UNIT-4

Pointer:- A pointer is a variable which store the address of another variable.

→ In other words, pointer is a variable that represent the location of a data item such as a variable or an array element.

→ Some of pointers Applications are:

- to pass information between functions.
- to return multiple data item from function.
- provide alternative way to access array elements.
- to pass array and strings as function arguments.
- for dynamic memory allocation of a variable.

DECLARING POINTER VARIABLES

→ A pointer provides access to a variable by using the address of that variable

→ Syntax of declaring variable can be given as:

```
data_type *pointer_name;
```

→ Here data_type is the data type of the value that the pointer will point to.

→ Example :-

```
int *ptr;
char *ptr;
float *ptr;
```

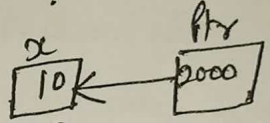
data type as below:

```
int *Ptr1;
char *ptr2;
float *Ptr3;

Pf("%d", sizeof(Ptr1)); ⟶ 2 byte
Pf("%d", sizeof(Ptr2)); ⟶ 2 byte
Pf("%d", sizeof(Ptr3)); ⟶ 2 byte.
```

→ Declaring a integer pointer :

```
int x = 10;
int *Ptr;
Ptr = &x;
```



Address → 2000

In above code '*' inform the Compiler that Ptr is a pointer variable & int specifier that it will store the address of an integer variable. The '&' operator take the address of (x) & copies that to the contents of the pointer Ptr.

Example :-

```
#include <stdio.h>
main()
{
int num, *Ptr;
Ptr = &num;
Pf("Enter the number");
Sf("%d", num);
Pf("%u", &num);
getche()
}
```

output :
Enter the num : 5
2000 ← address of 5

Note :- * ANY Number of pointers can point to the same address.

Pointers Arithmetics in C

→ The pointer can use following operations as:

1) increment / Decrement of a pointer
2) Addition of integer to a pointer
3) Subtraction of integer to a pointer
4) Subtracting Two pointers of same type.

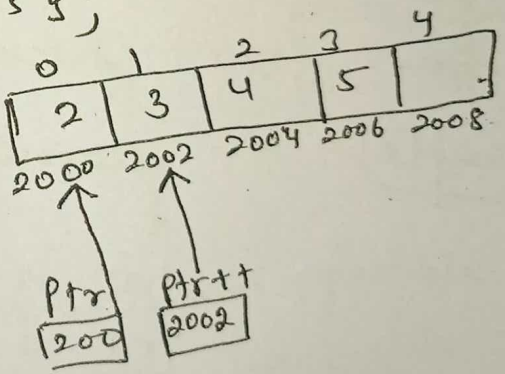
1) Increment / Decrement of pointer :-

→ When pointer is increment / decrement, it actualy increment / Decrement by the number equal to the size of the data type for which it is point.

f.e:-

```
int a[5] = { 2,3,4,5 };
int *ptr;
ptr = &a[0];
ptr++;
Pf("%u", ptr);
```

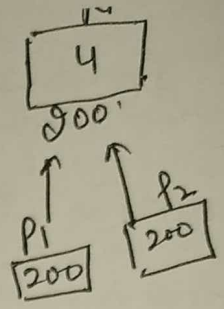
(2002)



2) Addition of integer to a pointer :-

When a pointer is added with a value, the value is first multiply by the size of data type & then added to

```
{
int N=4;
int *P1, *P2;
P1 = &N;
P2 = &N;
```



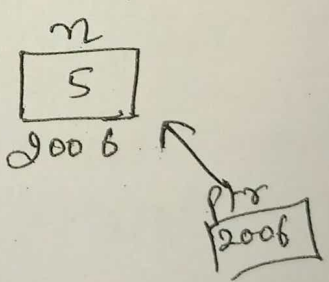

```
Pf("pointer before addition %u", P2); // 200
P2 = P2+3; // 200 + 3*2 = 206
Pf("pointer after addition %u", P2);
                                206
}
```

3. Subtraction of integer of a pointer :-

→ When a pointer is subtracted with a value, the value is first multiplied by the size of the data type & then subtracted from the pointer.

Example :-

```
main()
{
int n=5;
int *ptr;
ptr = &n;
Pf("pointer before Substraction %u", ptr);
                                    (2006)
ptr = ptr-3; // 2006-3*2;
Pf("pointer after substraction %u", ptr);
                                    (2000)
}
```



4) Subtracting two pointers of same type :-

→ Subtraction of two pointers is possible only when they have same data type & the result is generated by calculating the difference between the address of the two pointers.

