POINTERS

Introduction

Pointer is a variable that holds the address of another variable. All the pointer operations are done through two operators: '*' (star and '&' (ampersand). '&' (also known address of operator) is a unary operator that returs a memory address of a variable where as '*' (also known as value at address operator) returns value stored at a memory location stored in a pointer.

Pointer declaration

Syntax:

```
datatype *pointer_variable ;
```

For eg:

```
int *a;
float *b;
```

Pointer Initialization

Syntax:

```
datatype *pointer_variable = &variable ;
```

For eg:

```
int x=5;
int *p=&x;
```

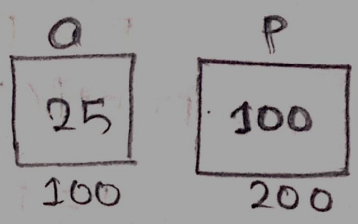
IMP: Pointer Arithmetic

We can add an integer to pointer, subtract an integer from pointer and subtract two pointer of same type. All these operations can be performed with pointers.

i) Increment/decrement of pointer

Integer, float, char, double data type pointers can be incremented and decremented. The pointers increment/decrement is depends on the data type.

For eg:




```
#include<stdio.h>
    int main()
    {
     int  a = 25;
     int *P;
      P= &a;
printf("Value of a=%d\n", a);      //25
printf("Value of a=%d\n", *P);     //25
printf("Address of a=%u\n", &a);   //100
printf("Address of a=%u\n", P);    //100
      P++;
printf("Address of P= %u", P);     //104
      ++P;
printf("Address of P = %u", P);    //108
      P--;
printf("Address of P= %u",P);      //104
return 0;
}
```

ii) Addition/Subtraction of a value to/from pointer

For eg:

```
#include<stdio.h>
  int main()
 {
```

```
int a=25;
int *p;
p= &a;
printf("a=%u", p);        // 100
p = p+3;
printf("p=%u", p);        // 100+ 3x4 = 112
p= p-2;
printf("p =%u", p);       // 112 - 2x4 = 104
return 0;
}
```

Pointerscannot be added to eachother. Addition and subtraction with float or double data type pointers are not allowed.

iii) Subtraction of two pointers/comparison

Example:

```
#include <stdio.h>
int main()
{
int a= 25;
int *p1, *p2;
p1 = &a;
p2 = &a;
P1 = P1+2;        // 100 + 2*4 = 108
P2 = P2-4;        // 100 + 4*4 = 116
printf("Subtraction of two pointers = %d", P2-P1);
// Comparison of two pointers
```

```
    if (P1 == P2)
    {
    printf("Both are pointing to same address");
    }
  return 0;
  }
```

I.M.P.

Returning multiple values from a functions using pointer

Using a call by reference intelligently we can make a function return more than one value at a time, which is not possible ordinarily. This is shown in the program given:-

Example:

i) WAP to display array elements using pointer.

```
#include<stdio.h>
  int * display();
   int main()
  {
   int *p, i;
   p= display();
  printf("Array elements are:\n");
   for(i=0; i<5; i++)
  {
   printf("%d\t", p[i]);
   }
   return 0;
  }
```

```
int *display()
{
int A[5], i;
printf("Enter elements:");
for(i=0; i<5; i++)
{
scanf("%d", &A[i]);
}
return A;
}
```

ii) Sorting array elements using function in pointer

```
#include<stdio.h>
int* sort (int*, int);
int main()
{
int A[100], n, i, *P;
printf("Enter size:");
scanf("%d", &n);
printf("Enter array elements:");
for(i=0; i<n; i++)
{
scanf("%d", &A[i]);
}
P= sort (A, n);
printf("Sorted data are :");
for (i=0; i<n; i++)
```

```
{ printf("%d\t", P[i]);
}
return 0;
}
int * sort(int *A, int n)
{
int i, j, temp;
for(i=0; i<n; i++)
{
for (j= j+1; j<n ; j++)
{
if (A [i] > A[j])
{
temp= A[i];
A [i] = A[j];
A[j]= temp;
}
}
}
return A;
}
```

iii) WAP to display Array elements.

```
#include<stdio.h>
int main()
{
int A[5], i;
```

```
    printf("Enter array: ");

     for (i= 0; i<5; i++)
     {
      scanf("%d",(A+i));
     }
    printf("\n Array elements :");
      for (i= 0; i<5; i++)
      {
      printf("%d\t", *(A+i));
      }
  return 0;
}
```

