POINTERS

Pointers to Variables

- A pointer is a variable that stores an address of another variable of same type.
- Pointer can have any name that is legal for other variable.
- Pointer variables are declared with prefix of '*' operator.
- Using a pointer variable, we can access the value of another variable assigned to it.

Syntax

*data_type *pointer_name;*

Example

*int *a;*

- variable *a can store the address of any integer type variable.
- A pointer is a variable whose value is also an address.
- Each variable has two attributes
  - ✓ Value
  - ✓ Address

We can define pointers in two ways.

i) First a pointer is a variable and assigns different values to a pointer variable.

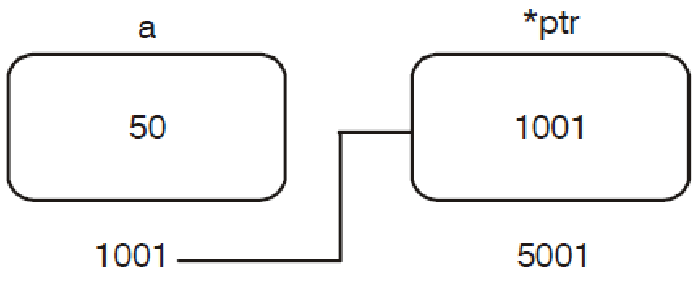
ii) Second the value contained by a pointer must be an address which indicates the location of another variable in the memory. So, pointer is called as "address variable".

Example

int a=50;

*int *ptr;*

ptr=&a;



- Here ‘a’ is a variable holds a value 50 and stored in a memory location 1001. ‘*ptr’ is pointer variable holds a address of a variable ‘a’.

Advantages of Using Pointers

- Pointers are more compact and efficient code.
- Pointers can be used to achieve clarity and simplicity.
- Pointers are used to pass information between function and its reference point.
- A pointer provides a way to return multiple data items from a function using its function arguments.
- Pointers also provide an alternate way to access an array element.
- A pointer enables us to access the memory directly.

Example Program 2.10

/*C program for printing value and address of a variable using pointer variable*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int  i=3;
int *ptr;
ptr=&i;
clrscr();
printf("Address of i=%u\n",ptr);
printf("value of i=%d\n",*ptr);
getch();
}
```

Output:

```
Address of i=65524
value of i=3
```

Example Program 2.11

/*C program for printing value and address of a variable using pointer variable by various methods*/

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

```
#include<conio.h>
void main()
{
int i=4;
int *j;
j=&i;
clrscr();
printf("Address of i=%u\n",&i);
printf("Address of i=%u\n",j);
printf("Address of j=%u\n",&j);
printf("value of j=%u\n",j);
printf("value of i=%d\n",i);
printf("value of i=%d\n",*(&i));
printf("value of i=%d\n",*j);
getch();
}
```

Output

```
Address of i=65524
Address of i=65524
Address of j=65522
value of j=65524
value of i=4
value of i=4
value of i=4
```

Example Program 2.12

/*C program to add two numbers using pointers*/

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

```
void main()
{
int a,b,*p,*q,sum;
clrscr();
printf("Enter two integers");
scanf("%d %d",&a,&b);
p=&a;
q=&b;
sum=*p+*q;
printf("sum=%d",sum);
getch();
}
```

Output

```
Enter two integers 2 3
sum=5
```

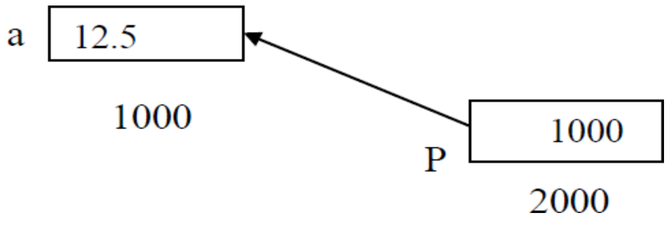
Pointer operators

a) Referencing a pointer

- A pointer variable is made to refer to an object.
- Reference operator(&) is used for this.
- Reference operator is also known as address of (&) operator.

Example

```
float a=12.5;
float *p;
p=&a;
```



b) Dereferencing a pointer

- The object referenced by a pointer can be indirectly accessed by dereferencing the pointer.
- Dereferencing operator (*) is used for this.
- This operator is also known as indirection operator or value- at-operator.

Example

