1. Structure

1.1 Basics of structure

Definition: Structure is a collection of one or more variables of same or different data types grouped together under a single name for easy handling. Structure is a user defined data type that can store related information about an entity. Structure is nothing but records about particular entity.

Declaration of Structure: Structure is declared using keyword struct followed by structure name and variables are declared within structure.

Syntax: struct structure_name
```
{
    data _type  var_name;
    data_type var_name;      structure members
};
```
Example:
```
struct employee
{
   int employee_number;
   char  employee_name[20];
   int  employee_age;
   float  employee_salary;
};
```
structure members

Initialization of structure: Initialization of structure is done after declaration.
Syntax: struct structure_name structure_variable_name;
Example: struct employee emp1;

Structure tag, structure elements, structure variables, structure data type:
Consider example
```
struct employee
{
   char empname[20];
    int empnum;
};
```
Structure template with tag
Here employee is the tag

```
struct employee emp1    /*emp1 is structure variable*/
```

Structure tag defines a template for variables which will be used later. It does not allocate memory. Example: employee
Structure variables are the variables using which members of structure are accessed with the help of dot operator. C allocates memory for structure variable. Example: emp1
Structure members are the variables within the struct. These are also called as elements of structure. Example: empname[20], empnum;
Structure data type: To define a structure, we must use the **struct** keyword. The struct defines a new data type.
Syntax: struct structure_name structure_variable_name;
Example:
struct employee

Scanned with OKEN Scanner

```
{
    int employee_number;
    char employee_name[20];
    int employee_age;
    float employee_salary;
};
struct employee emp1;
```

structure members

Accessing the members of structure:
Using structure_variable_name and dot (.) operator we can access the members of the structure as structure_variable_name.member_name
In the above example, emp1 is structure_variable_name

```
employee_number
employee _name
employee _age
employee _salary
```

member_name

Now we can assign values as
emp1.employee_number=23;
emp1.employee_name= "nanjesh";
emp1.employee_age =25;
emp1.employee_salary =25,000.52;

The above values can be read dynamically using scanf( ) function as below:
scanf(" %d", &emp1. employee_number);
scanf(" %s", &emp1. employee_name);
scanf(" %d", &emp1. employee_age);
scanf(" %f", &emp1. employee_salary);

The values can be displayed or printed using printf( ) function as below:
printf("employee number is %d\n", emp1.employee_number);
printf("employee name is %s\n", emp1.employee_name);
printf("employee age is %d\n", emp1.employee_age);
printf("employee salary is %f\n", emp1.employee_salary);

The complete program to show structure declaration, initialization, assigning value, reading and displaying values in structure can be written as below:

/*Write a 'C' program to read an employee number, name, age, salary and print details using structure.*/

```
#include<stdio.h>
#include<conio.h>
struct employee
{
   int empnumber ;
   char empname[20];
   int empage;
   float empsalary;
```


Scanned with OKEN Scanner

```
};
void main( )
{
        struct employee emp1;
        clrscr ( );
        printf ("enter employee number \n");
        scanf ("%d", &emp1.empnumber);
        printf ("enter employee name \n");
        scanf ("%s", emp1.empname);
        printf ("enter employee age \n");
        scanf ("%d", &emp1.empage);
        printf ("enter employee salary \n");
        scanf ("%f", &emp1.empsalary);
        printf ("employee details are  \n");
        printf(employee number is %d \n", emp1.empnumber);
        printf(employee name is %s \n", emp1.empname);
        printf(employee age  is %d \n", emp1.empage);
        printf(employee salary is %f \n", emp1.empsalary);
        getch( );
}
```

Output:

```
enter employee number
12
enter employee name
nanjesh
enter employee age
25
enter employee salary
28000
employee details are
employee number is 12
employee name is nanjesh
employee age is 25
employee salary is 28000.000000
```

1.2. Arrays of Structure (explained with example)

As we studied in basics of structure, considering employee example

```
struct employee
{
        int empnum;
        char empname[20];
        float empsalary;
};
```

structure members

```
struct employee emp;
```

structure_name structure_variable_name

In the above example we can read only one employee details using structure variable name emp

Scanned with OKEN Scanner

as emp.empnum, emp.empname, emp.empsalary. If we want to read more than one employee details using same structure, then structure variable name has to be declared as an array emp[20]
that is:
struct employee emp[20];
Now, emp[0].empnum, emp[0].empname, emp[0].empsalary stores details of first employee, and
emp [1].empnum, emp[1].empname, emp[1].empsalary stores details of second employee and so on. This reading and displaying details of each employee is done using for loop and generalized as emp[i];

/*Write a C program to read 3 employees details and display using structure concept. (number, name, salary)*/

```
#include<stdio.h>
#include<conio.h>
struct employee
{
   int empnum ;
   char  empname[20];
    float  empsal;
};
void main( )
{
          struct employee emp[10];
          int n, i;
          printf ("enter number of employees \n");
          scanf ("%d", &n);                                   /* here n=3 */
          printf ("enter %d employees details\n",n);
          for (i=0; i<n; i++)
          {
                printf ("enter employee %d details  \n" i+1);
                printf ("enter the emp number  \n");
                scanf ("%d", &emp[i].empnum);
                printf ("enter the employee  name  \n");
                scanf ("%s", emp[i].empname);
                printf ("enter the employee salary   \n");
                scanf ("%f", &emp[i].empsal);
          }
          printf(employee  details are  \n", );
          for (i=0; i<n; i++)
          {
                printf(details of employee %d are \n", i+1);
                printf(employee number   is %d', emp[i].empnum);
                printf("employee name is %s", emp[i].empname );
                printf ("employee salary is %f", emp[i].empsalary);
          }
          getch( );
}
```

Output:
Enter number of employees

Scanned with OKEN Scanner

```
3
Enter 3 employees details
Enter employee 1 details
Enter emp number
11
Enter employee name
Nanjesh
Enter employee salary
2500
Enter employee 2 details
Enter emp number
12
Enter employee name
mansi
Enter employee salary
26000
Enter employee 3 details
Enter emp number
13
Enter employee name
raksha
Enter employee salary
28000
details of employee 1 are
employee number is 11
employee name is nanjesh
employee salary is 25000
details of employee 2 are
employee number is 12
employee name is mansi
employee salary is 26000
details of employee 3 are
employee number is 13
employee name is raksha
employee salary is 28000
```

1.3. Nested structure (Structure within a structure)

Nested structure is a structure that contains another structure as its member.

syntax:

```
struct  structure1
{
        data_type structure1 _member_name;
        .
        .
};
struct structure2
{
       data_type structure2_member_name;
       .
```


Scanned with OKEN Scanner

```
    struct structure1 structure1_variable_name;
};
```

Example:
Here structure2 is the main structure within which structure1 is used as a member of it.

```
struct student _dob
{
    int day;
    int month;
    int year;
};
struct student
{
    int studnum;
    char name [20];
    struct student_dob  date;
};
struct student  stud1;
```

Structure variable of one structure is initialized within another structure. Here structure variable of structure1 student_dob is date which is initialized within another structure student. if we want to access members of student_dobs from student structure then we need to use:

structure2_variable_name.structure1_variable_name.structure1_member_name

```
stud1. date. month
stud1. date. year
stud1. date. day
```

Where: stud1 is structure2 variable name
date is structure1 variable name which is initialized within structure2.
month, year, day are the structure1 members

Example: /*C program to read and display information of a student using structure within a structure or nested structure*/

```
#include<stdio.h>
 #include<conio.h>
struct stud_dob
{
    int day;
    int month;
    int year;
};
struct student
{
     int studnum;
     char name[20];
     struct student_dob  date;
};
void main( )
{
```