Pointer to array [1D Array]

When an array is declared the compiler allocates base address and sufficient amount of storage memory to contain all the elements of the array in contiguous memory location.

for example:

i) Pointer to an array

```
#include<stdio.h>
  int main()
  {
   int A[5], i, *p;
     p= A;
   printf("Enter array: ");
```

```
for(i=0; i<5; i++)
{
scanf("%d",(p+i) );
 }
 printf("\n Array elements :");
  for(i=0; i<5; i++)
  {
  printf ("%d\t", *(p+i) );
  }
 return 0;
}
```

ii) Array of pointer

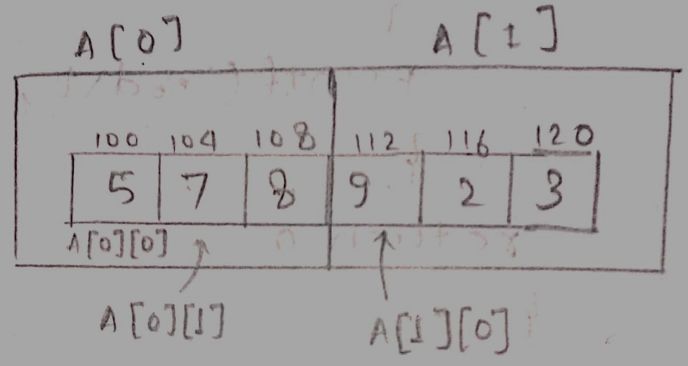
```
#include<stdio.h>
   int main()
{
 int A [5], i;
 int *p [5];
 for(i=0; i<5; i++)
  {
   p[i] = &A [i];
  }
printf("Enter array elements:");
   for(i=0; i<5; i++)
    {
     scanf("%d", p[i]);
    }
```

```
        printf("Array elements are :");
        for(i=0; i<5; i++)
        {
        printf("%d\t", *p[i]);
        }
    return 0;
}
```

2D Array

For eg:

```
int A[2][3]
int (*p)[2];
int *p[2];
```



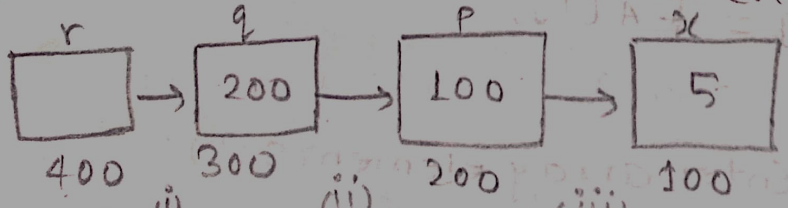
→ Represents whole Array

→ Represents base address

IMP

Double Indirection (Double pointer / Pointer to pointer)

C is very powerful programming language that can also support a mechanism to create pointer of pointer. It is also known as chain of pointer.



(i) int ***r = &q (ii) int **q = &p (iii) int *p = &x

For example:

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

```
int main()
{
 int x= 5;
 int *p= &x;
 int **q = &p;
 int ***r = &q;
printf("value of x= %d ", *p);
printf("value of x= %d ", *(*q));
printf("value of x= %d", *(*(*r)));
return 0;
}
```

Pointer & String

Strings are created like an array. Compiler automatically inserts NULL character at '\0' at the end of a string.

For eg:

```
char ch[20] = "Money";
 char *str;
  str = ch;
```

V.IMP

Dynamic Memory Allocation (DMA)

The exact size of array is unknown until the compile time. The size of array we have declared initially can be sometimes insufficient and sometimes more than required. DMA allows a program to obtain more memory space, while running or to release space when no space is required. It is the

process of allocation of memory at a runtime. It uses heap memory. There are four library routines known as memory management functions that can be used for allocating and freeing memory during program execution, and they are:

i) malloc()
ii) calloc()
iii) realloc()
iv) free()

i) malloc()

The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form. The malloc() function takes one argument that specifies the total memory to be allocated.

Syntax:

```
pointer_variable=(type_cast*)malloc(n*sizeof(datatype));
```

This function contains garbage value as a default value and stores data as a whole in memory.