Example:-

```
main()
{
int n = 4;
int *p1, *p2;
p1 = &n;   // p1 = 2000
p2 = &n;   // p2 = 2000
p2 = p2+3; // p2 = 2000 + 3x2 = 2006
x = p2 - p1; // 2006 - 2000 = 6 byte ⇒ 6/2 = 3
Pf("Subtraction of two pointers is: %d", x);
                                        (3)
}
```

n
4
2000

Note:-

* Addition, Multiplication & Division is not possible using two pointer variables of same type.

for Example:-

```
int *p, *q, a = 5;
p = &a;
q = &a;
q = p+q; / p*2 / p/2;
Pf("%u", q);
```

} Not allowed in 'C'

UNIT IV

1) Pointer to single dimensional array :-

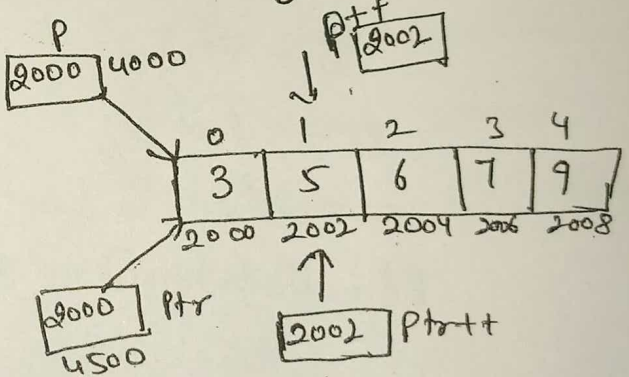
→ We can declare a pointer that can point to whole array instead of only one element of the array.

→ Syntax :- data_type (* var_name) [size_of_array] ;

→ Example :- int (* Ptr) [10] ;

→ Here 'Ptr' is pointer that can point to an array of 10 integers.

→ The type of Ptr is "pointer to an array of 10 integers"



Program :-

```
main()
{
int * P;
int (*Ptr)[5];
int arr[5];
P= arr;
Ptr = &arr;
Printf("%u,%u", P, Ptr);
P++;
Ptr++;
Printf("%u,%u", P, Ptr);
getche();
}
```

output :

2000, 2000

2002, 2002

2 Pointer to two dimensional array:-

→ In two dimensional array, we can access each element by using two subscripts, where first subscript represent the row number & second subscript represent the colomn number.

→ The element of 2-D array can be accessed with the help of pointer also.

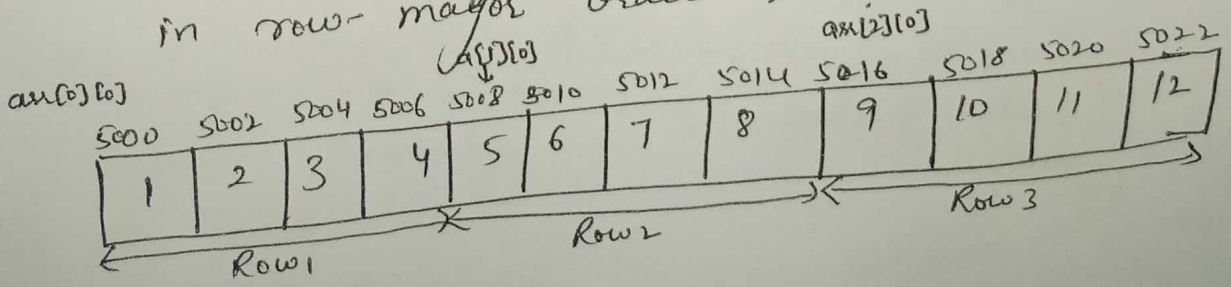
→ Syntax:- datatype *(*(variable name))[ ][ ];

Example:- *(*(arr+i)+j)

```
int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8},
                  {9, 10, 11, 12} }
```

	Col1	C2	C3	C4
Row1	1	2	3	4
Row2	5	6	7	8
Row3	9	10	11	12

→ Since 2-D array is stored the 2-D array in row-major order. like:



a Two dimensional array can be considered as a collection of one-dimensional array that are placed one another.

→ Here, 'arr' is an array of 3 element where each element is a 1-D array of 4 integer.

→ As name of array is constant pointer that points to 0th 1-D array & contain address 5000.

→ Since 'arr' is of 2byte integer therefore arr+1 = 5008 & arr+2 = 5016

→ In general, arr+i point to i^{th} element of arr. so expression *(arr+i) given us the base address of i^{th} 1-D array.

Example:

```
main()
{
int arr[3][4] = {
                  { 10, 11, 12, 13 },
                  { 20, 21, 22, 23 },
                  { 30, 31, 32, 33 }
                };

  int i, j;
for (j=0; j<4; j++)
Printf ("%d %d", arr[i][j], *(*(arr+i)+j));
getch()
}
```

DYNAMIC MEMORY ALLOCATION

→ The process of allocating memory to the variable at runtime is known as dynamic memory allocation

→ Dynamic memory allocation gives best performance in situation in which we do not know memory requirment in advance.

→ To allocate memory dynamicaly 'c' use four library functions as:

1) malloc()
2) calloc()
3) realloc()
4) free()

* All these function are declared in the header file as: <stdlib.h>

→ Dynamic allocation process can be done only with the help of pointers only.

Memory Allocation process:-

→ Dynamic memory is allocating in the form of 'heap'.

→ The size of heap is not constant as its keep

→ When memory is full then 'overflow' condition arises & it will return as null pointer.

Allocating a block of memory :-

→ 1) malloc() : fonction is used to allocate the memory block & return a pointer of type 'void'

→ Syntax:

```
ptr = (cast_type*) malloc (byte_size);
```

where 'ptr' is a pointer of type cast-type.