Scanned with OKEN Scanner

```
        struct student  stud1;
        printf(" enter the student number\n");
        scanf ("%d", &stud1.studnum);
        printf(" enter the student name\n");
        scanf ("%s", stud1.name);
        printf("enter date of birth as day, month, year");
        scanf( "%d%d%d", &stud1. date.day, &stud1.date.month, &stud1.date.year);
        printf(" student details are \n");
        printf ("student number is %d \n", stud1.studnum);
        printf ("student name is % s \n",stud1.name);
        printf("student dob is %d | %d | %d\n", stud1.date.day,stud1.date.month,
              stud1.date.year);
        getch( );
}
```

Output:

```
Emter the student number
12
Enter the student name
Nanjesh
Enter date of birth as day, month, year
7
6
1990
Student details are
Student number is 12
Student name is nanjesh
Student dob is 7 |  6 | 1990
```

1.4. Structure and Functions

Structure can be passed to functions and returned from it. Function can access members of structure in three ways, that is, **passing structure to functions can be done in following three ways.**

- passing individual members
- passing entire structure or structure variable
- passing address of structure

Passing individual members: while calling a function from main, in the place of actual parameters we can use or pass structure members as structure_variable_name.structure_member_name

Example program:

```
#include <stdio.h>
void add (int a, int b);
struct addition
{
    int a ;
    int b;
};
void main( )
```


Scanned with OKEN Scanner

```
{
    struct addition p;
    p.a=10;
    p.b=20;
    add (p.a, p.b);
    getch( );
}

void add (int a, int b)
{
      int sum;
      sum = a+b;
      printf(" sum is %d", sum);
}
```

Output:
sum is 30

Passing the entire structure or structure variable: Instead of passing individual members of structure, entire structure is passed as actual parameter while calling a function. Here the structure variable is passed to the function.

Example program:

```
#include <stdio.h>
struct addition
{
   int a;
   int b;
};
void add (struct addition p);  /* here function declaration is done after structure declaration*/
void main( )
{
      struct addition p;
      p.a=10;
      p.b=20;
      add (p);   /* structure variable p is passed*/
      getch( );
}
void add(struct addition p)
{
     int sum;
     sum=p.a+p.b;
     printf("sum is %d\n", sum);
}
```

Output:
sum is 30

Passing structure through pointers: Structure can be passed to a function using pointers.
Syntax: struct structure_name structure_variable_name, *structure_pointer_name;
Example: struct student stud, *ptr;
Then we need to assign, structure_pointer_name= &structure_variable_name;

Scanned with OKEN Scanner

Example: ptr=&stud;
Accessisng can be done as: structure_pointer_name → structure_member_name;
Example: ptr→name;

Example program:

```
#include<stdio.h>
#include<conio.h>
struct student
{
        int number;
        char name [20];
};
void main( )
{
        struct student stud , *ptr;
        ptr=&stud;
        printf (" enter number and name of student \n");
        scanf("%d %s", &ptr→number , ptr→name);
        printf("student name is %s\n", ptr→name);
        printf("student number is %d\n",ptr→number);
        getch( );
}
```

Output:
Enter number and name of student
12
Nanjesh
Student name is nanjesh
Student number is 12

1.5. Type definition

The C programming language provides typedef keyword which allows the user to create a new data type. typedef does not occupy memory
Syntax: typedef existing_data_type new_data_type;
Example: typedef int Integer;
Now instead of int a=10, we can declare as Integer a=10; which perform same as int. Similarly we can apply for structure as shown below:

```
   /*without typedef */
struct employee
{
        char empname[10];
        int empnum;
};
struct employee emp
```

```
    /*with typedef*/
typedef struct employee
{
        char empname[10];
        int empnum;
};
employee emp;
```

Here **employee** itself acts as data type which is the structure type declared using **typedef**. Hence, no need of writing **struct** while declaring structure variable **emp**.

Scanned with OKEN Scanner

1.6. Programming examples

Example 1: C program to read and display information about student (usn, name, percentage, sem)

```
#include<stdio.h>
#include<conio.h>
struct student
{
   int studentusn ;
   char studentname[20];
   int studentsem;
   float percentage;
};
void main( )
{
        struct student std1;
        clrscr ( );
        printf ("enter student number \n");
        scanf ("%d", &std1.studentusn);
        printf ("enter student name \n");
        scanf ("%s", std1.studentname);
        printf ("enter student sem\n");
        scanf ("%d", &std1.studentsem);
        printf ("enter student percentage \n");
        scanf ("%d", &std1.percentage);
        printf ("student details are  \n");
        printf(student number is %d \n", std1.studentusn);
        printf(student name is %s \n", std1.studentname);
        printf(student sem  is %d \n", std1.studentsem);
        printf(student percentage is %f \n", std1.percentage);
        getch( );
}
```

Output:

```
enter student number
12
enter student name
nanjesh
enter student sem
2
enter student percentage
78.21
student details are
student number is 12
student name is nanjesh
student sem is 2
student percentage is 78.210000
```


Scanned with OKEN Scanner

Example 2: C program to find sum of two numbers using structure.

```
#include<stdio.h>
#include<conio.h>
struct add
{
      int a, b, sum;
};
void main( )
{
      struct add p;
        printf ("enter the values of a and b \n");
        scanf ("%d%d", &p.a, &p.b);                    /*where p is structure variable name */
        p.sum = p.a + p.b;
        printf (" sum is % d \n", p.sum);
        getch( ) ;
}
```

Output:

```
enter the values of a and b
5          10
sum is 15
```

Example 3: Write a program to find biggest of 3 numbers using structure.

```
#include<stdio.h>
#include<conio.h>
struct findbig
{
     int a, b, c, big ;
};
void main( )
{
        struct findbig var;
        printf ("enter the values of a,b,c \n");
        scanf ("%d %d %d ", &var.a,&var.b,&var.c);
        if (var.a > var.b && var.a> var.c)
                              printf (" a = %d is big \n", var.a);
       else if (var.b>var.a && var.b.>var.c)
                              printf (" b=%d is big \n", var.b);
        else
                             printf ("c = %d is big /n", var.c.);
        getch( );
}
```

Output:

```
enter the values of a,b,c
11   5 · 81
c=81 is big
```


Scanned with OKEN Scanner

Example 4: C program to maintain a record of n student details using an array of structures with four fields (Roll number, Name, Marks, and Grade). Assume appropriate data type for each field. print the marks of the student, given the student name as input.

Refer Lab program 13, Page number:224

Questions: Write a C program to read and display information about a student (usn, name, percentage, sem) using structure.
Write a note on:

- structure tag
- structure elements
- structure variables
- structure data type

Explain the concept of arrays of Structure with an example program.
Explain the concept of Nested structure or Structure with in a structure with example.
Explain the different ways of passing structure to functions.

2. Files

File definition: File is a collection of data stored on a secondary storage device.
Types of files:

Text files : (ASCII based) Contains stream of characters processed sequentially.
Binary files : Contains any type of data encoded in binary form.

2.1. Opening and closing of file

Following are the steps to use a file in C:

- declare a file pointers variable
- open a file
- process the file
- close the file.

Declaring a file pointer: Before opening a file we need a pointer that points to a file, declaration of a file pointer variable is as below:
FILE *file_pointer_name;
Example: FILE *fp;

Opening a file: A file must be opened before data can be read from or written to it. fopen( ) function is used to open a file.
Syntax: FILE *fopen ("filename.fileextension", "file_mode");
Example: FILE *fopen("file1.txt","r");
Where:

- If we specify directly file name then it means file is in current directory or else we need to specify the path of the file.
- File types or extensions: filename.txt, filename.dat, filename.out are the generally used file types.

Scanned with OKEN Scanner

- File mode: mode in which the file has to be opened
  - r: open a file for reading, if file does not exist then error message is generated.
  - w: open a file for writing, if file does not exist then it creates a file.
  - a: append to a text file, if files does not exist then file is created.
  - Similarly rb,wb,ab for binary files.

