POINTERS

POINTERS

Definition:

pointer is a variable which holds the address of other variables such as arrays, structures and other data types.

Pointer operators:

*** → value at the address**

& → address of

Pointer variable creation:

int i=10;

int *j;

Now **j** is a variable which can hold the address of integer type variable.

j=&i;

Now **j** is holding the address of i. means we can access value in i by using the pointer variable j.

How?

How?

By using the pointer operator *.

*j is equals to i now.

*j means value at the address which is hold by j. j is holding i s address.

**k means k is a pointer which can hold the another pointer address.

Sample program:

```
#include<stdio.h>
int main() {
        int i=10, *j=&i, **k=&j;
        printf("\n i value is::%d",i);
        printf("\n i value address is::%p",&i);
        printf("\n j value is ::%d",*j);
        printf("\n j address is::%p",&j);
        printf("\n j holding address is::%p",j);
        printf("\n k value is::%d",**k);
        printf("\n k address is::%p",&k);
        printf("\n k holding address is::%p\n",k);
}
```

Output:

```
i value is::10

i value address is::0xbf9a93b0

j value is ::10

j address is::0xbf9a93ac

j holding address is::0xbf9a93b0

k value is::10

k address is::0xbf9a93a8

k holding address is::0xbf9a93ac
```

Need of pointers:

- we can directly communicate with memory addresses.
- Fast accessing is possible.
- By using the pointers we can achieve the concept of Dynamic Memory allocation. (run time memory allocation).

Program on function call by value

- ❑ With the first method, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.
- ❑ Note that the values of **a** and **b** remain unchanged even after exchanging the values of **x** and **y**

```
 void swapv (int x,int y)
{
int t; t=x; x=y; y=t;
printf(“x=%d\t=%d\n”,x,y);
 }
 int main(void)
{
int a= 10,b=20;
swapv (a,b);
printf(“a=%d b=%d\n”,a,b);
return  0;
}
```

Program on function call by Reference

- ❑ In the second method(call by reference), the address of the actual arguments in the calling function are copied into the formal arguments of the called function .
- ❑ This means that ,using these addresses , we would have an access to the actual arguments and hence we would be able to manipulate them .

```
void swapr(int *x,int *y)
{
            int t;
t=*x;*x=*y;*y=t;
 printf("*x=%d\t
*y=%d\n",*x,*y);
 }
int main()
{
  int a= 10,b=20;
  swapr(&a,&b);
  printf("a=%d b=%d\n",a,b);
  return 0;
}
```

Arithmetic operations on pointers:

Performing the arithmetic operations on pointers is called pointer arithmetic.

Add a number to a pointer:

```
int i[5]={1,2,3,4,5},*j;

j=&i[0];        *j →1

j=j+1;          *j →2

j=j+3;          *j →5
```

Subtract a number and pointer to a pointer:

int i[5]={1,2,3,4,5},*j,*k;

k=&i[4];	*k → 5
j=&i[0];	*j → 1
k-j;	3
*k-*j;	4

Comparison of two pointers:

*k==*j

K==j

Not to do with pointers:

Note: - Do not attempt the following operations on pointers :

1) Adding a pointer to another pointer.

2) Multiply a constant or pointer to pointer.

3) divide a constant or pointer to pointer.

POINTERS with ARRAYS

Now we are applying pointers concept on arrays. Means suppose

```
int a[]={1,2,3,4,5};
Int *p=&a[0];
```

Means *p is holding base address of array a.
array values are storing in continuous memory locations.
So, if add 1 to the p then it automatically pointes to the next index value.

```
int a[]={1,2,3,4,5};
int *p=&a[0];

*(p+0)  equals to  a[0]     value 1
*(p+1)  equals to  a[1]      value 2
*(p+2)  equals to  a[2]      value 3
*(p+3)  equals to  a[3]      value 4
*(p+4)  equals to  a[4]      value 5
```

We can access array elements by using pointer. If pointer points base address of an array.

Sample program:-

```
#include<stdio.h>
main()
{
        int a[20],i;
        int *p=&a[0],size;
        printf("\n how many elements you are going to enter::");
        scanf("%d",&size);
        printf("\n enter array elements::\n");
        for(i=0;i<size;i++)
        scanf("%d",(p+i));
        printf("\n displaying array elements by using the pointer::\n");
        for(i=0;i<size;i++)
        printf("%d\t",*(p+i));
        printf("\n");
}
```

Accessing 2D array elements by using pointer:

```
main(){
        int a[10][10],i,j,*p=&a[0][0],r,c;
        printf("\n enter 2D array size::");
        scanf("%d%d",&r,&c);
        printf("\n enter array elements::\n");
        for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        scanf("%d",(p+i*10+j));
        printf("\n displaying array elements by using the pointer::\n");
        for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        printf("%d\t",*(p+i*10+j));
}
```

Out put:

```
enter 2D array size::   2         2
enter array elements::
1         2         3         4

display array elements normally::
1       3214705204
2       3214705208
3       3214705244
4       3214705248

displaying array elements by using the pointer::
1       2       3       4
```

Passing 1D array by using pointers:

\programs\pointers\poi1dfun.c

Passing 2D array by using pointers:

programs\pointers\poi2dfun.c