Chapter - 8
Pointers

1] C Pointers :- A pointer is a variable that stores the address of some other variable. It is a derived data type that is created from the fundamental data types. The pointer variable can also access the value stored in the variable whose address they contain. A pointer is a variable that is stored at a location in a memory and it stores the address of variable to which it points if

Example :- If a variable "Var" having value 100 has been stored at memory location address 6000 in memory the pointer to "Var" will contain a value 6000. A pointer to "Var" can manipulate the value stored at location of var ie 100.

| Variable | Value | Address |
|---|---|---|
| Var | 100 | 6000 |
| P | 6000 | 4048 |

2] Advantages of Pointer :- The advantages of pointer are:

1] With the use of pointers, one can return multiple values from function.
2] We can allocate or deallocate space in memory by using pointers
3] Pointers are used to efficiently handle the arrays
4] Pointers are used to implement several data stru

use like stacks, Queues, Linked, lists and trees.

3) By using pointers, one can pass an array or a string as a parameter to a function

4) The use of pointers results into faster and more effective code

3) Address of (&) operator :- The address of" operator returns the address of the variable to which it precedes. We can mostly use this operator while using Scanf() function

Example :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
  int var = 100;
  printf("\n The address of Var is %u", &Var);
  getch();
}
```

4) Null pointer :- A pointer that is not assigned any value but Null is known as the null pointer. If we donot have any address to be specified in the pointer at the time of declaration we can assign Null value

Example :-

```
int *p = NULL;
```

Exc

5:] Pointer Program to swap two numbers without using third variable

```
#include <stdio.h>
Void main(){
int a=10,b=20,*p1=&a,*p2=&b;
printf("Before Swap:*p1=%d *p2=%d",*p1,*p2);
*p1 = *p1 + *p2;
*p2 = *p1 - *p2;
*p1 = *p1 - *p2;
printf("\n After Swap:*p1=%d *p2=%d",*p1,*p2);
}
```

6:] Pointer Variables Declaration :-

A pointer variable has to be declared before it can be used in program.

Syntax :-

```
Data_type *name;
```

Here "data type" is the type of the pointer variable and "name" is the name of pointer variable. The *[Asterisk] is used to inform the Compiler that the variable being declared is a pointer.

If we don't initialize pointers, they will have garbage values.

7:] Intializing pointers :- Pointers are also intialized by using assignment operator (=) of C

Pointers are intialized with addresses.

Example :- int a = 90;

```
int *ptr = &a;
```

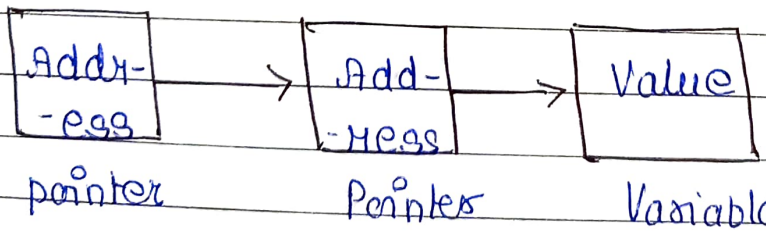
A pointer can point to the variable having same type

as the pointer Variable.

Using pointer in C :-

Pointers can be used to access the address as well as the data of variable whose address it contains. To Access address, the name of pointer Variable is Used, whereas to access the data stored in variable pointed by pointer, * is preceded with name of pointer variable. *[Asterisk] instructs the compiler that Variable being declared is a pointer variable

Pointer to Pointer [C Double Pointer] :- In C language we can also define a pointer to store the address of another pointer Such pointer is known as double pointer. The first pointer is used to store address of variable whereas second pointer is used to store address of first pointer




Syntax :-