Closing a file: To close an opened file, the fclose( ) function is used.
Syntax: fclose(file_pointer_name); or
fclose(FILE *stream);
Example: fclose(fp);

2.2. Input and Output operations of file

Reading Data from files
C provides following functions to read data from a file

- fscanf ( )
- fgetc( )
- getw( )
- fgets( )
- fread( )

fscanf( ):

fscanf( ) function is used to read formatted data from the stream. It is similar to scanf except the filepointername. The file must be opened in read mode to scan its contents.
Syntax:
fscanf(FILE *stream, "controlcharacter",variable address list);
or
fscanf (file_poiner_name, "control_character", variables address list);
Example:

```
#include<stdio.h>
void main( )
{
    FILE *fp;
    char name[10];
    fp=fopen("filename", "r");
    fscanf(fp, "%s",name);
    printf("%s", name);
    fclose (fp);
}
```

Output: It reads name from the file filename.txt and prints on output screen.

fgetc( ): file get character

fgetc( ) function returns the character from the stream and EOF if the end of file is reached . File must be opened in read mode. Using fgetc(ch) in for loop we can get all the contents present in a file. The file must be opened in read mode to scan its contents.
Syntax : fgetc(file_pointer_name);
or
fgetc (FILE * stream);

Scanned with OKEN Scanner

Example:

```
#include<stdio.h>
void main( )
{
    FILE * fp;
    char ch;
    fp=fopen ("filename. txt", "r");
    ch = fgetc (fp);
    printf (" %c", ch);
    fclose(fp);
}
```

Output: It reads a character from the file filename.txt and prints on output screen.

fgets( ): file get string

fgets( ) used to get a string from a stream. The file must be opened in read mode to scan its contents.

Syntax: fgets(char *str, int n, FILE * stream);

Where: str is the pointer to an array of characters where string is stored.
n is the maximum number of characters (including end null character) to be read.
stream is file pointer.

Example:

```
#include<stdio.h>
void main( )
{
    FILE * fp;
    char str[100];
    fp=fopen ("filename.txt", "r");
    fgets(str, 60, fp);
    puts(str);
    fclose(fp);
}
```

It reads first 60 characters from file and copies to str and it is printed using puts( ). If EOF end of file is reached and no characters are read, the contents of str remain unchanged and null pointer is returned.

getw( ):

getw( ) is an integer oriented function. It is used to read integer values. It is similar to **getc** function and this function would be useful when we deal with only integer data. The file must be opened in read mode to scan its contents.

Syntax: getw (FILE * stream);
or
getw(file_pointer_name);

Example:

```
#include<stdio.h>
void main( )
{
  FILE * fp;
  int number;
```


Scanned with OKEN Scanner

```
    fp=fopen ("filename. txt", "r");
    number =getw(fp);
    printf ("%d", number);
    fclose(fp);
}
```

It reads a number stored in a file and printed using printf (note: number must be present in a file).

fread( ):

fread( ) function is used to read data from a file. The file must be opened in read mode to scan its contents.

Syntax : int fread(void *str, size_t size, size_t num, FILE *stream);

where str: pointer to a memory block with minimum size of size*num bytes.

size : size in bytes of each element to be read

num: number of elements of size "size",

stream : file pointer.

Example :

```
#include<stdio.h>
void main( )
{
    FILE * fp;
    char str[20];
    fp=fopen("filename. txt", "r");
    fread(str,1,10,fp);
    printf (" first 10 character of file are %s", str);
    fclose(fp);
}
```

It opens file filename.txt and reads first 10 characters of file which are of 1 byte size and printed using printf.

Writing Data to files

C provides following functions to write data into a file:

- fprintf( )
- fputc( )
- putw( )
- fputs( )
- fwrite( )

fprintf( ):

fprintf( ) function is used to write formatted output to stream. The file must be opened in write mode to add contents to it.

syntax:

fprintf(FILE * stream , "control characters", list of variables);

or

fprintf (file pointer name, " control characters", list of variables) ;

Example :

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


Scanned with OKEN Scanner

```
void main( )
{
    FILE * fp;
    char name[10]= "nanjesh";
    fp=fopen("filename. txt", "w");
    fprintf (fp, " Name is %s" , name);
    fclose(fp);
}
```

Output:
It writes **Name is nanjesh** into file filename.txt.

fputc( ): file put character ⇒ putc()

fputc( ) function writes the character to a stream, The file must be opened in write mode to add contents to it.

Syntax:

fputc(int char , FILE * stream) ✓

where char is the character to be written

stream is file pointer

It return character to be written and return EOF if error is occurred.

Example:

```
void main( )
{
    FILE * fp;
    char n;
    fp=fopen ("filename. txt", "w");
    n='9';
    fputc(n, fp);
    fclose(fp);
}
```

Output:
Writes character 9 to file filename txt.

fputc('A', fp)
⇒ A is a char but it is automatically promote to int with an ASCII value of 65.

fputs( ):

fputs( ) function writes a string to the stream but excluding null character present at the end of string. The file must be opened in write mode to add contents to it.

syntax:

int fputs(const char *str, FILE * stream);

where str: Array containing null terminated sequence of character to be written

stream: File pointer.

It returns a non-negative value else it returns EOF,

Example:

```
#include<stdio.h>
void main( )
{
    FILE * fp;
    fp=fopen ("filename. txt", "w");
    fputs( " This is PCDS",  fp);
```


Scanned with OKEN Scanner

```
    fclose(fp);
    getch( );
}
```

Output: It writes This is PCDS into file filename txt

putw( ):

putw( ) is an integer oriented function. It is used to write integer values. It is similar to **putc** function and this function would be useful when we deal with only integer data. The file must be opened in write mode to add contents to it.

Syntax: putw(integer, fp);

Example :

```
#include<stdio.h>
void main( )
{
    FILE * fp;
    int number=10;
    fp=fopen ("filename. txt", "w");
    putw( number, fp);
    fclose(fp);
}
```

Output: writes number value 10 to file filename.txt

fwrite( ):

fwrite( ) function is used to write data to file. The file must be opened in write mode to add contents to it.

Syntax:- int fwrite (void *str, size_t size, size_t.num, FILE *stream);

where

str: pointer to array of elements to be written
size: size in bytes of each element to be written
num: number of elements which of size one byte each
stream: file pointer

Example :

```
#include<stdio.h>
void main( )
{
    FILE * fp;
    char str[20] = "Nanjesh";
    fp=fopen ("filename. txt", "w");
    fwrite(str,1, sizeof(str),fp);
    fclose(fp);
}
```

Output: It excludes null character while printing.

sizeof (str) is used because it has to write all the characters in string.

Scanned with OKEN Scanner

2.3. Functions for selecting records randomly in a file

C provides following functions for selecting records randomly in a file

- fseek ( )
- ftell ( )
- fgetpos ( )
- rewind ( )
- fsetpos( )
- feof( )

fseek ( ):

fseek( ) is used to set the file position pointer for the given stream to given offset.

Syntax:
int fseek (FILE *stream, long int offset, int where);
where: stream : File pointer
offset: Number of bytes to offset from where
where: position from where offset is added.
where can be
SEEK_SET: Move forward **m** bytes in file from the start of the file.
fseek(fp,m,SEEK_SET);
SEEK_CUR: Move forward or backward m bytes in file from the current location.
fseek (fp, m, SEEK_CUR); /* moves m bytes forward from current location*/
fseek (fp, -m, SEEK_CUR); /* moves m bytes backward from current location*/

SEEK_END: Move backwards m bytes from end of file.
fseek (fp, -m, SEEK_END);
fseek( ) returns zero if it is successful , else its returns non zero value.

