POINTERS

Pointer is a variable that stores the address of a variable or a function. Pointer is a derived data type. It is declared same as other variables but prior to variable name an asterisk „*‟operator is used.

Benefits of Pointers

- Reduce the length of the program.
- Support dynamic memory management.
- Pointers save memory space.
- More efficient in handling arrays.
- Execution speed increased.
- Used to return multiple values from a function.

Declaring a Pointer

The pointer variable is declared by using * operator.

syntax

datatype * pointervariable;

Example:

```
char *ptr;
float *P1;
```

Pointer Initialization

The process of assigning the address of a variable to a pointer variable is known as initialization. This is done through (&) ampersand operator.

Syntax:

```
Pointer variable = & variablename;
```

Example:

```
int n, *p;
 p = &n;
```

POINTER OPERATORS

Pointer is a variable that stores the address of a variable or a function. There are 2 pointer operators in C. They are,

1. Referencing operator
2. Dereferencing operator

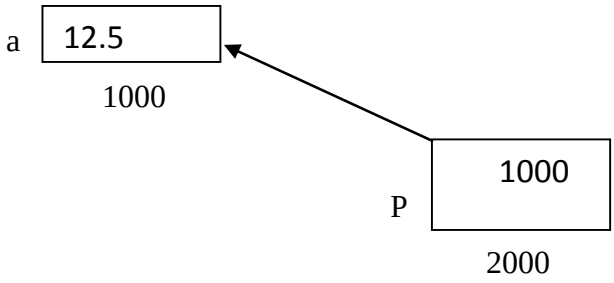
Operator	Operator Name	Purpose
*	Dereferencing Operator **OR** indirection operator	Gives Value stored at Particular address
&	Referencing Operator **OR** address of (&) operator	Gives Address of Variable

Pointer Operators

Referencing operator: (&)

& operator is a unary operator that returns the memory address of its operand. It is used to find the address of variable For example, if var is an integer variable, then &var is its address. Reference operator is also known as address of (&) operator.

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

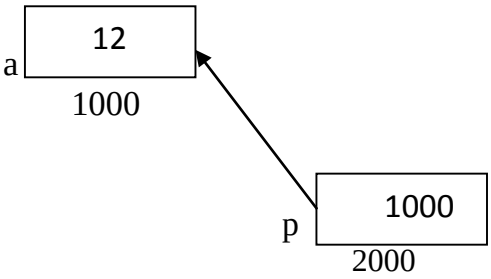


Pointer P Referencing the variable a

Dereferencing operator

The second operator is indirection Operator *, and it is the complement of &. It is a unary operator that returns the value of the variable located at the address specified by its operand. This operator is also known as indirection operator or value- at-operator

Example




```
int b;
int a=12;
int *p;
p=&a;
b=*p;     \\value pointed by p(or)value
                at 1000=12,
so b=12
```

Pointer P Dereferencing the variable a

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int Mark = 73,
        int *Ptr;
        Ptr = &Mark;
        printf("The value of the variable Mark = %d \n", *Ptr);
        printf("The address of the variable Mark = %d \n", Ptr);
        getch();
}
```

Output:

The value of the variable Mark = 73

The address of the variable Mark = 4082

For every execution time the compiler may allot the different memory. So, the address of the variable may be different when we execute the program next time.

NULL Pointer

The pointer variable that is assigned NULL value is called Null pointer. NULL pointer is a special pointer that does not point anywhere. It does not hold the address of any object or function. It has numeric value 0

Example: for Null Pointer

```
int *ptr = 0;
        ( or )
int *ptr = NULL;
```

Use of null pointer:

- As an error value
- As a sentinel value
- To stop indirection in a recursive strucure

POINTER ARITHMETIC

Pointer is a variable that stores the address of a variable or a function.

Valid operation

- Pointer can be added with a constant
- Pointer can be subtracted with a Constant
- Pointer can be Incremented or Decremented

Not Valid operation

- Two pointers cannot be added
- Two pointers cannot be subtracted
- Two pointers cannot be multiplied
- Two pointers cannot be divided

Example:

```
 int a=10
int *p=&a;
```

p=p+1;

- The pointer holds the address 2000. This value is added with 1.
- The data type size of the constant is added with the address. p= 2000+(2*1)=2002

The following table shows the pointer arithmetic.

S.no	Operator	Type of operand 1	Type of operand 2	Result type	Example	Initial value	Final value	Description
1	Addition (+)	Pointer to type T	int	Pointer to type T				Result = initial value of ptr +int operand * sizeof (T)
	Eg.	int *	int	int *	p=p+5	p=2000	2010	2000+5*2= 2010
2	Increment (++)	Pointer to type T	-	Pointer to type T				**Post increment** Result = initial value of pointer **Pre-increment** Result = initial value of pointer + sizeof (T) Value of ptr = Value of ptr +sizeof(T)
	Eg. post increment	float*	-	float*	ftr=p++	ftr=? p=2000	ftr=2000 p=2004	
3	Subtraction -	Pointer to type T	int	Pointer to type T				Result = initial value of ptr - int operand * sizeof (T)
	E.g.	float*	int	float*	p=p-1	p=2000	1996	2000 – 1 * 4 = 2000-4=1996

4	decrement	Pointer to type T	-	Pointer to type T				**Post decrement** Result = initial value of pointer **Pre-decrement** Result = initial value of pointer – sizeof(T)
	Eg.pre decrement	float*	-	float*	ftr=--p	ftr=? p=2000	ftr=1996 p=1996	Value of ptr = Value of ptr – sizeof(T)

Pointer Arithmetic

Program : Addition of Integers with Pointer

```
#include<stdio.h>
#include<conio.h>
void main ()
{
        int a[5] = {10, 5, 20, 5, 2};
        int i, sum = 0
        for (i = 0; i < 5; i++)
                sum = sum+*(a + i);
        printf ("Total = %d", sum);
        getch();
}
```

Output:

Total = 42

Program : Subtraction of Pointers

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

```
#include<conio.h>
void main ()
{
        double a[2], *p, *q;
        p = a;     // Assign a[0] address to p
        q = p + 1;    // Assign a[1] address to q
        printf("No. Of elements between p & q = %d", q - p);
        getch();
}
```

Output:

No. Of elements between p & q = 2