POINTER

define :- A pointer is a variable whose value is the address of another variable. i.e direct address of a memory location.

Like variable or constant we must declare a pointer before using it to store any variable address.

General form of pointer variable declaration is :-

type *var-name ;

type is pointer's base type i.e valid C data type
var-name is name of pointer variable
asterisk * is used to declare a pointer (is same used for multiplication)

eg → int *ip; // pointer to an integer.
double *dp // pointer to a double
float *fp // pointer to a float
char *cp // pointer to a char

CODE

```
int var = 10
int *p;
p = &var;
```

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

int var = 10;

int *p;          // pointer declaration

p = &var;        // pointer initialization

printf("Address of var is : %p", &var);
printf("\n Address of var is : %p", p);

printf("\n value of var is : %d", var);
printf("\n value of var is : %d", *p);
printf("\n value of var is : %d", *(&var));

printf("\n value of pointer p is : %p", p);
printf("\n Address of pointer p is : %p", &p);

return 0; }
```

Self Referential Structures

Those structures that have one or more pointers which point to the same type of structures as their member.

USE OF POINTERS

(1) pointers are more efficient in handling arrays and structures.

(2) It allow C language to store dynamic memory management

(3) It reduces length of program and its execution time as well.

(4) pointers allow refernces to ~~from~~ function and thereby helps in passing of function as arguments to other function.

Date / /
Page No.

STRUCTURE

DEFINE

Structure is a user-defined data type in C language which allow us to combine data of different types together.
It construct a complex data type which is more meaningful

`struct` keyword is used to define a structure. it defines a new data type which is a collection of primary & derived data types.

Syntax :-

```
struct [structure_tag]
{
    // member variable 1
    // member variable 2
    ....
} [structure_variables];
```

Example :-

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    char gender; // F for female, M for Male };
```

Structure variables

```
int main()
{
    struct Student S1;

    S1.age = 18;

    strcpy(&S1.name, "Varchasvi");

    strcpy(S1.gender, "F");

    printf("Name of Student 1 : %s \n", S1.name);
    printf("Gender of student 1 : %s \n", S1.gender);
    printf("Age of Student 1 : %d \n", S1.age);

    return 0;
}
```

Array of Structures

In C it can be defined as the collection of multiple structures variable where each variable contains information about different entities.
It is used to store information about multiple entities of different data types.
The array of structures is also called the collection of structures.

Array of Structures

```
#include <studio.h>

struct student
  {
    char name[25];
    int id;
    float marks;
  };

void main(){
    struct student s1, s2;
    int dummy;

printf("Enter name, id, marks of student 1:");
scanf("%s %d %f", s1.name, &s1.id, &s1.marks);
scanf("%c", &dummy);

printf("Enter name, id, marks of student 2:");
scanf("%s %d %f", s2.name, &s2.id, &s2.marks);
scanf("%c", &dummy);

  }

    int main(){
  int i;
    struct student st[5];
printf("Enter records of 5 students");

for(i=0; i<5; i++)
  { printf("\n Enter Roll no:");
    scanf("%d", &st[i].rollno);
```

POINT..

Date / /
Page No.

```
printf("\n Enter Name: ");
scanf(" %s", &st[i].name);
}
return 0; }
```

RECURSION

Define :- It is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until we get to a small enough problem that can be solved trivially. Usually recursion involves a function calling itself.

CODE

* fibonacci Series

```
#include <studio.h>
    int fibonacci(int);
    void main() {
     int n, f;
printf("Enter value of n:");
scanf("%d", &n);
 f = fibonacci(n);
  printf("%d", f);
}

  int fibonacci(int n)
    {
      if (n == 0)
        { return 0; }
     else if (n == 1)
       { return 1; }
     else
       { return fibonacci(n-1) + fibonacci(n-2); }
```

Output :-

Enter value of n : 12
144

0+1+1+2+3+5+8+13+21+34+55+89=144

* factorial Numbers

```
# include <studio.h>
  int fact(int);
  int main() {
   int n, f;
print("Enter number whose factorial you want to
          Calculate : ");
  f = fact(n);
  printf("factorial = %d", f); }
  }
int fact(int n)
  {
    if (n==0)
    { return 0; }
    else if (n==1)
     { return 1; }
    else
     { return n* fact(n-1); }
  }
```

Output :-
Number = ~~000~~ 5

factorial = 12

1*2*3*4*5 = 120

CALL BY VALUE Vs CALL BY REFERENCE

Call By Value	Call By Reference
(1) Copy of the value is passed into function	(1) An address of value is passed into function
(2) Changes made inside function is limited to the function only	(2) changes made inside the function validates outside the function also.
(3) Actual parameter value do not change by changing formal parameters	(3) Actual parameters do change by changing value of formal parameters
(4) Actual and formal arguements are created at the different memory location.	(4) Actual and formal arguements are created at the same memory location

function → It is a group of statements that together perform a task.

Every C program has at least one function which is main().

~~Built in libraries~~

1-D ARRAY Vs 2-D ARRAY

1-D Array	2-D Array
① Store Single list of the element of similar data type.	① store "list of lists" of the element of a similar data type
① A simple data structure that stores a collection of similar type data in a one contiguous block of memory.	① A type of array that stores multiple data elements of same type in matrix or table like form with number of rows and columns.
② Also called single dimensional array	② Also called multi-dimensional array
③ Syntax : datatype[] name = new datatype[size];	③ Syntax : datatype[][] name = new datatype[rows][columns];
④ stores data in list	④ stores data in rows + columns format

Array Define :- It is a collection of data items all of the same type, accessed using a common name.