Example:
fseek(fp,5,SEEK_SET)
It moves 5 bytes forward from start location in a file.
fseek (fp, -2, SEEK_CUR)
It moves 2 bytes backward from current location in a file
fseek (fp, -2, SEEK_END)
It moves 2 bytes backward from end of file.
fseek (fp, 0, SEEK_END)
It remains in end of file position

Example program:

```
#include<stdio.h>
void main( )
{
        FILE *fp;
        fp=fopen ("filename.txt", "w");
        fputs ("Karnataka_is_in_India",fp);
        fseek (fp,9, SEEK_SET);
        fputs ("Bengaluru_is_in_India",fp);
        fclose (fp);
}
```


Scanned with OKEN Scanner

Output:
0 1 2 3 4 5 6 7 8 9
K a r n a t a k a is_in_India

writing head is placed at 9th position from first then overwriting from 9th position Bengaluru_is_in_India
Final output: KarnatakaBengaluru_is_in_India

ftell( ):

ftell( ) returns the current file pointer position from which next I/O operation is performed.
Syntax: long int ftell (FILE * stream)
ftell returns current value of position indicator if successful or else -1.
Example: len =ftell(fp);
here fp is file pointer name,
len gives the size (in bytes) of file pointed by fp if fp is placed at end of files.

```
void main( )
{
        FILE *fp;
        int len;
        fp=fopen ("filename.txt", "r");
        fseek(fp, 0, SEEK_END);        /*placing file pointer at end of file*/
        len = ftell(fp);
        fclose (fp);
        printf("Total size of file=%d bytes\n",len);
        getch( ) ;
}
```

Output: if file is having word nanjesh in it then it gives output as
Total size of file=7 bytes since 7 characters each of 1 byte is counted.

fgetpos( ) and fsetpos( ):

fgetpos( ): It is used to determine the current position of the stream
syntax: int fgetpos (FILE * stream ,fpos_t *pos);
where: stream is file pointer
fpos_t is defined in stdio.h,
variable pos declared with fpos_t stores the position information
It returns zero and nonzero if unsuccessful.

fsetpos( ): It is used to move the file pointer to the position indicated by pos variable of fgetpos( );
syntax : int fsetpos (FILE * stream, fpos_t *pos);
Example program for fgetpos ( ) & fsetpos ():

```
#include <stdio.h>
void main( )
{
        FILE * fp;
        fpos_t position;
```


Scanned with OKEN Scanner

```
fp=fopen ("filename.txt","w");
fgetpos(fp, &position);
fputs("nanjesh",fp);
fsetpos (fp, &position);
fputs("bit is in davanagere",fp);
fclose (fp);
}
```

/*gets the file pointer position that is zero at the initial stage*/

/*Writes content to file and now position is 7*/

/*But position variable is having value zero, now fsetpos sets to the value present in position variable that is zero*/
/*Now if we use fputs for writing, then it starts from first position over writing existing content*/

Output: filename.txt file will be having content: **bit is in davanagere**

rewind( ):

rewind( ) function is used to adjust the position of file pointer so that next I/O operation take place at the beginning of file.

Syntax: void rewind (FILE *stream)

It is similar to fseek (FILE *stream, 0, SEEK_SET);

Example:

```
void main( )
{
    FILE * fp;
    fp=fopen ("filename.txt","w");
    fputs("ait is in ckm",fp);
    rewind(fp);
    fputs("bit is in dvg",fp);
    fclose (fp);
}
```

Output:

filename .txt

ait is in ckm

Over writes from beginning position changed to beginning before writing when rewind( ) is used

filename .txt

bit is in dvg

feof ( ):

feof( ) function tests the end of file indicator for given stream.

Syntax: int feof (FILE *stream);

where stream: file pointer name.

This function returns a non zero value when EOF is reached or else it returns zero.

Example:

```
#include <stdio.h>
void main( )
{
    FILE * fp1;
    char ch;
```


Scanned with OKEN Scanner

```
fp=fopen ("filename.txt","w");
fgetpos(fp, &position);
fputs("nanjesh",fp);
fsetpos (fp, &position);
fputs("bit is in davanagere",fp);
fclose (fp);
}
```

/*gets the file pointer position that is zero at the initial stage*/

/*Writes content to file and now position is 7*/

/*But position variable is having value zero, now fsetpos sets to the value present in position variable that is zero*/
/*Now if we use fputs for writing, then it starts from first position over writing existing content*/

Output: filename.txt file will be having content: **bit is in davanagere**

rewind( ):

rewind( ) function is used to adjust the position of file pointer so that next I/O operation take place at the beginning of file.

Syntax: void rewind (FILE *stream)

It is similar to fseek (FILE *stream, 0, SEEK_SET);

Example:

```
void main( )
{
    FILE * fp;
    fp=fopen ("filename.txt","w");
    fputs("ait is in ckm",fp);
    rewind(fp);
    fputs("bit is in dvg",fp);
    fclose (fp);
}
```

Output:

filename .txt

ait is in ckm

Over writes from beginning position changed to beginning before writing when rewind( ) is used

filename .txt

bit is in dvg

feof ( ):

feof( ) function tests the end of file indicator for given stream.

Syntax: int feof (FILE *stream);

where stream: file pointer name.

This function returns a non zero value when EOF is reached or else it returns zero.

Example:

```
#include <stdio.h>
void main( )
{
    FILE * fp1;
    char ch;
```


Scanned with OKEN Scanner

```
    fp1 = fopen ("file name .txt", "r");
    while (!feof (fp1) )
    {
        ch= fgetc(fp1);
        printf ("%c", ch);
    }
    fclose (fp1);
    getch( );
}
```

o/p The above program reads file filename.txt, One character at a time using fgetc( ) until it reaches EOF.

Remember: **Error handling functions** used while doing file operation are:
clearerr( ): It is function used to clear the EOF and error indicators for stream
Syntax:- void clearerr(FILE *stream);
Example : clearerr(fp1);
perror( ): It is function used to print error message.
Syntax: void perror(char * message) ;
Example: perror("file not found");

Questions: Define a file. Write a note opening and closing afile.
Write a note on functions used for reading content from file: fscanf ( ), fgetc( ), getw( ), fgets( ),fread( ).
Write a note on functions used for writing content to file: fputs( ), fputc( ),fwrite( ), putw( ).
Write a note on functions for selecting records randomly in a file: fseek( ), ftell( ), fsetpos( ), fgetpos(), feof( ), rewind( ).
Study below programs similar to the Lab program based on files.

2.4. Programming Examples

Example 1: Write a C program to read student name, DOB, address and 5 subject marks and print record using nested structure.

```
#include <stdio.h>
#include<conio.h>
struct DOB
{
   int day;
   int month;
   int year ;
},
struct student
{
      int marks [10];
      char address[100];
      char name  [20];
      struct DOB date;
}
```


Scanned with OKEN Scanner

```
void main( )
{
        int i;
        struct student std;
        clrscr( );
        printf ("enter the student name \n");
        scanf ("%s",std.name );
        printf ("enter the student address \n");
        scanf (" %s", std.address );
        printf ("enter the student five subjects marks\n");
        for (i=0; i<5; i++)
                scanf ("%d",&std.a[i] );
        printf ("enter the student DOB \n");
        scanf ("%d%d%d", &std.date.day, &std.date.month, &std.date.year);
        printf (" the student details are  \n");
        printf ("name: \t %s", std. name );
        printf ("address:\t %s", std. address);
        printf ("five subjects marks are:  \n");
        for (i=0;i<5;i++)
                printf ("%d\t",std.a [i]);
        printf ("DOB \t %d - %d-  %d\n ", std.date.day,std.date.month,std.date.year);
        getch( );
}
```

Output:

```
enter the student name
nanjesh
enter student address
chikmagalur
enter student five subjects marks
21      22      23      24      25
enter the student DOB
7       6       1990
The student details are
name:           nanjesh
address:        chikmagalur
five subjects marks are:
21      22      23      24      25
DOB:  7-6-1990
```

Example 2: Program to demonstrate fscanf and fprintf

```
#include<stdio.h>
#include<conio.h>
void main( )
{
        FILE * fp1,*fp2;
        int number ;
       fp1=fopen ("filename1.txt", "r");
       if (fp1==NULL)
       {
```


