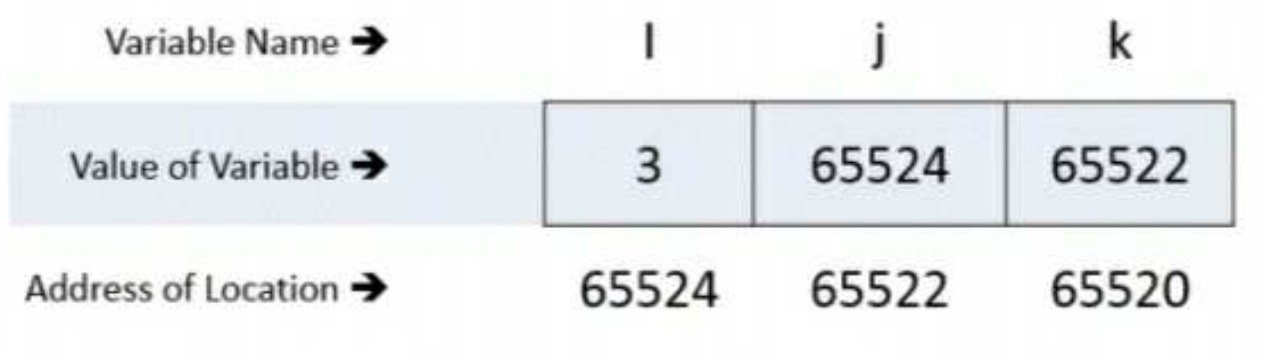
**POINTERS**

Pointer Overview




Consider above Diagram which clearly shows pointer concept in c programming –

**i** is the name given for particular memory location of ordinary variable.

Let us consider it's Corresponding address be 65624 and the Value stored in variable **'i'** is 5

The address of the variable **'i'** is stored in another integer variable whose name is **'j'** and which is having corresponding address 65522

thus we can say that –

j = &i;

i.e

j = Address of i

Here j is not ordinary variable , It is special variable and called pointer variable as it stores the address of the another ordinary variable. We can summarize it like –

| Variable Name | Variable Value | Variable Address |
|---|---|---|
| i | 5 | 65524 |
| j | 65524 | 65522 |

B. C Pointer Basic Example:

```
#include <stdio.h>

int main()
{
 int *ptr, i;
 i = 11;

 /* address of i is assigned to ptr */
```

```
 ptr = &i;

 /* show i's value using ptr variable */
 printf("Value of i : %d", *ptr);

 return 0;
}
```

See Output and Download »

You will get value of i = 11 in the above program.

C. Pointer Declaration Tips :

1. Pointer is declared with preceding * :

```
int *ptr;  //Here ptr is Integer Pointer Variable
int ptr;   //Here ptr is Normal  Integer Variable
```

2. Whitespace while Writing Pointer :

pointer variable name and asterisk can contain whitespace because whitespace is ignored by compiler.

```
int *ptr;
int      * ptr;
int *          ptr;
```

All the above syntax are legal and valid. We can insert any number of spaces or blanks inside declaration. We can also split the declaration on multiple lines.

D. Key points for Pointer :

Unline ordinary variables pointer is special type of variable which stores the address of ordinary variable.

Pointer can only store the whole or integer number because address of any type of variable is considered as integer.

It is good to initialize the pointer immediately after declaration

& symbol is used to get address of variable

* symbol is used to get value from the address given by pointer.

E. Pointer Summary :

Pointer is Special Variable used to Reference and de-reference memory. (*Will be covered in upcoming chapter)

When we declare integer pointer then we can only store address of integer variable into that pointer.

Similarly if we declare character pointer then only the address of character variable is stored into the pointer variable.

| Pointer storing the address of following DT | Pointer is called as |
|---|---|
| Integer | Integer Pointer |
| Character | Character Pointer |
| Double | Double Pointer |
| Float | Float Pointer |

Pointer is a variable which stores the address of another variable

Since Pointer is also a kind of variable , thus pointer itself will be stored at different memory location.