For eg:

```
int *ptr;
ptr=(int*)malloc(5*sizeof(int));
```

ii) calloc()

The calloc() takes two arguments, the first argument is the number of items and the second argument is size of each item for which the memory is to be allocated.

Syntax:

```
pointer_variable=(typecast*)calloc(n, sizeof(int));
```

It contains zero as default value and stores data one by one in memory

iii) realloc()

If the previously allocated memory is insufficient or more than sufficient, then we can change memory size previously allocated using realloc().

Syntax:

```
pointer_variable=(type_cast*)realloc(pointer_variable, newsize);
```

for eg:

```
int *ptr;
  ptr=(int*)malloc(5*sizeof(int));
  ptr=(int*)realloc(ptr, 10*sizeof(int));
```

This function contains garbage value and when we use it, data remains safe(doesn't loss)

iv) free()

Dynamically allocated memory with either calloc() or malloc() does not get return on its own. The programmer must use free() explicitly to release space.

Syntax:

```
free(pointer_variable);
```

Eg: `free(ptr);`

Programs:

1. To input 'n' numbers and display items.

```
#include<stdio.h>
#include<stdlib.h>
    int main()
{
  int *ptr, n, i;
  printf("Enter n:");
  scanf("%d", &n);
  ptr=(int*)malloc(n*sizeof(int));
      if(ptr==NULL)
      {
       printf("Memory allocation failed");
       exit(0);
      }
      else
      {
       printf("Enternumbers:");
          for(i=0; i<n; i++)
           {
           scanf("%d",(ptr+i));
           }
           printf("Enterednumbers are:\n");
            for(i=0; i<n ; i++)
             {
           printf("%d\n"*(ptr+i));
           }
   free(ptr);
  return 0;
}
```

Assignment

1) WAP to calculate sum of 'n' natural number using DMA.

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

int main()
{ int n, i, *ptr, sum=0;
  printf("Enter the value of n :");
  scanf("%d",&n);

  ptr=(int*) malloc(n * sizeof(int));
   if(ptr==NULL)
    {
    printf("Memory allocation failed");
     exit(0);
     }

 //Initializing array elements
   for(i=0; i<n; i++)
      {
      *(ptr+i)= i+1;
      }

  // Calculating sum of natural number
    for(i=0; i<n; i++)
      {
       sum = sum+ *(ptr+i);
      }

 printf("The sum of first %d natural
        number = %d", n, sum);
  free(ptr);
 return 0;
 }
```

STRUCTURE AND UNION

Structure

C supports structure which allows us to wrap one or more variables with different data types. A structure can contain any valid data types like int, char, float even arrays or even other structures Structure is a collection of heterogeneous data types in a continuous memory location under a common name called structure tag or structure name.

Syntax:

```
struct structure_name
{
 data-type1 var1;
 data-type2 var2;
    :
 data-typeN varN;
};
```

For eg.

```
struct student
{
 char name[20];
 int roll;
 float marks;
};
```

Declaring structure variable

Syntax:

```
struct structure_name variable list;
```

For eg:

```
struct student S1, S2;
    S1.roll = 5;
    S2. marks = 2.5;
```

Program

```
#include<stdio.h>
  struct student
  {
   char name[20];
    int roll;
   float marks;
  };
 int main()
{
 struct student s;
 printf("Enter student detail\n");
 scanf("%s %d %f", &s.name, &s.roll, &s.marks);
 printf("Detail of student:\n");
 printf("Name = %s", s.name);
 printf("\nRoll = %d", s.roll);
 printf("\nMarks = %f", s.marks);
 return 0;
}
```

Nested Structure

Sometimes, a member of a structure needs to have multiple values of different types. For such situations the member should be variable of another structure type. C program provides such facility which is known as nesting of structure. If a structure encounter within another structure as a member, then such complex data structure is known as nested structure.

For eg:

```
#include<stdio.h>
  struct DOB
{
  int yy, mm, dd;
};
 struct student
{
  char name[20];
   int roll;
  struct DOB d;
};
struct student s;
printf("Enter details of student:");
scanf("%s%d%d%d%d", s.name, &s.roll,
                &s.d.yy, &s.d.mm, &s.d.dd);

printf("Details of student:\n");
printf("Name=%s:\n Roll=%d", s.name,
                    s.roll);
```

