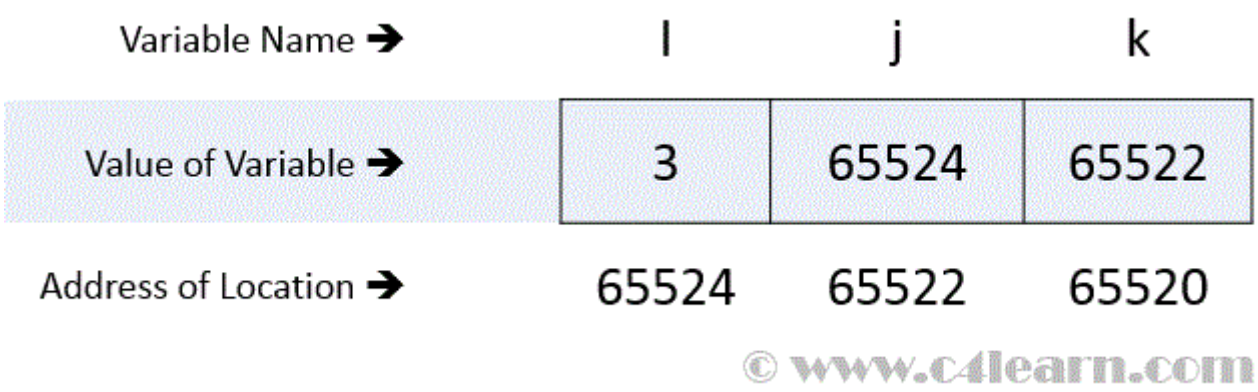
A. C Programming Pointer Overview :



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

1. **i** is the name given for particular memory location of ordinary variable.
2. Let us consider it's Corresponding address be 65624 and the Value stored in variable **'i'** is 5
3. 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;
}
```

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 :

1. Unline ordinary variables pointer is special type of variable which stores the address of ordinary variable.
2. Pointer can only store the whole or integer number because address of any type of variable is considered as integer.
3. It is good to initialize the pointer immediately after declaration
4. & symbol is used to get address of variable
5. * symbol is used to get value from the address given by pointer.

E. Pointer Summary :

1. Pointer is Special Variable used to Reference and de-reference memory. (*Will be covered in upcoming chapter)
2. When we declare integer pointer then we can only store address of integer variable into that pointer.
3. 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

C pointer basic concept

Understanding pointers in c ?

In the previous chapter we have learnt about Basic Concept of Pointer. In this chapter we are looking more into pointer.

We know that –

1. Pointer is a variable which stores the address of another variable
2. Since Pointer is also a kind of variable , thus pointer itself will be stored at different memory location.

2 Types of Variables :

1. Simple Variable that stores a value such as integer,float,character
2. 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 :

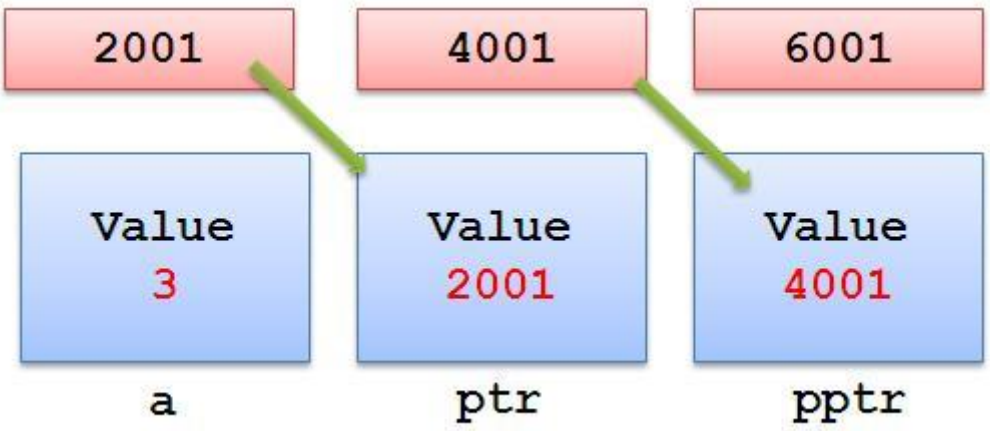
Point	Variable 'a'	Variable 'ptr'
Name of Variable	a	ptr
Type of Value that it holds	Integer	Address of Integer 'a'
Value Stored	3	2001
Address of Variable	2001 (Assumption)	4001 (Assumption)

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
Address of Variable	2001	4001	6001

C pointer address operator

In the previous chapter we have learnt about more c programming basic pointer concept. In this chapter we will be learning about pointer address operator in C programming.

Pointer address operator in C Programming

1. Pointer address operator is denoted by '&' symbol
2. 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);
```