Scanned with OKEN Scanner

```
            printf( "file  does not exist \n");
            exit(0);
        }
        fscanf (fp1, "%d", &number);
        fp2 =fopen ("filename2 .txt", "w");
        if (fp2==NULL)
        {
            printf( "file  does not exist \n");
            exit(0);
        }
        fprintf (fp2, "number is %d", number);
        fclose  (fp1);
        fclose(fp2);
        getch( ) ;
}
```

Example3: Program to demonstrate fgetc and fputc

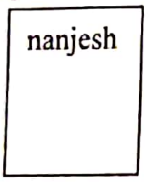
```
#include <stdio.h>
#include<conio.h>
void main( )
{
        FILE * fp1,*fp2;
        char ch;
        fp1=fopen (" filename1.txt", "r");
        if (fp1==NULL)
         {
            printf( "file  does not exist \n");
            exit(0);
        }
        fp2 = fopen ("filename2 .txt", "w");
        if (fp2==NULL)
        {
            printf( "file  does not exist \n");
            exit(0);
        }
        while(!feof(fp1))
        {
            ch=fgetc(fp1);
            fputc(ch, fp2);
        }
        fclose(fp1);
        fclose(fp2);
        getch( ) ;
}
```


Scanned with OKEN Scanner

Example 4: Program to demonstrate fgets and fputs

```
#include <stdio.h>
#include<conio.h>
void main( )
{
        FILE * fp1,*fp2;
        char str [50];
        fp1=fopen (" filename1.txt", "r");
        if (fp1==NULL)
        {
                printf( "file  does not exist \n");
                exit(0);
        }
        fp2 = fopen ("filename2 .txt", "w");
        if (fp2 = = NULL)
        {
                printf( "file  does not exist \n");
                exit(0);
        }
        fgets(str, 4, fp1);
        fputs(str, fp2);
        fclose(fp1);
        fclose(fp2);
        getch( );
}
```

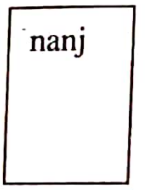
Output: Here filename1.txt



Takes 4 characters and writes to

⇒

filename2.txt



Example5: Program to demonstrate getw( ) and putw ( )

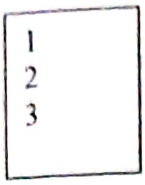
```
#include <stdio.h>
#include<conio.h>
void main( )
{
        FILE * fp1,*fp2;
        int number;
        fp1=fopen (" filename1.txt", "r");
        if (fp1==NULL)
        {
                printf( "file  does not exist \n");
                exit(0);
        }
        fp2=fopen ("filename2 .txt", "w");
        if (fp2==NULL)
```


Scanned with OKEN Scanner

```
    {
        printf( "file  does not exist \n");
        exit(0);
    }
    while (!feof(fp1))
    {
        number =getw(fp1);
        putw(number, fp2);
    }
    fclose  (fp1);
    fclose(fp2);
    getch( ) ;
}
```

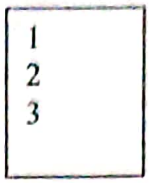
Output:

(old context)
Here filename1.txt



⇒

(newly written context)
filename2.txt



Example 6: Program to demonstrate fread( ) and fwrite( )

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    FILE * fp1,*fp2;
    char str [20];
     fp1=fopen("filename1.txt", "r");
    if (fp1==NULL)
    {
        printf( "file  does not exist \n");
        exit(0);
    }
     fp2 = fopen ("filename2 .txt", "w");
     if (fp2==NULL)
     {
        printf( "file  does not exist \n");
        exit(0);
     }
     fread (str, 1,10, fp1);
     fwrite (str,1,10, fp2);
     fclose  (fp1);
     fclose(fp2);
     getch( ) ;
}
```


Scanned with OKEN Scanner

Output:

filename1.txt

chikmagalur

filename2.txt

chikmagalur

Example 7: Given two university information files "studentname.txt" and "usn.txt" that contains students Name and USN respectively. Write a C program to create a new file called "output.txt" and copy the content of files "studentname.txt" and "usn.txt" into output file in the sequence shown below . Display the contents of output file "output.txt" on to the screen. (lab program)

Refer Lab program 12, Page number:222

Example 8: Given two text documentary files "Ramayana.in" and "Mahabharatha.in". Write a C program to create a new file "Karnataka.in" that appends the content of file "Ramayana.in" to the file "Mahabharatha.in". Display the contents of output file "Karnataka.in" on to screen. Also find number of words and newlines in the output file.

(it is similar to above example 7)

First create file:

ramayana.in

Abhi
Adarsh
Salman
Nanju

Next create another file:

mahabharatha.in

4SH14CS014
4SH14CV015
4SH14IS016
4SH14IP017

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
void main()
{
        FILE *fp,*fp1,*fp2,*fp3;
        int nlines=0,charcount=0,wordcount=0;
```


Scanned with OKEN Scanner

```
char chr;
char str1[20],str2[20], str[20];
clrscr( );
fp1=fopen("ramayana.in","r");
if(fp1==NULL)
        printf("file not found\n");
fp2=fopen("mahabharatha.in","r");
if(fp2==NULL)
        printf("file not found\n");
fp3=fopen("karnataka.in","w");
while(!feof(fp2))
{
        fscanf(fp2,"%s",str2);
        fprintf(fp3,"\n%s",str2);
}
while(!feof(fp1))
{
        fscanf(fp1,"%s",str1);
        fprintf(fp3,"\n%s",str1);
}
fclose(fp1);
fclose(fp2);
fclose(fp3);
fp3=fopen("karnataka.in","r");
while(!feof(fp3))
{
        fscanf(fp3,"%s",str);
        printf("\n%s",str);
}
fclose(fp3);
fp=fopen("karnataka.in","r");
while((chr=getc(fp))!=EOF)
{

    if(isspace(chr)||chr=='\n'||chr=='\t')
            wordcount++;
    if(chr!='\n'&&chr!='\t')
            charcount++;
    if(chr=='\n')
            nlines=nlines+1;

 }
 printf("\nnumber of words in file is %d\n",wordcount);
 printf("\nnumber of characters excluding new line and tab
 space characters are %d",charcount);
 printf("\nnumber of lines are %d",nlines+1); /*including initial line*/
fclose(fp);
getch( );
}
```


Scanned with OKEN Scanner

Output:
4SH14CS014
4SH14CV015
4SH14IS016
4SH14IP017
Abhi
Adarsh
Salman
Nanju
number of words in file is 8
number of characters excluding new line and tab space characters are 53
number of lines are 9

Example 9: Write a C-program to read names of employee in one file (f1) and read their basic salary in another file (f2) and print name and basic salary into 3rd file (f3) where salary is more than 50,000 and also print in stdoutput screen

Assumption f1.txt

nanjesh
ram
neha

f2.txt

60000
30000
70000

output.txt

nanjesh 60000
neha 70000

Output file has to contain details of employees having salary greater than 50000.

```
#include <stdio.h>
#include<conio.h>
void main( )
{
        FILE * fp1,*fp2,*fp3;
        char name[20];
        int salary;
        clrscr ( ) ;
        fp1= fopen(" f1.txt", "r");
        if (fp1 = = NULL)
                printf( "file   not found ");
       fp2 = fopen ("f2. txt", "w");
      if (fp2 = = NULL)
                printf( "file  not found");
      fp3 = fopen ("output. txt", "w");
      while (!feof (fp1)&&!feof (fp2))
      {
              fscanf (fp1 , " %s ", name );
              fscanf (fp2, "%d", salary):
              if (salary >50000)
                    fprintf(fp3, "\n%s%d", name, salary);
    }
```

Note: \n must be given at beginning or else last line will be printed twice as it moves the cursor to next line having no content, previous content only displayed.

