UNIT :→ 2

Date / /

☆ Pointers :→

1. Pointer Variable :- A pointer is a variable that store memory address like all other variables. It also has a Name, has to be declare and occupie some space in memory. It is called pointer, because it points to a particular location in memory by storing the address of that location.

2. Declaration of pointer variables :-
Like other variables pointer variable should also be declare before being used. The general syntax of declaration of variable is :-

```
data_type *p_name;
```

Here p_name is a name of pointer variable which should be a valid C identifier.
The Asterisk '*' preceding this name informs the compiler that the variable is declare as a pointer.
Here data_type is known as the base type of the pointer.

Let us take some pointer declaration.

```
int *iptr;
float *fptr;
char *cptr, ch1, ch2;
```

Here iptr is a pointer that should point to variable of type int.
Similarly fptr and cptr are should point to variables of float and char-type.

Date / /

We have done in the 3rd declaration statement where Ch1 and Ch2 are declare as a variable of type char.

Operators Use with Pointers :→

There are two basic operators used with pointers:-

① Address Operator : & (Ampersand) :-
C provides an address operator which writtens the address of a variable when placed before a.

② Indirection Operator : * (Asterik) :-

☆ Pointer Arithemetic :→

1st sess.

All types of arithemetic operations are not possible with pointers. The only valid operations that can be perform are as :-

1. Addition of an intiger to a pointer & increment operator.
2. Subtraction of a intiger from a pointer and decrement operations.
3. Subtraction of a pointer from another pointer of a same time.

For Ex :-
If we have an integer value PI which contain address 1000, then on incrementing we get,
Incrementing we get 1002 instead of 1001.

Date / /

This is because int data-type is 2.

Similarly on decrementing PI we will get, 998 instead of 999.

The expression (Pi+3) will represent the address, Let us see pointer arthemetic for int, float and char pointers.

```
int   a=5 , *Pi=&a;
float b=2.2, *Pf=&b;
char  c='x', *Pc=&c;
```

Suppose the address of variable a, b, c are 1000, 4000, 5000 so initially values of P1, P2, P3 will be 1000, 4000, 5000

Pi++ or ++Pi	= 1002
Pi = Pi-3	= 996
Pi = Pi+5	= 1006
Pi-- or --Pi	= 1004

Pf++ or ++Pf	= 4004
Pf = Pf-3	= 3992
Pf = Pf+5	= 4012
Pf-- or --Pf	= 4008

Pc++ or ++Pc	= 5001
Pc = Pc-3	= 4998
Pc = Pc+5	= 5003
Pc-- or --Pc	= 5002

Date / /

☆ Array with Pointer :→

```
#include<stdio.h>
#include<conio.h>
 void  main()
{
   int  marks[] = {10, 20, 30};
   int  *point[3], i;
   for(i=0; i<3; i++);
   {
      printf("%d", marks[i]);
      point[i] = &marks[i];
   }for(i = 0; i<3; i++)
    {
       printf("%d", *point[i]);
    }
      getch();
    }
```

Output :- 10 20 30,

10 20 30

Q :→ Write a program for Understanding the concept of Array of Pointer :→

```
#include<stdio.h>
#include<conio.h>
   void  main()
  {
    int  *pa[3];
```

Date / /

```
int i, a=5, b=10, c=15;
  Pa[0] = &a;
  pa[1] = &b;
  pa[2] = &c;

for(i=0; i<3; i++)
{
    printf("%d", pa[i]);
    printf("%d", *pa[i]);

    getch();
}
```

Output :-

5
10
15

Call by Value	Call by Reference
`void disp(int x);`	`void disp(int *x);`
`main()`	`main()`
`{`	`{`
`int a=100;`	`int a=100;`
`disp(a);`	`disp(&a);`
`void disp(int x)`	`void disp(int *x)`
`{`	`{`
`x=x+100;`	`*x=*x+100;`
`}`	`}`

Date / /

Call by Value	Call by Reference
Output :-	Output :-
x = 200	a = 100
a = 100	x = 2156

9m ☆ Dynamic Memory Allocation :→

The memory Allocation that we have done till now was static memory allocation.