2 Types of Variables :

Simple Variable that stores a value such as integer,float,character

Complex Variable that stores address of simple variable i.e pointer variables

Simple Pointer Example #1 :

```
#include<stdio.h>

int main()
{
int a = 3;
int *ptr;
ptr = &a;

return(0);
}
```

Explanation of Example :

| Point | Variable 'a' | Variable 'ptr' |
|---|---|---|
| Name of Variable | a | ptr |

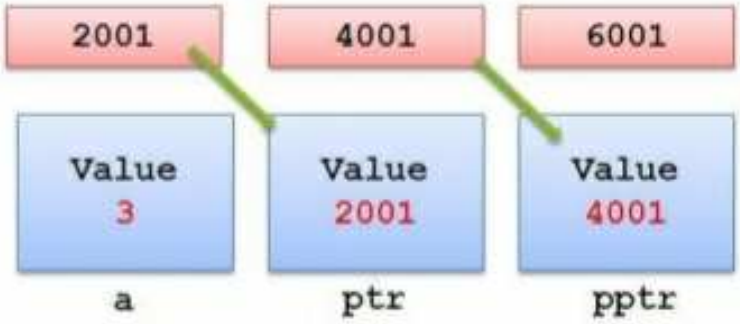
| Point | Variable 'a' | Variable 'ptr' |
|---|---|---|
| Type of Value that it holds | Integer | Address of Integer 'a' |
| Value Stored | 3 | 2001 |
| Address of Variable | 2001 (Assumption) | 4001 (Assumption) |

Simple Pointer Example #2 :

```
#include<stdio.h>

int main()
{
int a = 3;
int *ptr,**pptr;
ptr = &a;
pptr = &ptr;
return(0);
}
```

Explanation of Example

With reference to above program –




We have following associated points –

| Point | Variable 'a' | Variable 'ptr' | Variable 'pptr' |
|---|---|---|---|
| Name of Variable | a | ptr | pptr |
| Type of Value that it holds | Integer | Address of 'a' | Address of 'ptr' |
| Value Stored | 3 | 2001 | 4001 |

| Point | Variable 'a' | Variable 'ptr' | Variable 'pptr' |
|---|---|---|---|
| Address of Variable | 2001 | 4001 | 6001 |

Pointer address operator in C Programming

Pointer address operator is denoted by '&' symbol

When we use ampersand symbol as a prefix to a variable name '&', it gives the address of that variable.

lets take an example –

&n - It gives an address on variable n

Working of address operator

```
#include<stdio.h>
void main()
{
int n = 10;
printf("\nValue of n is : %d",n);
printf("\nValue of &n is : %u",&n);
}
```

**Output :**

Value of n is : 10

Value of &n is : 1002

Consider the above example, where we have used to print the address of the variable using ampersand operator.

In order to print the variable we simply use name of variable while to print the address of the variable we use ampersand along with %u

printf("\nValue of &n is : %u",&n);

Understanding address operator

Consider the following program –

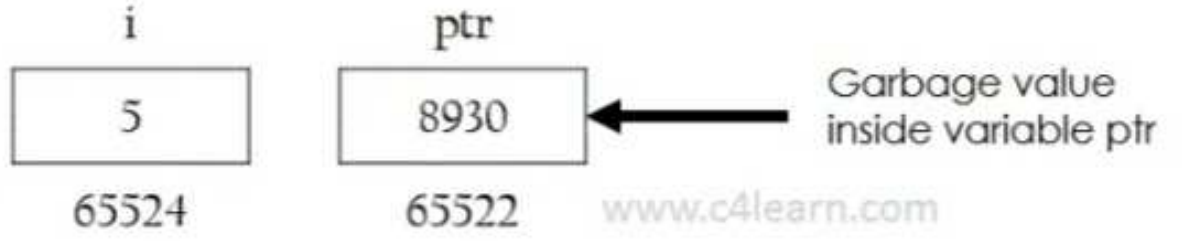
```
#include<stdio.h>
int main()
{
int i = 5;
int *ptr;
```

```
ptr = &i;

printf("\nAddress of i    : %u",&i);
printf("\nValue of ptr is : %u",ptr);

return(0);
}
```

