M+1)
```

4) undefined macro

A defined macro can be undefined using the stutement

```
#undef identifier
#undef m
```

This is useful when we want to restrict the definition only to a Perticular Part of the Program.

=)(2) File inclusion directives :-

An external file containing the function or macro definition can be included as a part of program so we need not rewrite macro definition

#include "file name";

where filename is the name of file containing the required definition.

when the file name is included within double quotation mark the search for the file is made first in the current directiorary

#include "sum.c"