Scanned with OKEN Scanner

```
        fclose (fp1);
        fclose (fp2);
        fclose (fp3);
        fp3 =fopen(output.txt", "r");
        printf(" name \t \t salary \n");          /* to print in output screen */
        while(!feof (fp3))
        {
                fscanf (fp3 , " %s ", name );
                fscanf (fp3, "%d", salary);
                printf( "%s\t\t%d\n", name, salary);
        }
        fclose (fp3);
        getch( ) ;
}
```

Output:

name	salary
nanjesh	60000
neha	70000

Example 10: Write a C program to input the following details of 'n' students using structure:
Roll number: integer, name: string, marks: float, grade: char.
Print the name of the students with marks >=70.0%.

```
#include<stdio.h>
#include<conio.h>
void dummy( float *a)
{
   float b =*a;
   dummy(&b);
}
struct student
{
 int rollnumber;
 float marks;
 char name[50], grade;
};
 void main()
 {
        int i,n,found=0;
        struct student s[10];
        clrscr();
        printf("enter the number of students\n",n);
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("enter the student%d details\n",i+1);
                printf("enter the rollnumber\n");
                scanf("%d",&s[i].rollnumber);
                printf("enter the student name without space\n");
```

This dummy function is needed only if you are executing this program. If you execute without this it gives an **error "scanf: floating point formats not linked"**. It is common issue in Turbo C/Borland compiler.Because these compilers sometimes leave out floating point support and use non floating-point version of printf and scanf. The dummy call to a floating point function will force the complier to load the floating point support and remaining program works normally.

Scanned with OKEN Scanner

```
                scanf("%s",s[i].name);
                printf("enter the marks\n");
                scanf("%f",&s[i].marks);
                printf("enter the grade\n");
                fflush(stdin);
                scanf("%c",&s[i].grade);
            }
            printf("student details are\n");
            printf("rollnumber\tname\tmarks\tgrade\n");
            for(i=0;i<n;i++)
                printf("%d\t %s\t %f\t %c\n",s[i].rollnumber,s[i].name,s[i].marks,s[i].grade);
            printf("name of the students with more than or equal to 70 percentage are\n");
            for(i=0;i<n;i++)
            {
                if(s[i].marks>=70.0)
                {
                    printf("%s\n",s[i].name);
                    found=1;
                }
            }
            if(found==0)
                printf("nill\n");
            getch( );
}
```

Output:

```
enter the student1 details
enter the rollnumber
21
enter the student name without space
nanju
enter the marks
54.21
grade
b
enter the student2 details
enter the rollnumber
41
enter the student name without space
usiru
enter the marks
74.86
grade
a
student details are
rollnumber      name        marks          grade
  21            nanju       54.210000        b
  41            usiru       74.860000        c
name of the students with more than or equal to 70 percentage are
usiru
```


Scanned with OKEN Scanner

1. Pointer

Pointer definition: Pointer is a variable that holds the address of another variable.

Declaration and Initilization of pointers:
The operators used to represent pointers are:

- Address operator (&)
- Indirection operator (*)

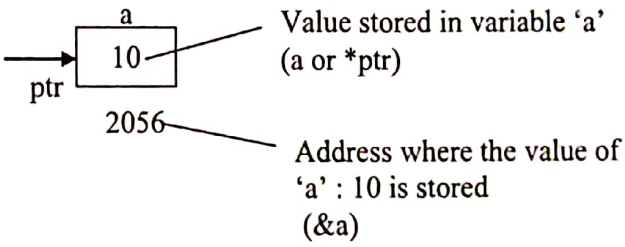
Syntax:
ptr_data_type *ptr_variable_name
ptr_variable_name = &variable_name
where variable_name is the variable whose address has to be stored in pointer.

Example:
int a =10;
int *ptr;
then ptr = &a
*ptr= a;
that is **ptr** is a pointer holding address of variable 'a' and ***ptr** holds the value of **a.**
Example program:

```
#include<stdio.h>
#include<conio.h>
void main( )
{
        int a=10;
        int *ptr;
        ptr=&a;
        printf ("%d\n",  a);
        printf ("%d\n", &a);
        printf ("%d\n",  ptr);
        printf ("%d\n", *ptr);
        getch( ) ;
}
```

Output:

```
10
2056
2056
   10
```



1.1. Pointers and functions (call by reference)

Call by reference method involves use of address of variables as actual parameters in calling function and pointer variables with (*) indirection operator is used at called function to perform required operations that is as formal parameters.

Consider an example of swapping two numbers using call by reference or using pointers

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


Scanned with OKEN Scanner

```
void swap (int *a, int *b);
void main( )
{
        int x =10, y = 20;
        swap(&x, &y);
        printf ("after swapping:\nx=%d\ny=%d", x,y);
        getch( ) ;
}
void swap (int *a, int *b)
{
        int  temp;
        temp = *a;
       *a = *b;
       *b = temp;
}
```

Output:
after swapping:
x=20
y=10
Here instead of passing actual values of x and y pointers, address of x and y are passed.

1.2. Pointers and arrays

The operations performed using array concept can also be done using pointers.
Syntax:

```
data_type *ptr_name;
ptr_name = &array_name or ptr_name = array_name;
```

Here pointer does not point to all the elements of an array, instead initially it points to the first element of an array later which is incremented to get other elements.
Example: int a[10] = {11,12,13,14};

```
int * ptr;
ptr= &a or ptr=a; here ptr is pointing to 11 initially.
```

It can be explained using below program

```
/* program to demonstrate pointers to arrays concept */
#include <stdio.h>
#include<conio.h>
void main( )
{
        int a[10]={11,12,13,14};
        int *ptr;
        ptr=a;                          /*initially pointing to first element 11*/
        for (i=0, i<4; i++)             /* four elements*/
        {
             printf ("%d\t", a[i]);
             printf ("%d\n", &a[i]);
             printf ("%d\t", *ptr);
             printf ("%d\t", ptr);
             ptr++;                     /* making ptr to point next value by doing
        }                                ptr = ptr+1*/
```


Scanned with OKEN Scanner

```
    getch( ) ;
}
```

Output:

```
11      2056  /*value a[0] pointed by ptr at first iteration and address where that value is stored*/
11      2056  /*value *ptr pointed by ptr at first iteration and address where that value is stored*/

12      2058  /*value a[1] pointed by ptr at second iteration and address where that value is stored*/
12      2058  /*value *ptr pointed by ptr at first iteration and address where that value is stored*/

13      2060  /*value a[2] pointed by ptr at third iteration and address where that value is stored*/
13      2060  /*value *ptr pointed by ptr at first iteration and address where that value is stored*/

14      2062  /*value a[3] pointed by ptr at fourth iteration and address where that value is stored*/
14      2062  /*value *ptr pointed by ptr at first iteration and address where that value is stored*/
```

position	0	1	2	3
values	11	12	13	14
adddress	2056	2058	2060	2062

Initial position of ptr

since integer type occupies 2 bytes for each element. If it reserves 2056 for first element as starting address, 2056+2=2058 as second element's starting address, 2058+2=2060 for third element and so on.

Note: if ptr=2056, for every increment of pointer by one (ptr++ that is 2056++) it does not add one to existing address (2056+1 or 2056++ or 2057)stored in pointer, instead it adds size (in bytes) required by one element of that particular datatype, here elements are of integer datatype which requires 2 bytes each hence ptr++ or 2056++ adds 2 to existing value 2056, it gives 2058 and now array is pointing to the value present at the address 2058.

position	0	1	2	3
values	11	12	13	14
adddress	2056	2058	2060	2062

ptr

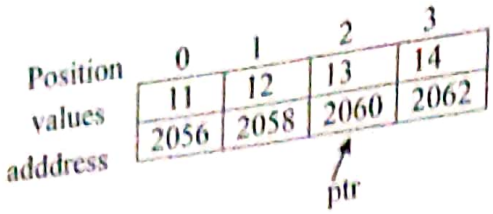
at i=0, a[0] = 11, &a[0] = 2056
*ptr = 11 , ptr = 2056
i++ , ptr++
i=1, ptr=2058

position	0	1	2	3
values	11	12	13	14
adddress	2056	2058	2060	2062

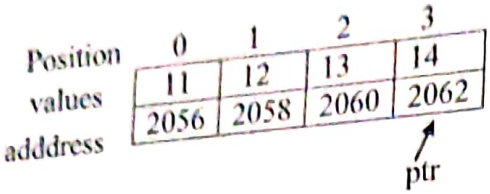
ptr

at i=1, a[1] = 12, &a[1] = 2058
*ptr = 12 , ptr = 2058
i++ , ptr++;
i=2, ptr=2060

Scanned with OKEN Scanner



at i=2, a[2] = 13, &a[2] = 2060
*ptr = 13 , ptr = 2060
i++ , ptr++;
i=3, ptr=2062



a. i-3, a[3] = 14, &a [30] = 2062
*ptr = 14 , ptr = 2062
i++ , ptr++;
i=4, ptr=2064

Note: In exam remember explaination and program (tracing not needed) but depends on marks.