After declaration memory map will be like this –

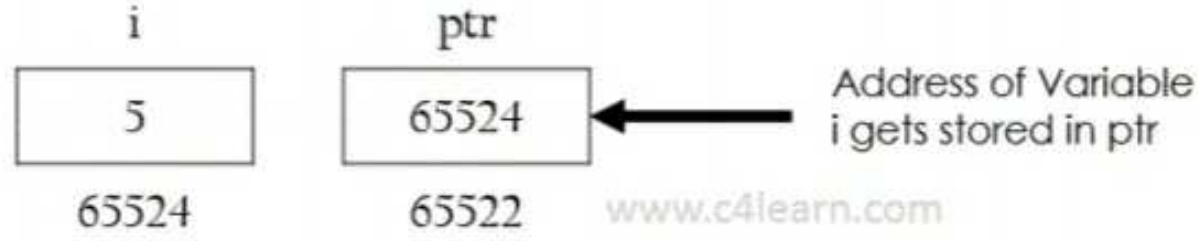
int i = 5;

int *ptr;




after Assigning the address of variable to pointer , i.e after the execution of this statement –

ptr = &i;




Invalid Use of pointer address operator

Address of literals

In C programming using address operator over literal will throw an error. We cannot use address operator on the literal to get the address of the literal.

&75

Only variables have an address associated with them, constant entity does not have corresponding address. Similarly we cannot use address operator over character literal –

&('a')

Character 'a' is literal, so we cannot use address operator.

Address of expressions

(a+b) will evaluate addition of values present in variables and output of (a+b)is nothing but Literal, so we cannot use Address operator

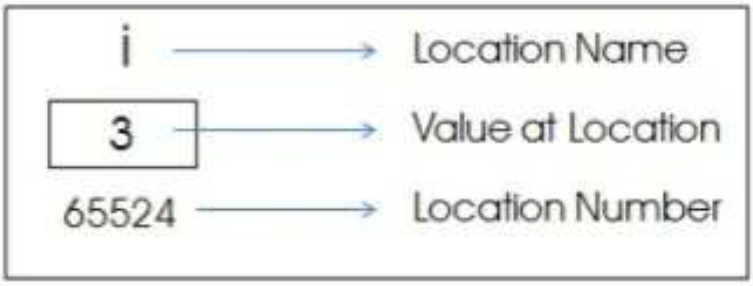
&(a+b)

Memory Organization for Pointer Variable:

When we use variable in program then Compiler keeps some memory for that variable depending on the **data type**

The address given to the variable is Unique with that variable name

When Program execution starts the **variable name** is automatically translated into the corresponding **address**.




Explanation :

Pointer Variable is nothing but a memory address which holds another address .

In the above program "i" is name given for memory location for human understanding , but compiler is unable to recognize "i" . Compiler knows only address.

In the next chapter we will be learning , Memory requirement for storing pointer variable.

Syntax for Pointer Declaration in C :

data_type *<pointer_name>;

Explanation :

data_type

Type of variable that the pointer **points to**

OR data type whose address is stored in **pointer_name**

Asterisk(*)

Asterisk is called as **Indirection Operator**

It is also called as **Value at address Operator**

It Indicates **Variable declared is of Pointer type**

pointer_name

Must be any **Valid C identifier**

Must follow all Rules of Variable name declaration

Ways of Declaring Pointer Variable:

[box] * can appears anywhere between Pointer_name and Data Type

```
int *p;
int *       p;
int    *  p;
```

Example of Declaring Integer Pointer:

int n = 20;

int *ptr;

Example of Declaring Character Pointer:

char ch = 'A';

char *cptr;

Example of Declaring Float Pointer:

float fvar = 3.14;

float *fptr;

How to Initialize Pointer in C Programming?

pointer = &variable;

Above is the syntax for initializing pointer variable in C.

Initialization of Pointer can be done using following 4 Steps :

Declare a Pointer Variable and Note down the Data Type.

Declare another Variable with Same Data Type as that of Pointer Variable.