→ Example:-

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

Program:-

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
main()
{
int n, *ptr, i;
printf("Enter number n");
scanf("%d", &n);
ptr = (int*) malloc (n*sizeof(int));
  if (ptr==NULL)
  {
  printf("Memory is full");
  exit(1);
  }
 else{
  printf("Enter the number");
  for(i=0; i<n; i++)
  scanf("%d", ptr+i);
  printf("output values are");
  for(i=0; i<n; i++)
  printf("%d", *(ptr+i));
  free(ptr);
  } }
```

2 Calloc() :- Calloc() stand for Contiguous memory allocation & it is used to allocate memory for arrays.

Syntax :- Ptr = (cast_type*) calloc (n, elem_size);

→ The above statement allocates contiguous space for 'n' blocks each size of element size byte.

→ The only difference between malloc() & calloc() is that when we use calloc(), all byte are initialized to zero. where as, malloc() byte are initialized to garbage value.

→ If there are enough space in memory then NULL pointer will be returned.

Example :-

```
#include <stdio.h>
#include <stdlib.h>
main()
{
int *ptr, n, i;
Printf("enter number n");
Scanf("%d", &n);
ptr = (int*) calloc (n, sizeof(int));
Printf("Enter element");
for(i=0;i<n;i++)
Scanf("%d", ptr+i);
Printf("elements are");
for(i=0; i<n; i++)
Printf("%d", *(ptr+i));
free(ptr);
getch()
}
```

→ Some times the memory allocated by using calloc() or malloc() might be insufficient.

→ So to change the memory size already allocated by calloc() or malloc(), we can use realloc().

→ This process is called reallocation of memory.

Syntax:- ptr = (cast-type*) realloc (ptr, newsize);

→ The function realloc() allocates new memory space of size specified by newsize to the pointer variable ptr.

→ realloc() takes two argument. The first is the pointer referencing the memory & second is the total number of bytes you want to reallocate.

Example:-

```
#include <stdio.h>
#include <stdlib.h>
main()
{
int *ptr, i;
ptr = (int*) malloc (2* sizeof(int));
printf ("Enter no");
for(i=0; i<2; i++)
scanf ("%d", ptr+i);

ptr = (int*) realloc (ptr, 2* sizeof (int));
printf ("Enter new element");
for (i=2; i<4; i++)
scanf ("%d", ptr+i);
printf ("elements are");
for (i=0; i<4; i++)
printf ("%d", *(ptr+i));
}
```

4 free( ) ;-

→ When a variable is allocated space during the Compile time, then the memory used by the variable is automatically released by the system in accordance with its storage class.

→ But when we dynamically allocate memory then it is responsibility of programmer to release it space when it is not required.

→ we can free the memory block by using the free( ) function.

Syntax:- free (Ptr) ;

→ Where ptr is a pointer that has been created by using malloc() or calloc( ).

→ When memory is de-allocated using the free(), it is returned back to the free list with in the heap.

Example Program:-

```
#include<stdio.h>
#include<stdlib.h>
main()
{ int *Ptr;
Ptr = (int*)malloc (2*sizeof(int));
free(Ptr);
getch()
}.
```

Pointer and functions

→ We can use pointer with function by 3 ways:

1) Pass address as argument to function
2) Pass pointer as argument to function
3) function call using function pointer.

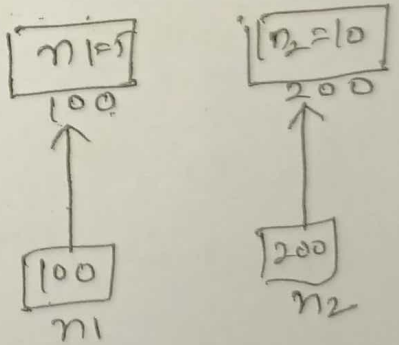
1) Pass address as argument to function:-

→ In this methode passing the address as argument will be copies into formal parameter of function.

→ Inside the function, the address is used to access the actual argument used in the call.

Example:

```
#include<stdio.h>
Void swap(int*, int*);
main()
{
int n1=5, n2=10;
swap(&n1, &n2);
Printf("%d %d", n1, n2);
}
Void swap(int *n1, int *n2)
{
    int temp;
    temp = *n1;
    *n2
```



→ In C, we can pass pointer to a function by declaring the function parameter as a pointer type.

Example:-

```
#include<stdio.h>
void add(int*);
main()
{
int *p, i=10;
p= &i;
add(p);
printf("%d",*p);
getch();
}
void add(int *ptr)
{
(*ptr)++;
}
```

3 Function call using function pointers:-

→ In C, we can call the function by using function pointers.

→ Unlike normal pointer, a function pointer points to code, not data.

→ function pointer stores the starting address of Executable code.

Sytax of function pointer:-

return_type (* pointer_name) (argument list);

Example:-

```
#include <stdio.h>
int add(int, int);
main()
{
int (* ptr)(int, int);
ptr = &add;
int z = ptr(5, 6);
printf("%d", z);
}
int add(int x, int y)
{
int k = x+y;
return (k);
}
```