```
printf("\n DOB = %d-%d-%d", s.d.yy, s.d.mm,
                                s.d.dd);
return 0;
}
```

Array of structure

Structure is used to store the information of a particular object but if we need to store such multiple objects then array of structure is used.

For eg:

```
#include<stdio.h>
  struct DOB
  {
   int yy,mm,dd;
  };
  struct student
  {
    char name[20];
      int roll;
      struct DOB d;
  };
 int main()
 {
  struct student s[100];
    int n, i;
  printf("Enter no. of students : ");
  scanf("%d", &n);
   for(i=0; i<n; i++)
    {
```

```
        printf("\nEnter details of student:%d ", i+1);
        scanf("%s%d%d%d%d", s[i].name, &s[i].roll,
              &s[i].d.yy, &s[i].d.mm, &s[i].d.dd);
    }
    printf("\nDetails of students:\n");
    for(i=0; i<n; i++)
    {
        if(s[i].roll==5){
        printf("\nStudent %d ", i+1);
        printf("\nName= %s", s[i].name);
        printf("\nRoll=%d ", s[i].roll);
        printf("\nDOB =%d-%d-%d ", s[i].d.yy,
               s[i].d.mm, s[i].d.dd);
    }
    }
    return 0;
}
```

Assignment

1. Given a structure

Name	Address	Phone	Salary	Date_of_join		
				mm	dd	yy

Write a program to read details of 100 employees and display the records of employee living in "Dhangadhi".

```
#include<stdio.h>
struct DOJ
{
 int mm, dd, yy;
};
struct employee
 {
  char Name[100], Address[100];
  int Phone;
  float Salary;
  struct DOJ d;
 };
  int main()
 {
  struct employee e[100];
  int i;
 for(i=0; i<100; i++)
  {
  printf(" Enter employee detail:\n");
  scanf("%s%s%d%f%d%d%d", e[i].Name,
        e[i].Address, &e[i].Phone, &e[i].Salary,
        &e[i].d.mm, &e[i].d.dd, &e[i].d.yy);
  }
printf("Details of employee:\n");
for(i=0; i<100; i++)
 {
  if(e[i].Address="Dhangadhi")
```

```
        {
        printf("\n Employee %d ", i+1);
        printf("\n Name = %s ", e[i].Name);
        printf("\n Address = %s ", e[i].Address);
        printf("\n phone = %d", e[i].Phone);
        printf("\n salary = %f ", e[i].Salary);
        printf("\n Date of Join = %d-%d-%d",
                    e[i].d.mm, e[i].d.dd,
                        e[i].d.yy);
        }
    }
  return 0;
}
```

Structure and pointer

i) Referencing pointer to another address to access memory

```
#include <stdio.h>
  struct student
 {
  char name [20];
   int roll;
 };
 int main()
 {
 struct student *p, s;
    p = &s;
```

```
printf("Enter details :");
scanf("%s%d", p->name, &p->roll);
printf("\nDetails of student ");
printf("\n Name = %s ", p->name);
printf("\n Roll = %d", p->roll);
return 0;
}
```

ii) Using dynamic memory allocation

```
#include<stdio.h>
  Struct student
 {
 char name[20];
   int roll;
 };
  int main()
 {
 struct student *p;
   int n, i;
 printf("Enter n:");
 scanf("%d", &n);
 p=(struct student *)malloc(n * sizeof(struct
                          student));
 printf("Details of student:");
 for(i=0; i<n; i++)
 {
 scanf("%s%d", (p+i)->name, &(p+i)->roll);
```

```
    printf("Details of students:\n");
    for(i=0; i<n; i++)
{
 printf("\nName: %s",(p+i)->name);
 printf("\nRoll: %d", (p+i)->roll);
 }
return 0;
}
```

Structure and function

A structure can be passed as a function argument just like any other variable. For eg:

i)

```
#include<stdio.h>
 struct person
 {
  char name[20];
   int age;
 };
 void display(struct person);
  int main()
 {
  struct person p;
  printf("Enter name and age:");
  scanf("%s %d", p.name, &p.age);
  display(p);
  return 0;
 }
```

```
void display(struct person p1)
 {
  printf("Person detail :\n");
  printf("\nName = %s", p1.name);
  printf("\n Age = %d ", p1.age);