1.3. Character pointer and functions or Pointers to the strings

Strings are array of characters instead of integer values of array, here pointer points to the character present in string represented as an array.

Syntax: data_type &ptr_name;
ptr_name = string_name;

Example: char str[20] = "nanjesh"
char *ptr;
ptr = str;

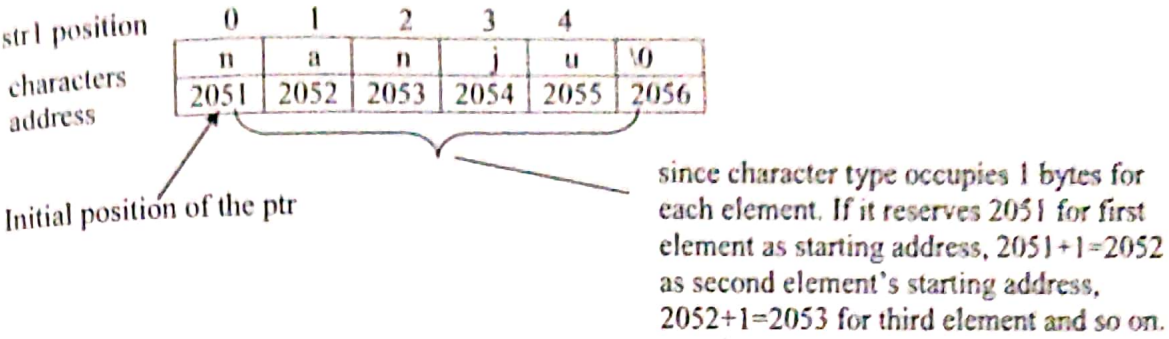
Here pointer does not point to all the character of a string, instead initially it points to the first element or first character of a string later which is incremented to get other elements. It can be explained using below program.

```
/*string copy using pointer to string concept (using single pointer) */
#include<stdio.h>
#include<conio.h>
void main( )
{
   int i;
  char str1[20]= "nanju";
  char str2[20];
  char *ptr;
  ptr = str1;
  for (i=0; str1[i]!='\0'; i++)
  {
```

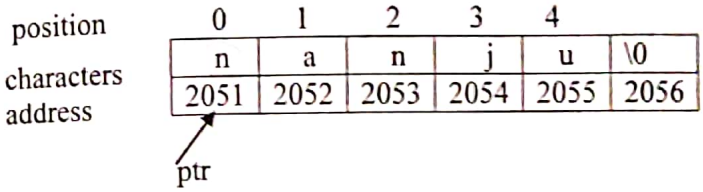

Scanned with OKEN Scanner

```
        str2[i]=*ptr;
        ptr++;
    }
    str2[i]='\0';
    printf("string2 after copying is %s", str2);
    getch( ) ;
}
```

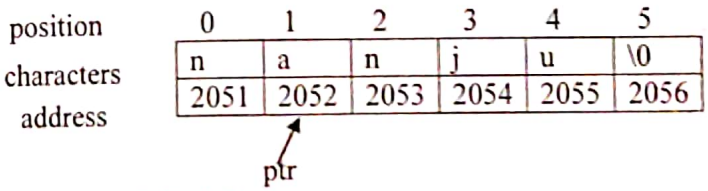
Output:
string2 after copying is nanju



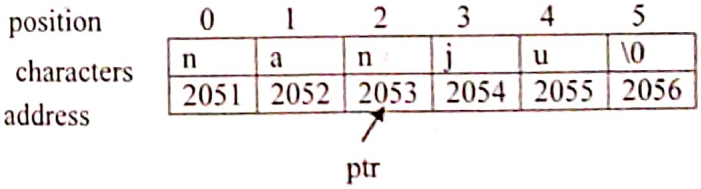
Note: if ptr=2051, for every increment of pointer by one (ptr++ that is 2051++) it adds size (in bytes) required by one element of that particular datatype, here elements are of character datatype which requires 1 byte each hence ptr++ or 2051++ adds 1 to existing value 2051, it gives 2052 and now array is pointing to the character present at the address 2052.



at i=0, str1[0]! = '\0'
str2[0] = *ptr = n
i++ = 1 , ptr++ = 2052



at i=1, str1[1]!='\0'
str2[1] = *ptr = a
i++=2 , ptr++=2053



Scanned with OKEN Scanner

at i=2, str1[2]!= '\0'
str2[2] = *ptr = n
i++=3 , ptr++=2054

position	0	1	2	3	4	5
characters	n	a	n	j	u	\0
address	2051	2052	2053	2054	2055	2056

ptr

at i=3, str1[3] ='\0'
str2[3] = *ptr = j ,
i++=4 , ptr++=2055

position	0	1	2	3	4	5
characters	n	a	n	j	u	\0
address	2051	2052	2053	2054	2055	2056

ptr

at i=4, str1[4] != '\0'
str2[4] = *ptr = u
i++=5 , ptr++=2056

position	0	1	2	3	4	5
characters	n	a	n	j	u	\0
address	2051	2052	2053	2054	2055	2056

ptr

at i=5 , str1[5] = = '\0' so here iterations stops since while(str1[i]=='\0')

Note: In exam remember explaination and program (tracing not needed) but depends on marks.

1.4. Pointer to pointer

Pointer is a variable that stores the address of another variable. Pointer storing the address of another pointer, that is pointer pointing to another pointer is called as pointer to pointer.

Declaration:

```
data_type **pointer_name;
```

Initialization:

```
pointer_name = &another_pointer_name
```

The below example demonstrates pointer to pointer concept:

```
int a = 10;
int *ptr1, **ptr2;
ptr1 = &a;
ptr2 = &ptr1;          /*ptr2 is the pointer to the another pointer ptr1*/
```


Scanned with OKEN Scanner

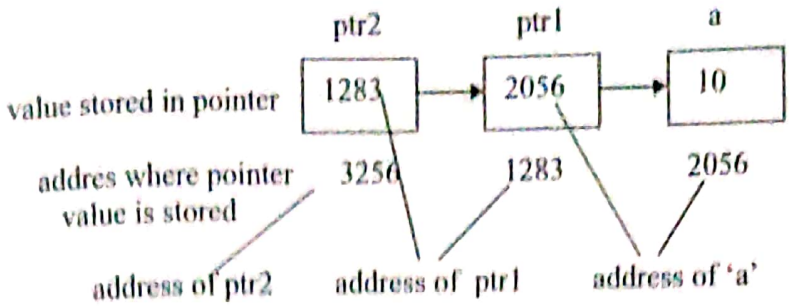


Figure 1: Pointer to pointer

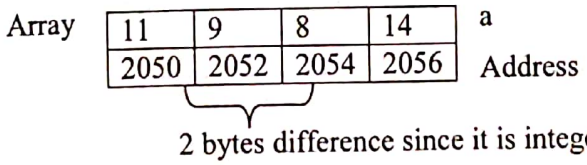
```
#include <stdio.h>
#include<conio.h>
void main( )
{
      int  a=10;
      int *ptr1, **ptr2;
      ptr1 = &a;
      ptr2 = &ptr1;
      printf ("%d\n", a);
      printf ("%d\n", &a);
      printf ("%d\n", ptr1);
      printf ("%d\n", &ptr1);
      printf ("%d\n", *ptr1);
      printf ("%d\n", ptr2);
      printf ("%d\n", *ptr2);
      printf ("%d\n", **ptr2);
      getch( );
}
```