```
int **p;
```

Example :-

```
#include<stdio.h>
Void main() {
int a = 10;
int *p;
```

```
int **pp;
p = &a;
pp = &p;
printf("Address of a = %x", p);
printf("Address of p = %x", pp);
printf("Value stored at p = %d", *p);
printf("Value stored at pp = %d", **pp);
}
```

Pointer Arithmetic in C:- C Language allows you to perform arithmetic operations on pointer. The result of an arithmetic operation performed on pointer will also be a pointer if other operator is of type integer. In pointer from pointer subtraction the result will be an integer value. We can perform following arithmetic operators on pointer in C:

1. Increment
2. Decrement
3. Addition
4. Subtraction
5. Comparison

1. Incrementing Pointer in C:- If we increment a pointer by 1, the pointer will start pointing to immediate next location. This is somewhat different from general arithmetic since value of pointer will get increased by size of the data type to which pointer is pointing.

The Rule to Increment pointer is :

new_address = current_address + i * size_of(data type)

Example :-

```
#include<stdio.h>
Void main(){
int number=50;
int *p;
p=&number;
printf("Address of p variable is %u",p);
p=p+1;
printf("After increment address of p variable is %u",p);
}
```

Traversing an array by using pointer :-

```
#include<stdio.h>
Void main()
{
  int arr[5]={1,2,3,4,5};
  int *p=arr;
  int i;
  printf("Printing array elements...");
  For(i=0;i<5;i++)
  {
    printf("%d",*(p+i));
  }
}
```

2) Decrementing Pointer in C :- We can decrement a pointer variable If we decrement a pointer, It will start pointing to previous location

Rule :-

new_address= current_address - i * size_of(data type)

Example :-

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

```
Void main(){
int number = 50;
int *p;
p = &number;
printf("Address of p variable is %u ",p);
p = p-1
printf("After decrement address of p variable is %u",p);
}
```

3.] Addition pointer in C :- we can add a value to the pointer variable the formula to add addition pointer is below :-

new_address = current_address + (number * size_of(data type))

Example :-

```
#include<stdio.h>
Void main(){
int number = 50;
int *p;
p = &number;
printf("Address of p variable is %u",p);
p = p+3;
printf("After adding 3 Address of p variable is %u \n",p);
}
```

4.] Subtraction pointer in C :- In C we can subtract a value from pointer variable subtracting any number from pointer will give an address. The formula of subtracting value from pointer variable is

new_address = current_address - (number * size of(data type))

Example :-

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

```
Void main() {
int number = 50;
int *p;
p = &number;
printf("Address of p variable is %u", p);
p = p-3
printf("after subtracting 3 address of p variable is %u", p);
}
```

Illegal arithmetic with pointers :-

Address + Address = illegal
address % address = illegal
address * address = illegal
address / address = illegal
address & address = illegal
address ^ address = illegal
address | address = illegal
~ address = illegal

Pointer to function in C :- A pointer can point to a function in C. However the declaration of pointer variable must be the same as function

Example :-

```
#include <stdio.h>
int addition();
int main()
{
  int result;
  int (*ptr)();
  ptr = &addition;
```

```
    result = (*ptr)();
    printf("The Sum is %d", result);
getch();
}
int addition()
{
    int a, b;
    printf("Enter two numbers");
    scanf("%d%d", &a, &b);
    return a+b;
}
```

Array of Pointer in C :- Pointers are also used to handle array of string or two dimensional character array. In it each row is handled be a pointer with unique index number. A single for loop is used to access the contents of array of pointer.

Example :-

```
#include<stdio.h>
#include<conio.h>
Void main() {
char* str[3] = {"Microsoft", "Crack", "Sun"};
int i;
    for (i=0; i<3; i++)
    {
        printf("The element is :");
        puts(str[i]);
    }
getch();
}
```

Pointers as Arguments :- In call by reference address of actual parameters is passed and parameters are directly manipulated without creating an copy. This is done with the help of pointers.

Program of use of pointers while calling a function :-