```
int b;
int a=12;
a int *p;
```

Example program 2.13

```
#include<stdio.h>
void main()
{
int a=12;
int *p;
int **pptr;
p=&a;
pptr=&p;
printf("Value=%d",a);
printf("value by dereferencing p is %d \n",*p);
printf("value by dereferencing pptr is %d \n",**pptr);
printf("value of p is %u \n",p);
printf("value of pptr is %u\n",pptr);
}
```

Output

```
Value=12
value by dereferencing p is 12
value by dereferencing pptr is 12
value of p is 1000
value of pptr is 2000
```

Arrays and pointers

- Array elements are always stored in consecutive memory locations according to the size of the array.
- The size of the variable with the pointer variables refers to, depends on the data type pointed by the pointer.
- A pointer when incremented, always points to a location after skipping the number of bytes required for the data type pointed to by it.

  Example

  int a[5]={10,20,30,40,50};

  a[5] means the array 'a' has 5 elements and of integer data type

Program 2.14

/*C program to print the value and address of an array elements*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50};
int i;
clrscr();
for(i=0;i<5;i++)
{
printf("The value of a[%d]=%d\n",i,a[i]);
printf("Address of a[%d]=%u\n",i,&a[i]);
}
getch();
}
```

Output

```
The value of a[0]=10
Address of a[0]=4000
```

```
The value of a[1]=20
Address of a[1]=4002
The value of a[2]=10
Address of a[2]=4004
The value of a[3]=10
Address of a[3]=4006
The value of a[4]=10
Address of a[4]=4008
```

Example Program 2.15

/*C program to print the value and address of an array elements using pointer*/

```
#include<stdio.h>
#include<conio.h>
void main()
    {
int arr[5]={10,20,30,40,50};
int i,*p;
p=arr;
clrscr();
for(i=0;i<=5;i++)
{
printf("\nAddress=%u\t",(p+i));
printf("Element=%d",*(p+i));
}
getch();
}
```

Output

```
Address 4000 Element=10
Address 4002 Element=20
Address 4004 Element=30
Address 4006 Element=40
Address 4008 Element=50
```

Example Program 2.16

/*C program to add sum of elements of an array using pointer*/

```
#include<stdio.h>
main()
{
int i,sum;
int arr[5];
int *ptr;
for(i=0;i<5;i++)
{
printf (“Enter the number”);
scanf(“%d”,&arr[i]);
}
ptr=arr;
for(i=0;i<5;i++)
{
sum=sum+*ptr
Functions and Pointers 3.29
ptr=ptr+1;
}
printf(“Total=%d”,sum);
}
```

Output

```
Enter the number
10
20
```

```
30
40
50
Total= 150
```

Pointers with Multi-Dimensional Array

- A multi-dimensional array can also be represented with an equivalent pointer notation. A two dimensional array can be considered as a collection of one-dimensional arrays.

  Syntax

  *data_type (*pointer variable) [expression];*

  data_type array name[expression 1][expression 2];

Example Program 2.17

/*C program to print the value and address of the element using array of pointers*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int * int *a[3];
int b=10,c=20,d=30,i;
a[0]=&b;
a[1]=&c;
a[2]=&d;
clrscr();
for(i=0;i<3;i++)
{
printf("Address=%u\n",a[i]);
printf("Value=%d\n",*(a[i]));
}
getch();
}
```

Output

```
Address=4000
Value=10
Address=5000
Value=20
Address=6000
Value=30
```

Functions Pointers

- Function pointers in C can be used to create function calls to which they point. This allows programmers to pass them to functions as arguments. Such functions passed as an argument to other functions are also called callback functions.
- In C programming, it is also possible to pass addresses as arguments to functions. To accept these addresses in the function definition, we can use pointers. It's because pointers are used to store addresses.

Example Program 2.18

Write a C Program for Swapping of two numbers using function pointers.

```
#include <stdio.h>
void swap(int *n1, int *n2);
int main()
{
   int num1 = 5, num2 = 10;
   // address of num1 and num2 is passed
   swap( &num1, &num2);
   printf("num1 = %d\n", num1);
   printf("num2 = %d", num2);
   return 0;
}
void swap(int* n1, int* n2)
{
   int temp;
   temp = *n1;
   *n1 = *n2;
   *n2 = temp;
}
```

Output

```
num1 = 10
num2 = 5
```

- The address of num1 and num2 are passed to the swap() function using swap(&num1, &num2);
- When *n1 and *n2 are changed inside the swap() function, num1 and num2 inside the main() function are also changed.
- Inside the swap() function, *n1 and *n2 swapped. Hence, num1 and num2 are also swapped.