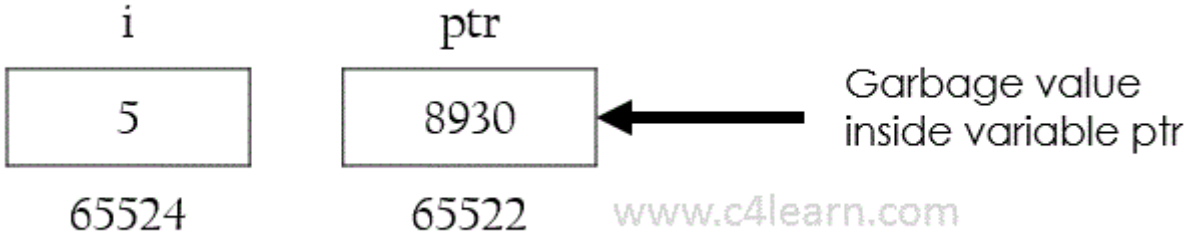
Understanding address operator

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);
}
```

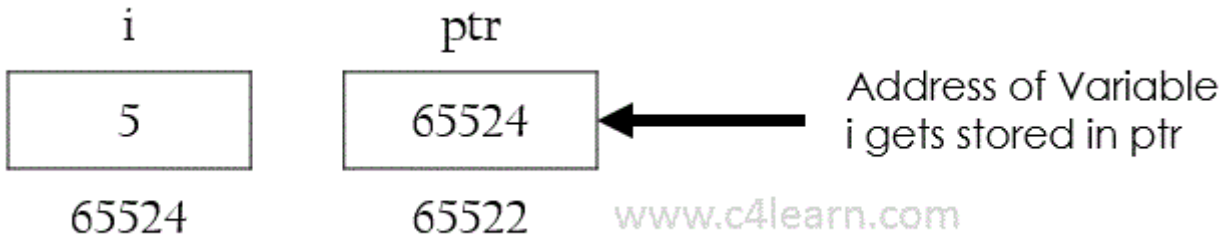
After declaration memory map will be like this –

```
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)
```

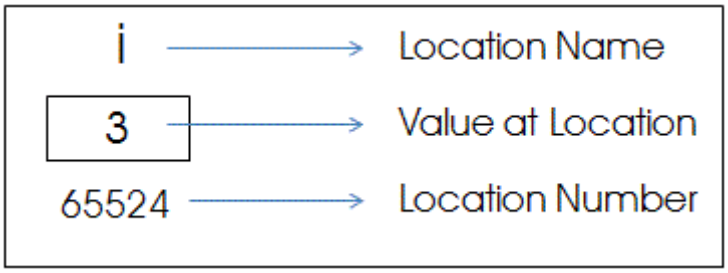
C pointer memory organization

In the last chapter we have learnt about Address operator in C Programming. In this chapter we are going to learn the memory organization for pointer variable.

Memory Organization for Pointer Variable :

1. When we use variable in program then Compiler keeps some memory for that variable depending on the **data type**
2. The address given to the variable is Unique with that variable name

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



Explanation :

1. Pointer Variable is nothing but a memory address which holds another address .
2. 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.

C pointer variable memory required

How much Memory required to store Pointer variable?

1. Pointer Variable Stores the address of the variable
2. Variable may be **integer,character,float** but the **address** of the variable is always **integer** so Pointer requires **2 bytes** of memory in Turbo C Compiler.
3. If we run same program in any other IDE then the output may differ.

4. This requirement is **different for different** Compilers

Program 1: Size of Integer Pointer

```
#include<stdio.h>

int main()
{
int a = 10, *ptr;
ptr = &a;

printf("\nSize of Integer Pointer : %d",sizeof(ptr));

return(0);
}
```

Output :