output

10
2056
2056
1283
10
1283
2056
10

1.5. Address Arithmetic

1. An integer value can be added or subtracted from a pointer. It can be incremented or decremented.




```
#include <stdio.h>
#include<conio.h>
void main( )
{
      int  a[4]={11,9,8,14};
      int *ptr
      ptr = &a;
      printf ("%d\n", ptr) ;
      printf ("%d\n", *ptr ) ;
      ptr++;  /* we can use ptr=ptr+1 or ptr+=1 */
      printf ("%d\n", ptr ) ;
```

output

2050
11
2052

Scanned with OKEN Scanner

```
printf ("%d\n",* ptr ) ;
ptr--; /* we can use ptr=ptr-1 or ptr- =1*/
printf ("%d\n", ptr) ;
printf ("%d\n", *ptr ) ;
getch( );
}
```

2050
11

Note: If one is added to the pointer then it means that number of bytes required to store one element of that particular type are added to existing pointer value.
In the above example ptr = ptr+1
= 2050+1=2052 since it is an integer (2 bytes)

Remember:
Suppose If it is a float, then ptr=ptr1+1=2050+1=2054 (4 bytes)
If it is a double then ptr=ptr1+1=2050+1=2058 (8 bytes)
If it is a character then ptr=ptr1+1=2050+1=2054 (1 bytes)

2. If two pointers pointing to same type of data then one pointer value can be assigned to another.
Example: int *ptr1, *ptr2;
ptr1=ptr2;
3. Pointer can be assigned a Null
Example: int *ptr1;
ptr1=NULL;
4. Subtraction of two pointer variables can be performed when both are pointing to elements of same array.
5. Two pointers cannot be multiplied added or devided directly
6. Relational operators can be used between the pointer
Example: int * ptr1, * ptr2;
ptr1>ptr2 , ptr1=ptr2, ptr1<ptr2 etc.

Questions: Define pointer. Explain declarartion and initialization with example.
Explain pointer to array concept with example.
What is pointer to pointer? Explain with example.
Write a C program to read n elements to an array and print those elements using pointer to an array.

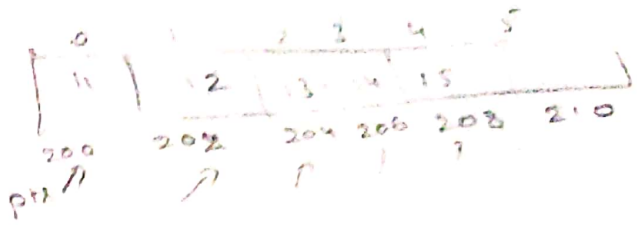
1.6. Programming examples

Example 1: Write a C program to read n elements to an array and print those elements using pointer to an array.

```
#include <stdio.h>
#include<conio.h>
void main( )
{
    int a[100], *ptr, i;
    printf ("enter number of elements \n");
```


Scanned with OKEN Scanner

```
    scanf ("%d", &n);
    ptr=a;
    for (i=0; i<n; i++)
    {
          scanf ("%d", ptr);   /* no & symbol since ptr itself an address*/
          ptr++;
    }
    ptr=a;                          /* initialize ptr back to first element*/
    printf("array elements are");
    for (i=0; i<n; i++)
    {
          printf ( "%d\t", *ptr);
          ptr++;
    }
    getch( );
}
```



Output:-

```
enter number of elements
5
11      12      13      14      15
Array elements are
11      12      13      14      15
```

Example 2: program to demonstrate pointers to string using separate pointer to the string1 and string2.

```
#include <stdio.h>
#include<conio.h>
void main( )
{
        int i
        char str1[20]= "ait",
        char str2[20],*ptr1 * ptr2;
        ptr1=str1;
        ptr2=str2;
        for (i=0; *ptr1! ='\0';i++)
        {
            *ptr2 =*ptr1;
            ptr1++;
            ptr2++;
        }
        *ptr2='\0';
        printf("string2 after reversing  is %s", str2);
        getch( );
}
```

Example 3: Write a C program using pointers to compute the sum, mean and standard deviation of all elements stored in an array of n real numbers.

Refer Lab Program 14, Page number: 225

Scanned with OKEN Scanner

1.7. Dynamic Memory Allocation

The process of allocation of memory size at runtime is called as dynamic memory allocation. Mainly there are four functions for memory management, namely:

- malloc( )
- calloc( )
- free( )
- realloc( )

malloc( ): It allocates memory from a freely available memory and returns a pointer to a block of contriguous memory of specified size.
Syntax: pointer_name = (data_type *)malloc(byte_size);
Example: int *a;
a = (int*) malloc(20);

calloc( ): It allocates memory for array elements and initialize them to zero and also returns a pointer to memory.
Syntax: pointer_name = (data_type*)calloc (n, size)
where n is number of memory blocks
size is size of blocks to be allocated in bytes
Example: int *a;
a= (int*)calloc(5,2);

free( ): free function is used to free the memory blocks allocated by malloc and calloc.
Syntax: free(pointer_name);
Example : int * ptr;
free (ptr);

realloc( ): It is used to modify or change the size of already allocated memory block.
Syntax: pointer_name = realloc(pointer_name , new_size)
Example: int *ptr;
ptr = realloc(ptr,4); /*4 bytes to the variable pointed by pointer ptr*/

Program to demonstrate allocating memory at runtime using malloc( ):

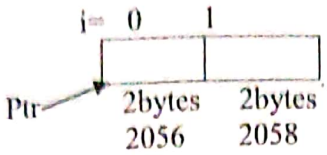
Here array of elements are stored in dynamically allocated memory without any static allocation of size.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int  *ptr, n, i;
    printf ("enter number of elements \n");
    scanf ("%d", &n);
    ptr = (int *)malloc(n*sizeof(int));
    for (i=0; i<n; i++)
    {
        scanf ("%d", ptr +i);
    }
    printf(" elements are \n");
```


Scanned with OKEN Scanner

```
        for (i=0; i<n; i++)
        {
                printf("%d", *(ptr+i));
        }
        getch( );
}
```

Output:
if we give n=2
ptr=(int*)malloc(2*size of (int)); /*allocates 4 bytes memory block */



at i =0
scanf ("%d", ptr+i)
stores new element to ptr+i=2056+0=2056 location

at i =1
scanf ("%d", ptr+i)
stores new element to ptr+i=2056+1=2058 location (since 2056 + 1 =2056+2bytes=2058)
similarly printing is done using *(ptr+i)

2. Preprocessor Directives

Preprocessor is a program which is invoked by complier before the compilation of user written program. The declaration of preprocessor statements always begin with (#), usually these are placed before the main( ) function.

- **#include**: specifies the files to be included
- **#define**: defines a macro substitution
- **#undef**: undefining a macro
- **#ifdef**: verifies whether marco is defined or not
- **#ifndef**: verifies whether marco is defined or not, if it is not defined, it assumes condition to be true.
- **#if** : used to write generalized conditions using relation operator
- **#else**: used as an alternative to # if
- **#error**: prints error message on stdrr
- **#programa**: give special commands to compiler using standardized method
- **#endif**: ends the # if statements

2.1. #include:

#include loads specified file before compilation of user written program.
Syntax: #include<header_file_name.h> or #include "headerfilename.h"
Example: #include<stdio.h> or #include "stdio.h"
stdio.h has macros stdin, stdout ,stderr, stdprn etc.
ctype.h has macros isalpha(x) islower(x), isupper (x) etc. For more details on header files available in C, Refer section 3.2 of Module-III for header files.

Scanned with OKEN Scanner