Initialize Ordinary Variable and assign some value to it.

Now Initialize pointer by assigning the address of ordinary variable to pointer variable.

below example will clearly explain the initialization of Pointer Variable.

```
#include<stdio.h>
int main()
{

int a;      // Step 1
int *ptr;   // Step 2
```

```
a = 10;     // Step 3
ptr = &a;   // Step 4

return(0);
}
```

Explanation of Above Program :

Pointer should not be used before initialization.

"ptr" is pointer variable used to store the address of the variable.

Stores address of the variable **'a'** .

Now "ptr" will contain the address of the variable "a" .

Note :

[box]Pointers are always initialized before using it in the program[/box]

Example : Initializing Integer Pointer

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

ptr = &a;
printf("\nValue of ptr : %u",ptr);

return(0);
}
```

**Output :**

Value of ptr : 4001

**Pointer arithematic**

Incrementing Pointer:

Incrementing Pointer is generally used in array because we have contiguous memory in array and we know the contents of next memory location.

Incrementing Pointer Variable Depends Upon data type of the Pointer variable

Formula : ( After incrementing )

new value = current address + i * size_of(data type)

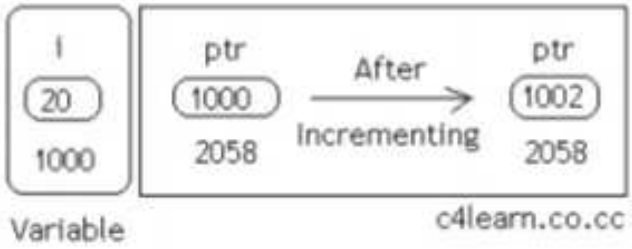
Three Rules should be used to increment pointer –

Address + 1 = Address

Address++ = Address

++Address = Address

**Pictorial Representation :**




| Data Type | Older Address stored in pointer | Next Address stored in pointer after incrementing (ptr++) |
|---|---|---|
| int | 1000 | 1002 |
| float | 1000 | 1004 |
| char | 1000 | 1001 |

Explanation : Incremeting Pointer

Incrementing a pointer to an integer data will cause its **value to be incremented by 2** .

This differs from compiler to compiler as memory required to store integer **vary compiler to compiler**

[box]**Note to Remember :** Increment and Decrement Operations on pointer should be used when we have Continues memory (in Array).[/box]

Live Example 1 : Increment Integer Pointer

```
#include<stdio.h>

int main(){

int *ptr=(int *)1000;

ptr=ptr+1;
printf("New Value of ptr : %u",ptr);
```

```
return 0;
}
```

**Output :**

New Value of ptr : 1002

Live Example 2 : Increment Double Pointer

```
#include<stdio.h>

int main(){

double *ptr=(double *)1000;

ptr=ptr+1;
printf("New Value of ptr : %u",ptr);

return 0;
}
```

**Output :**

New Value of ptr : 1004

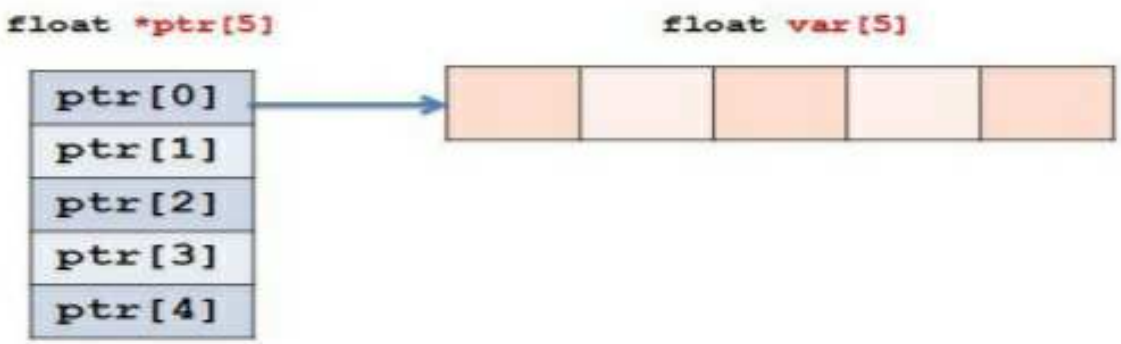
Live Example 3 : Array of Pointer