```
Size of Integer Pointer : 2
```

Program 2 : Size of Character Pointer

```
#include<stdio.h>

int main()
{
char a = 'a', *cptr;
cptr = &a;

printf("\nSize of Character Pointer : %d",sizeof(cptr));

return(0);
}
```

Output :

```
Size of Character Pointer : 2
```

Program 3 : Size of Float Pointer

```c
#include<stdio.h>

int main()
{
float a = 3.14, *fptr;
fptr = &a;

printf("\nSize of Character Pointer : %d",sizeof(fptr));

return(0);
}
```

Output :

```
Size of Float Pointer : 2
```

C size of pointer variable

In the previous chapter, we have learnt about the memory organization of the pointer variable. In this chapter we are learning to compute or find the size of pointer variable.

Size of Pointer variable in C

Pointer is the variable that stores the address of another variable.

Size calculation

Consider the following memory map before declaring the variable.In the memory comprise of blocks of 1 byte.

This is memory before declaring variable. Each Square represents one byte of memory

Whenever we declare any variable then random block of memory is chosen and value will be stored at that memory location.

Similarly whenever we declare any pointer variable then random block of memory will be used as pointer variable which can hold the address of another variable.

Program

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

```
int main()
{
int *iptr = NULL;
float *fptr = NULL;
char *cptr = NULL;

printf("\nSize of Integer Pointer   : %d Bytes",sizeof(iptr));
printf("\nSize of Character Pointer : %d Bytes",sizeof(cptr));
printf("\nSize of Float Pointer     : %d Bytes",sizeof(fptr));

return 0;
}
```

Output :

```
Size of Integer Pointer   : 4 Bytes
Size of Character Pointer : 4 Bytes
Size of Float Pointer     : 4 Bytes
```

Explanation

1. Pointer stores the address of the Variable.
2. Size of Pointer Variable can be evaluated by using sizeof operator
3. Address of variable is considered as integer value.
4. For Borland C/C++ Compiler, for storing integer value 2 bytes are required.
5. Thus Pointer variable requires 2 bytes of memory.

Size of pointer in different IDE

Compiler	Size of Integer	Size of Integer Pointer
Borland C/C++	2 Bytes	2 Bytes
Turbo C/C++	2 Bytes	2 Bytes
Visual C++	4 Bytes	4 bytes

C pointer declaration

In the previous chapter we have learnt about Pointer operators supported by C Language. In this chapter we will be learning about declaration of 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[/box]

```
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;
```

C initialization of pointer

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 :

[box]

1. Declare a Pointer Variable and Note down the Data Type.
2. Declare another Variable with Same Data Type as that of Pointer Variable.
3. Initialize Ordinary Variable and assign some value to it.
4. Now Initialize pointer by assigning the address of ordinary variable to pointer variable.

[/box]
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
```

C dereferencing pointer

Dereferencing Pointer in C Programming Language :

1. **Asterisk(*) indirection operator** is used along with pointer variable while Dereferencing the pointer variable.
2. Asterisk Operator is also called as **value at operator**
3. When used with Pointer variable, it refers to variable being pointed to,this is called as **Dereferencing of Pointers**

What is Dereferencing Pointer ?

[box]

1. Dereferencing Operation is performed to access or manipulate data contained in memory location pointed to by a pointer
2. Any Operation performed on the **de-referenced pointer** directly affects the value of variable it pointes to.

[/box]

Live Example : Dereferencing Pointer

```
// Sample Code for Dereferencing of Pointer
int n = 50 , x ;

int *ptr ; // Un-initialized Pointer

ptr = &n; // Stores address of n in ptr

x = *ptr; // Put Value at ptr in x
```

Evaluation :

```
// Sample Code for Dereferencing of Pointer

*(ptr) = value at (ptr)
       = value at (2001)
       = 50
```

```
//Address in ptr is nothing but address of n
```

Summary :

1. **Name of Normal Variable** always points to the **Value of variable**
2. Pointer variable always **Stores Address of variable**
3. ***ptr** and **num** will give us same value.

Variable	Value in it
num	50
&num	1002
ptr	1002
*ptr	50

In the next chapter we will be learning the void pointers. Void Pointer is special type of pointer which can store the address of any variable.

C void pointers

In the previous chapter we have learnt about dereferencing of pointer variable. In this chapter we will be looking special type of pointer called void pointer or general purpose pointer.