 }
```

ii)

```
#include<stdio.h>
 struct person
{
 char name[20];
  int age;
 };
 struct person input();
    int main()
 {
  struct person p;
    p= input();
printf("\nPerson details:\n");
printf("\nName = %s", p.name);
printf("\n Age = %d ", p.age);
 return 0;
}
 struct person input()
  {
    struct person p1;
```

```
    printf(" Enter name and age: ");
    scanf("%s%d", p1.name, &p.age);
    return p1;
}
```

Union

Both structures and unions are used to group a number of different variables together. Syntactically both structures and unions are almost same. The main difference is in storage. In structure each member has its own storage location but all member of a union use the same memory location.

Syntax:

```
union union_name
{
datatype1 var1;
datatype2 var2;
   :
datatype N varN;
};
```

⊛ Declaration union variable

Syntax:

```
union union_name variable_list;
```

⊛ Accessing member

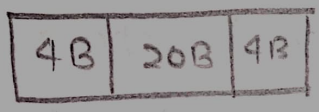
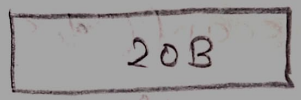
Syntax:

```
union_variable.member;
```

Assignment

i) Defferentiate structure and union.

Structure	Union

i) struct student
{
int roll ;
char name[20];
float marks;
};
struct student s;
s.roll = 5
s.name = "Ram";
 
 	
	i) union student
	{
	int roll;
	char name[20];
	float marks;
	};
	union student s;
	s.roll = 5;
	s.name = "Ram";
	 

ii) The keyword struct is used to define a structure	The keyword union is used to define a union.
iii) Each member within a structure is assigned unique storage area of location.	iii) Memory allocated is shared by individual member of union.
iv) It is reusable.	iv) It cannot reuse.
v) Data remains safe.	v) Data loss due to overwriting of data in same location.
vi) It requires more memory.	vi) It requires less memory.
vii) Individual memory can be accessed at a time	vii) Only one member can be accessed at a time.
viii) Easy to program	viii) Hard to program.

Program:

i)

```
#include< stdio.h>
 union st
 {
 char name [20];
  int roll;
  float marks;
 };
 int main()
 {
  union student s;
  printf("Enter name of student :");
  scanf("%s ", s.name);
  printf("Name=%s", s.name);
  printf("Enter roll:");
  scanf("%d", &s.roll);
  printf("\nRoll=%d ", s.roll);
  printf("Enter marks :");
  scanf("%f", &s.marks);
   printf ("\n Marks=%f ". s.marks );
 return 0;
 }
```

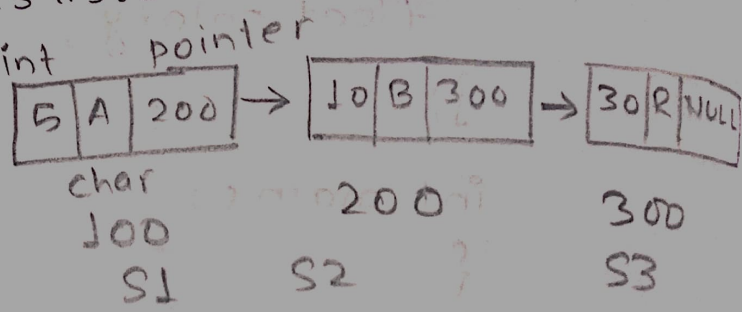
Assignment

1) WAP to create a union employee having member name, age and salary. Find the total salary of 3 employee.

```
#include <stdio.h>
  union Employee
{
  char name [20];
   int age ;
   float salary ;
};
int main()
{
 union Employee e1, e2 , e3;
 float totalsalary = 0;
printf("Enter the name, age and salary of
        1st employee:");
scanf("%s%d%f", e1.name, &e1.age, &e1. salary);

printf("Enter name, age and salary of 2nd employee);
scanf("%s%d%f ", e2.name, &e2.age. &e2.salary);

printf("Enter name, age and salary of 3rd employee);
scanf("%s%d%f ", e3.name, &e3.age, &e3.salary);

totalsalary= e1. salary+e2. salary+e3.salary;
printf("Total salary of 3 employee = %f",
                    totalsalary);
return 0;
}
```

Self Referential Structure

A self referential structure is a structure that contains a pointer to a structure of the same type. Self-referential structures are very useful in applications that involve linked data structures, such as lists and trees.



for eg:

```
#include<stdio.h>
   struct node
   {
    int a;
    char ch;
    struct node *next;
   };
    int main()
    {
    struct node S1, S2, S3;
     S1.a= 5;
     S1.ch= 'A';
     S2.a= 'B';
     S3.a= 30;
     S3.ch= 'R';

     S1.next= &S2;
     S2.next= &S3;
     S3.next= & NULL;
  printf("For S1:\n");
  printf("a =%d", S1.a);
  printf("For S2:\n");
  printf("a =%d", S1.next->a);
  retur 0;
  }
```

} Linking

Assignment

1) Use and application of self referential structure.

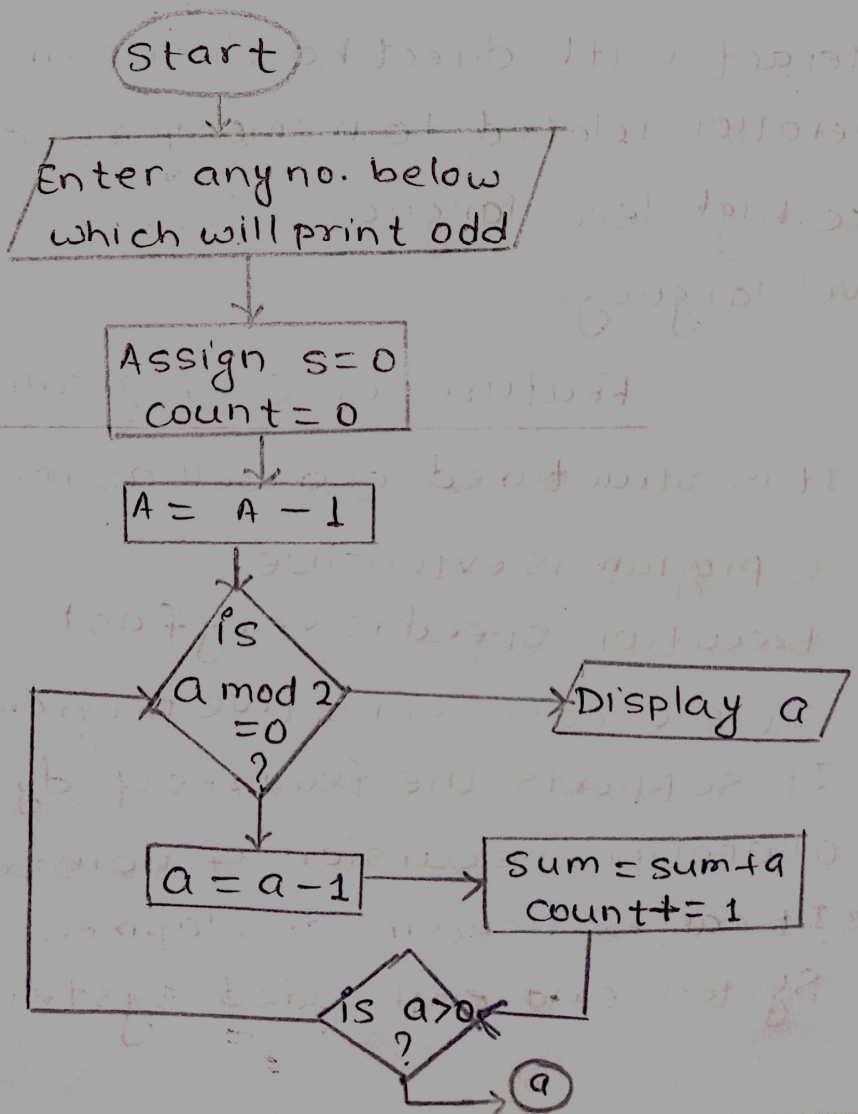
Self referential structure are also known as linked structure or linked lists. They are useful for storing and manipulating data that is related in some way, such as a list of items.

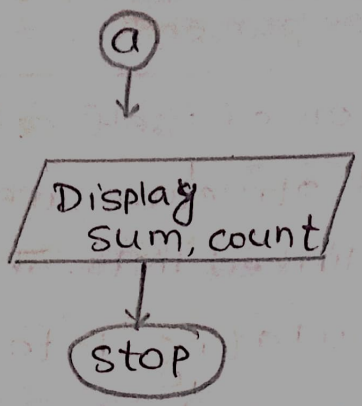
One of the most common applications of self-referential structure is to create linked lists. linked lists can be used to implement a variety of data structures, such as stacks, queues, and hash tables.

First Terminal Examination 2022

2) Q. No. 1 (b)

Ans:-





3) Q. No. 2 (b)

Ans:

C is called middle level language

C has the feature of both assembly level language and higher level language. It bridges the gap between traditional machine level language and more conventional high-level language. We are able to manipulate bits, bytes and address. It can interact with direct hardware and also carry out operation related to memory management behave like high level language. So, C is called middle level language.

Features of C Program

→ It is structured as well as middle level language.
→ C program is extensible.
→ Execution speed is very fast.
→ It is a case-sensitive programming language.
→ It supports the features of dynamic memory allocation, recursion & pointer etc.
→ It can be used in development of operating system and embedded system.

4) Q. No. 5(b) (i)