```
#include<stdio.h>

int main(){

float var[5]={1.1f,2.2f,3.3f};
float(*ptr)[5];

ptr=&var;
printf("Value inside ptr : %u",ptr);

ptr=ptr+1;
printf("Value inside ptr : %u",ptr);

return 0;
}
```

**Output :**

Value inside ptr : 1000

Value inside ptr : 1020




Explanation :

Address of ptr[0] = 1000

We are storing Address of float array to ptr[0]. –

Address of ptr[1]

= Address of ptr[0] + (Size of Data Type)*(Size of Array)

= 1000 + (4 bytes) * (5)

= 1020

**Address of Var[0]...Var[4] :**

Address of var[0] = 1000

Address of var[1] = 1004

Address of var[2] = 1008

Address of var[3] = 1012

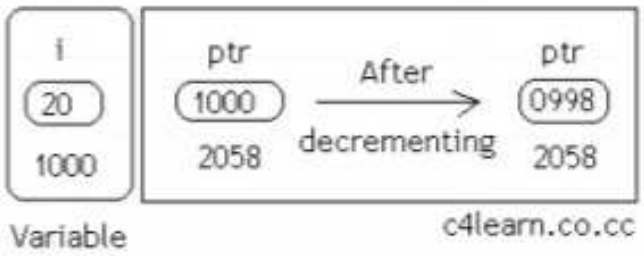
Address of var[4] = 1016

Formula : ( After decrementing )

new_address = (current address) - i * size_of(data type)

[box]Decrementation of Pointer Variable Depends Upon : data type of the Pointer variable[/box]

Example :



| Data Type | Older Address stored in pointer | Next Address stored in pointer after incrementing (ptr–) |
|---|---|---|
| int | 1000 | 0998 |
| float | 1000 | 0996 |
| char | 1000 | 0999 |

Explanation:

**Decrementing a pointer** to an integer data will cause its value to be decremented by 2

This differs from compiler to compiler as memory required to store integer vary **compiler to compiler**

Pointer Program: Difference between two integer Pointers

```
#include<stdio.h>

int main(){

float *ptr1=(float *)1000;
float *ptr2=(float *)2000;

printf("\nDifference : %d",ptr2-ptr1);

return 0;
}
```

**Output :**

Difference : 250

Explanation :

Ptr1 and Ptr2 are two pointers which holds memory address of Float Variable.

Ptr2-Ptr1 will gives us number of floating point numbers that can be stored.

ptr2 - ptr1 = (2000 - 1000) / sizeof(float)

= 1000 / 4

= 250

Live Example 2:

```
#include<stdio.h>

struct var{
```

```
    char cvar;
    int ivar;
    float fvar;
};

int main(){

struct var *ptr1,*ptr2;

ptr1 = (struct var *)1000;
ptr2 = (struct var *)2000;

printf("Difference= %d",ptr2-ptr1);

return 0;
}
```

Output :

Difference = 142

Explanation :

ptr2-ptr1 = (2000 - 1000)/Sizeof(struct var)

= 1000 / (1+2+4)

= 1000 / 7

= 142

Adding integer value with Pointer

In C Programming we can add any integer number to Pointer variable. It is perfectly legal in c programming to add integer to pointer variable.

In order to compute the final value we need to use following formulae :

final value = (address) + (number * size of data type)

Consider the following example –