The memory that could be used by the pro-gram was fixed we could not increase or the decrease the size of memory during the execution of program.

In many applications it is not possible to predict how much memory would be needed by the program at run time.

Ex :-

```
int emp[200]
```

In an array it is must to specify the size of array while declaring, so the size of this array will be fixed during run time.

Now two types of problem make occur.

① The First case is that the no. of values is to be store is less then the size of array, and hence, there is wastage of memory.

For example :- If we have to store only 50 value then space for 150 value (300 bytes) is wasted.

② In second case, our program face, if want to store more values than the size of array.

For Ex :- If there is need to store 210 value

Date / /

in the above array.

To overcome these problems, we should be able to allocate memory at run time.
The process of Allocating memory at the time of execution is called dynamic memory allocation.
The allocation and release of this memory space can be done with the help of some built-in-functions, whose protypes are found in alloc.h, stdlib.h, Hearder files.

① Malloc ( )
Declaration :→

```
void *malloc(size_t size);
```

This function is use to allocate memory dynamicly. The argument size specify the no. of bytes to be allocated.
The type_t define in <stdlib.h> on success malloc ( ) returns a pointer to the first byte of allocated memory.

It is generally use as :-

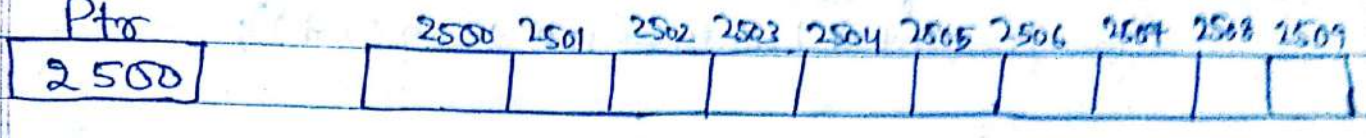
```
Ptr = (datatype *) malloc (Specified size);
```

Where ptr is a pointer of type_datatype and specified size, is the size in bytes required to be reserved in memory.
The expression (datatype *) is used to typecast the pointer written by malloc ( ).

For Example :→

Date / /

```
int *ptr;
ptr = (int *) malloc(10);
```



This allocates 10 continguous bytes of memory space and the address of first byte is stored in the pointer variable. This space can hold 5 intigers. The allocated memory contains garbage ~~memory~~ or valve.

We can use sizeof operator to make a program portable and regdable.

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

This allocates the memory space to hold 5 intiger values.

If there is not sufficient memory available in heap then malloc() returns 'NULL'. so, we should always check the valve return by malloc().

```
ptr = (float *) malloc(10 * sizeof (float));

if(ptr == NULL)
    printf("Sufficient Memory not Available");
```

Q:- Write a program to understand Dynamic allocation of a memory ?

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

```
void main()
{
  int *p, n, i;
printf("Enter the number");
scanf("%d", &n);
p=(int *) malloc(n * sizeof(int));

 if(p == NULL)
 {
 printf("Memory not Available");
    exist(1);
  }
    for(i=0; i<n; i++)
     {
       printf("Enter an integer");
       scanf("%d", p+i);
      }

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

① Declaration :-

① alloc() :-

```
void *calloc (size_t n, size_t size);
```

Date / /

The called Alloc() is used to allocat multiple blocks of memory. It is somewhat smiliar to the malloc() except for two difference.

(i) It takes two arguement the :-
1. Specify the no. of blocks.
2. Specify size of each blocks.

(ii) The other difference between alloc() & malloc() is that the memory allocate by malloc() contain garbage value. While the memory allocated by alloc() is 0.

② realloc() Declaration :→

void * realloc (void * ptr, size_t newsize);

If we want to increase or decrease the memory allocated by malloc() and alloc().
The function realloc() is use to change the size of the memory block. It alter the size of memory block without losing the all data. This is known as reallocation of memory.