```
// Program to illustrate call by reference
#include <stdio.h>
#include <conio.h>
Void main() {
 int sum(int*, int*);
 int a, b, ans;
 printf("Enter two numbers");
 scanf("%od %od", &a, &b);
 ans = sum(&a, &b);
printf("\n The result after addition is %od", ans);
getch();
}
 int sum(int *j1, int *j2)
 {
   int result;
   result = *j1 + *j2;
   return (result);
 }
```

Function returning pointers :- We can also create function that return pointer variables with this method we can return multiple values from function depending on some condition.

Program Using functions returning pointers:-

```
#include<stdio.h>
#include<conio.h>
int *great(int*, int*);
void main()
{
  int a,b;
  int *ans;
  printf("\n Enter 2 numbers");
  scanf("%d%d",&a,&b);
  ans = great(&a,&b);
  printf("The biggest number is %d", *ans);
  getch();
}
int* great (int *F1, int *F2)
{
  if(*F1>*F2)
    return(F1);
  else
    return(F2);
}
```

2.7 Dynamic Memory Allocation :- Memory allocation means to reserve the block of memory for some variable. If we declare array in static declaration

Eg int a[10][10];

In this case 10x10 = 100 elements of 2 bytes each thus 200 bytes of memory is allocated to the variable a. But at run time we require 3x3 matrix which means Nine[9] elements of 2 bytes each

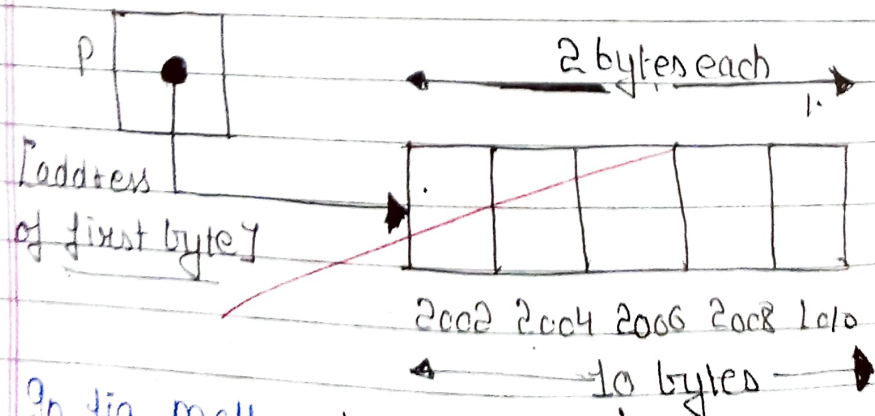
only 18 bytes are allocated So 200 - 18 = 182 bytes
It means 182 bytes are waster. So solution of this problem is to allocate memory at runtime allocation
So such type of memory allocation is called as dynamic memory allocation. In dynamic memory allocation we can allocate memory to variables during execution time. For dynamic allocation we use four functions malloc(), calloc(), free() and realoc()

4.4 malloc() :- The malloc() function reserves a block of memory of specified size and returns a pointer of specified datatype. Malloc() is dynamic memory allocation function.

Eg p=(int *)malloc(5*sizeof(int));

Here p is the pointer of type int.
(int*) tells to what type it will be pointing.
malloc is memory allocation function
5 is the number of elements to be allocated

Sizeof(int) is used to tell the size of particular datatype




In fig malloc function reserves a memory equal to 10 bytes and address is stored in p.

2.) Calloc() :- It also work like malloc() function. It allocates space for elements, initialize them to zero and then returns a pointer to the memory calloc is also a dynamic memory allocation function.

eg: p=(int*)calloc(5*sizeof(int));

In this example calloc function reserves a memory equal to a 10 bytes. It also initializes all the 5 elements to zero and returns a pointer in p.

3.) Free() function :- This function deallocates or frees a block of memory that was previously allocated. This function takes a pointer to block of memory as its argument and after its execution that block is no more availble.

Syntax :-

```
int *p;
free(p);
```

4.) realloc() function :- This function is used to modify the size of previously allocated space

eg :- if p=(int*)malloc(5);