```
#include<stdio.h>
void main()
{
int x=0,i=0;
for(i=1; i<10; i++)
 {
  if(i%2==0)
  x+=i;
  else
  {
   x--;
  printf("%d\t",x);
  }
 }
printf("\nx=%d",++x);
getch();
}
```

Output,

x=0								
i=1	2	3	4	5	6	7	8	9
x=0	-1	1	0	4	3	9	8	16
	-1+2		0+4		3+6		8+8	
0-1		1-1		4-1		9-1		16-1
-1		0		3		8		15

x = 1+15

Output =

-1 0 3 8 15

x = 16

5) Q. No.5(b)(ii)

```
#include<stdio.h>
#include<string.h>
  int main()
{
  char str[]= "ABCDE";
```

```
    int i, j;
  for(i=0; i<strlen(str); i++)
   {
   for(j=0; j<=i; j++)
   {
    printf("%c ", str[j]);
   }
   printf("\n");
  }
 return 0;
}
```

6) Q. No. 6 (a)

```
#include<stdio.h>
  int main()
{
  int OP, a, b, c, s, n, f=1, i;
 printf(" Enter 1 for calculating sum of three
   number :");
 printf("\n Enter 2 for Calculate factorial and
       3 for check whether a number is
       odd or even:);
 scanf("%d",&OP);
 switch(OP)
 {
  case 1:
  printf("Enter three no.:");
  scanf("%d%d %d", &a, &b, &c);
```

```
    s= a+b+c;
  printf("Sum =%d", S);
    break;
 case 2:
    Printf("Enter any no.:");
     scanf("%d", &n);
       for (i= 1; i<=n ; i++)
       {
       f= f* i;
         }
     printf ("Factorial =%d ", f );
   break;
 case 3:
  printf("Enter any number:");
    scanf("%d", &n);
    if (n % 2 == 0)
      {
       printf("%d is even", n);
      }
     else
      {
      printf("%d is odd", n);
      }
  break;
default:
printf(" Invalid choice!");
  }
 return 0;
}
```

FILES AND FILE HANDLING

Introduction

A file is a place the disk where a group of related data is stored. File handling is a process of creating a new file, append records in file and read records from file.

i) Text File

A text file is a human readable sequence of character and the words they form that can be encoded into computer readable format such as ASCII. It's extension is .txt.

ii) Binary files

A binary file is a collection of bytes made up of machine readable symbols that represents 1's and 0's. It's extension is .dat.

I.M.P.

File Opening Modes

Mode	Symbol		Meaning
	Binary	Text	
read	rb	r	For read only.
write	wb	w	For write only, new file is created if no file is exist otherwise overwrite in existing file
append	ab	a	Same as write but does not overwrite
read/ write	rb+	r+	Reading existing contents, writing new contents, modifying existing contents of the file.
write/ read	wb+	w+	Creates a file for both reading and writing.
write/ read	ab+	a+	opens a text files for reading and writing preserving previous contents.

File Operations

i) Opening a file

FILE pointer is used to locate file.

Syntax:

(a) To create file pointer

```
FILE *pointer_variable
```

(b) To open a file

fopen()

Syntax:

```
file pointer_variable= fopen("Filename", "opening modes");
```

The fopen() firstly searches the file on the disk, then it loads the file from the disk into a place in memory.

ii) Closing a file

Syntax: `fclose(pointer_variable);`

Reading a file (one character at a time)

Syntax:

```
character_variable = fgetc(filepointer);
```

Writing a file

Syntax

```
fputs(character_variable, file_pointer);
```

Program

i) Reading character one by one

```
#include<stdio.h>
#include<stdlib.h>
    int main()
    {
     FILE *fp;
      char ch;
      fp= fopen("abc.txt", "r");
       if (fp==NULL)
       {
       printf(" File can't be open");
        exit(0);
        }
   while (1)
   {
    ch = fgetc(fp);
     if (ch==EOF)
     {
      break;
      }
    else {
      printf("%c", ch);
      }
    }
  fclose(fp);
   return 0;
}
```

ii) Writing character one by one

```
#include<stdio.h>
#include<stdlib.h>
    int main()
   {
    FILE *fp;
    char ch, choice = 'Y';
    fp = fopen("abc.txt", "a+");
       if (fp== NULL)
       {
       printf("File cannot be open");
        exit (0);
        }
    while(choice=='Y')
    {
     printf("Enter character:");
     scanf("%c", &ch);
      fputs(ch, fp);
    printf("Do you want to write more?");
     scanf("%c", &choice);
      }
     fclose(fp);
     return 0;
    }
```

Read & write in a file (String wise)

i) fgets()

It is used to read string from a file.

Syntax:

```
fgets(string_variable, maximum length, file_pointer);
```

ii) fputs()

It is used to write string (character array) in a file.

Syntax:

```
fputs(string_variable, filepointer);
```

Example:

```
#include<stdio.h>
#include<stdlib.h>
    int main()
  {
    FILE *fp;
     char str[43];
     fp=fopen("abc.txt","w");
      if(fp==NULL)
      {
       printf("File cannot be opened");
        exit(0);
       }
       else
       {
       printf("Enter text");
        gets(str);
       fputs(str,f
       fclose(fp);
       }
```

```
- fp=fopen("abc.txt","r");
  fgets(str, 43, fp);
   puts(str);
  fclose(fp);
return 0;
}
```

iii) fscanf()
It is used to read data from file.

Syntax:

```
fscanf (filepointer, "format_specifier", listof
                                        variables);
```

iv) fprintf()
It is used to write data in a file.

Syntax:

```
fprintf(filepointe, "format_specifier", listof
                                        variables);
```

Example:

```
#include<stdio.h>
#include<stdlib.h>
    int main()
  {
    FILE *fp;
    char name[20];
     int age ;
    fp=fopen ("abc.txt", "w+");
        if (fp == NULL)
```

```
{
printf("File cannot be opened");
 exit(0);
 }
else
 {
printf("Enter name and age: ");
 scanf("%s%d", name, &age );
fprintf(fp, "%s%d", name, age);
   rewind(fp);
fscanf(fp, "%s%d", name, &age);
printf("name=%s", name);
printf("\nAge = %d", age);
  }
   fclose(fp);
  return 0;
  }
```

IMP Q. Write a program to read name, author and price of 500 books and store the information in a file book.txt. Display the bookname and price of those books whose price is above 300.

```
#include<stdio.h>
#include<stdlib.h>
    struct book
```

```
{
char name[20], author[20];
 float price;
};
int main()
{
 struct book b[500];
 FILE *fp;
 int i;
fp=fopen("book.txt", "w+");
 if(fp== NULL)
    {
     printf("File cannot open");
     exit(0);
     }
   printf("Enter book details:");
    for(i= 0; i<500; i++)
    {
     scanf("%s %s%f", b[i].name, b[i].author,
                          &b[i].price);

   fprintf(fp,"%s %s%f ", b[i].name, b[i].author,
                                b[i].price);
   }
    rewind(fp);
   printf("Book details :\n");
    for(i= 0; i<500; i++)
     {
   fscanf(fp, "%s%s%f", b[i].name, b[i].author,
                          &b[i].price);
```

```
        if(b[i].price>300)
        {
         printf("\nBook=%s", b[i].name);
         printf("\nPrice=%f", b[i].price);
        }
       }
      fclose(fp);
      return 0;
      }
```