```
int *ptr , n;

ptr = &n ;

ptr = ptr + 3;
```

Live Example 1 : Increment Integer Pointer

```
#include<stdio.h>

int main(){

int *ptr=(int *)1000;
```

```
ptr=ptr+3;
printf("New Value of ptr : %u",ptr);

return 0;
}
```

Output :

New Value of ptr : 1006

Explanation of Program :

In the above program –

int *ptr=(int *)1000;

this line will store 1000 in the pointer variable considering 1000 is memory location for any of the integer variable.

Formula :

ptr = ptr + 3 * (sizeof(integer))

= 1000 + 3 * (2)

= 1000 + 6

= 1006

Similarly if we have written above statement like this –

float *ptr=(float *)1000;

then result may be

ptr = ptr + 3 * (sizeof(float))

= 1000 + 3 * (4)

= 1000 + 12

= 1012

Suppose we have subtracted "n" from pointer of any data type having initial addess as "init_address" then after subtraction we can write –

ptr = initial_address - n * (sizeof(data_type))

Subtracting integer value with Pointer

int *ptr , n;

ptr = &n ;

ptr = ptr - 3;

Live Example 1 : Decrement Integer Pointer

```
#include<stdio.h>

int main(){

int *ptr=(int *)1000;

ptr=ptr-3;
printf("New Value of ptr : %u",ptr);

return 0;
}
```

Output :

New Value of ptr : 994

Formula :

ptr = ptr - 3 * (sizeof(integer))

= 1000 - 3 * (2)

= 1000 - 6

= 994

Summary :

Pointer - Pointer = Integer

Pointer - Integer = Pointer

Differencing Pointer in C Programming Language :

Differencing Means **Subtracting two Pointers**.

Subtraction gives the Total number of objects between them .

Subtraction indicates "How apart the two Pointers are ?"

C Program to Compute Difference Between Pointers :

```
#include<stdio.h>

int main()
{
```

```
int num , *ptr1 ,*ptr2 ;

ptr1 = &num ;
ptr2 = ptr1 + 2 ;

printf("%d",ptr2 - ptr1);

return(0);
}
```

Output :

2

ptr1 stores the **address of Variable** num

Value of ptr2 is incremented by **4 bytes**

Differencing two Pointers

Important Observations :

Suppose the Address of Variable num = 1000.

| Statement | Value of Ptr1 | Value of Ptr2 |
|---|---|---|
| int num , *ptr1 ,*ptr2 ; | Garbage | Garbage |
| ptr1 = &num ; | 1000 | Garbage |
| ptr2 = ptr1 + 2 ; | 1000 | 1004 |
| ptr2 - ptr1 | 1000 | 1004 |

Computation of Ptr2 – Ptr1 :

Remember the following formula while computing the difference between two pointers –

Final Result = (ptr2 - ptr1) / Size of Data Type

Step 1 : Compute Mathematical Difference (Numerical Difference)

ptr2 - ptr1 = Value of Ptr2 - Value of Ptr1

= 1004 - 1000

= 4

Step 2 : Finding Actual Difference (Technical Difference)

Final Result = 4 / Size of Integer

= 4 / 2

= 2

Numerically Subtraction ( ptr2-ptr1 ) differs by 4

As both are Integers they are numerically Differed by 4 and Technically by 2 objects

Suppose Both pointers of float the they will be differed numerically by 8 and Technically by 2 objects

Consider the below statement and refer the following table –

int num = ptr2 - ptr1;

and

| If Two Pointers are of Following Data Type | Numerical Difference | Technical Difference |
|---|---|---|
| Integer | 2 | 1 |
| Float | 4 | 1 |
| Character | 1 | 1 |

Comparison between two Pointers :

**Pointer comparison is Valid** only if the **two pointers are Pointing to same array**

All Relational Operators can be used for comparing pointers of **same type**

**All Equality and Inequality Operators** can be used with all Pointer types

Pointers **cannot be Divided or Multiplied**

Point 1 : Pointer Comparison

```
#include<stdio.h>

int main()
{
int *ptr1,*ptr2;

ptr1 = (int *)1000;
ptr2 = (int *)2000;

if(ptr2 > ptr1)
  printf("Ptr2 is far from ptr1");

return(0);
}
```

Pointer Comparison of Different Data Types :

```
#include<stdio.h>

int main()
{
int *ptr1;
float *ptr2;

ptr1 = (int *)1000;
ptr2 = (float *)2000;

if(ptr2 > ptr1)
  printf("Ptr2 is far from ptr1");

return(0);
}
```

Explanation :

**Two Pointers of different data types can be compared** .

In the above program we have compared two pointers of different data types.

It is perfectly **legal in C Programming**.

[box]As we know Pointers can store Address of any data type, address of the data type is "Integer" so we can compare address of any two pointers although they are of different data types.[/box]

Following operations on pointers :

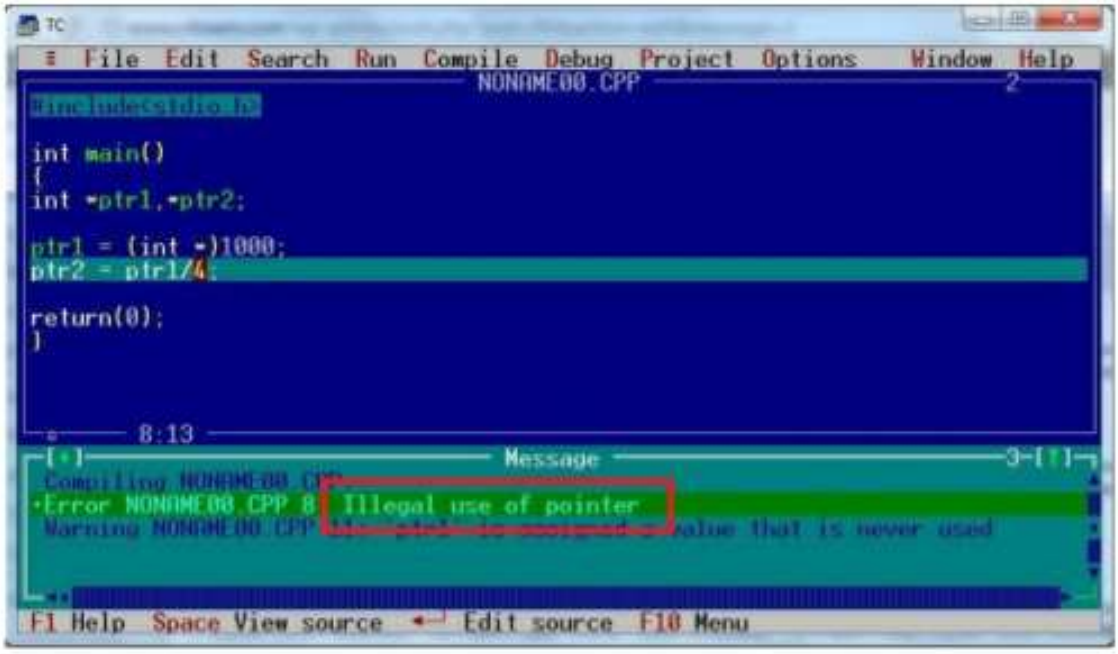
| | |
|---|---|
| > | Greater Than |
| < | Less Than |
| >= | Greater Than And Equal To |
| <= | Less Than And Equal To |
| == | Equals |
| != | Not Equal |

Divide and Multiply Operations :

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

```
int main()
{
int *ptr1,*ptr2;

ptr1 = (int *)1000;
ptr2 = ptr1/4;

return(0);
}
```

**Output :**




Pointer to pointer

Pointer to Pointer in C Programming

**Declaration : Double Pointer**

int **ptr2ptr;

**Consider the Following Example :**

int num = 45 , *ptr , **ptr2ptr ;

ptr = #

ptr2ptr = &ptr;

What is Pointer to Pointer ?

Double (**) is used to denote the **double Pointer**

Pointer Stores the address of the Variable

Double Pointer **Stores the address of the Pointer Variable**


| Statement | What will be the Output ? |
|---|---|
| *ptr | 45 |
| **ptr2ptr | 45 |
| ptr | &n |
| ptr2ptr | &ptr |

**Notes :**

Conceptually we can have Triple ….. n pointers

Example : *****n,****b can be another example

**Live Example :**

```
#include<stdio.h>

int main()
{
int num = 45 , *ptr , **ptr2ptr ;
ptr    = &num;
ptr2ptr = &ptr;

printf("%d",**ptr2ptr);
```

```
return(0);
}
```
**Output :**

45