Void Pointers in C : Definition

1. Suppose we have to declare integer pointer,character pointer and float pointer then we need to declare 3 pointer variables.
2. Instead of declaring different types of pointer variable it is feasible to declare single pointer variable which can act as integer pointer,character pointer.

Void Pointer Basics :

1. In C **General Purpose Pointer** is called as void Pointer.
2. It does not have any data type associated with it
3. It can store address of any type of variable
4. A void pointer is a C convention for a raw address.
5. The compiler has no idea what type of object a void Pointer really points to ?

Declaration of Void Pointer :

```
void * pointer_name;
```

Void Pointer Example :

```
void *ptr;    // ptr is declared as Void pointer

char cnum;
int inum;
float fnum;

ptr = &cnum;  // ptr has address of character data
ptr = &inum;  // ptr has address of integer data
```

```
ptr = &fnum;  // ptr has address of float data
```

Explanation :

```
void *ptr;
```

1. **Void pointer** declaration is shown above.
2. We have declared 3 variables of integer,character and float type.
3. When we assign **address of integer** to the void pointer, pointer will become Integer Pointer.
4. When we assign **address of Character** Data type to void pointer it will become Character Pointer.
5. Similarly we can assign address of any data type to the void pointer.
6. It is capable of storing address of any data type

Summary : Void Pointer

Scenario	Behavior
When We assign address of integer variable to void pointer	Void Pointer Becomes Integer Pointer
When We assign address of character variable to void pointer	Void Pointer Becomes Character Pointer
When We assign address of floating variable to void pointer	Void Pointer Becomes Floating Pointer

C dereferencing void pointer

Why it is necessary to use Void Pointer ?

Reason 1 : Re-usability of Pointer

```
float *ptr;
int num;


ptr = &num ;  // Illegal Use of Pointer
```

In the above example we have declared float pointer which is of no use in the program. We cannot use float pointer to store the integer pointer so So in order to increase the re-usability of the Pointer it is necessary to declared pointer variable as void Pointer.

Dereferencing Void Pointer :

Consider this example –

```
void *ptr;    // ptr is declared as Void pointer


char cnum;
int inum;
float fnum;
```

Suppose we have assigned integer Address to void pointer then –

```
ptr = &inum;  // ptr has address of integer data
```

then to De-reference this void pointer we should use following syntax –

```
*((int*)ptr)
```

Similarly we should use following for Character and float –

```
*((float*)ptr)  // De-reference Float Value
*((char*)ptr)   // De-reference Character Value
```

Following table will summarize all details –

Type of Address Stored in Void Pointer	**Dereferencing Void Pointer**
Integer	*((int*)ptr)
Charcter	*((char*)ptr)
Floating	*((float*)ptr)

Live Example : Dereferencing Void Pointer

```
#include<stdio.h>

int main()
{
int inum = 8;
float fnum = 67.7;
void *ptr;

ptr = &inum;
printf("\nThe value of inum = %d",*((int*)ptr));

ptr = &fnum;
printf("\nThe value of fnum = %f",*((float*)ptr));
```

```
return(0);
}
```

Output :

```
The value of inum = 8
The value of fnum = 67.7
```

Explanation : Void Pointer Program

```
ptr = &inum;
printf("\nThe value of inum = %d",*((int*)ptr));
```

Here we are assigning the address of integer variable to void pointer so that void pointer can act as Integer pointer.

```
ptr = &fnum;
printf("\nThe value of fnum = %f",*((float*)ptr));
```

And now we are storing the address of floating point variable to void pointer , thus void pointer becomes floating pointer.

C size of void pointer

What is the size of Void Pointer variable in C ?

Pointer :

Pointer is the variable that stores the address of another variable .

How to calculate size of Pointer ?

1. Size of **Void Pointer** can be evaluated by using sizeof operator
2. Pointer stores the address of the Variable .
3. Address of variable is nothing but the integer value .

4. In C , for storing integer value 2 bytes are required .
5. So void Pointer variable requires 2 bytes of memory.

```
int main(void)
{
  int num = 0;

  void *ptr;

  num = 15;

  ptr = &num;

  /* Output the size   */

 printf("Size of Pointer : %d Bytes",sizeof(pointer));

  return 0;
}
```

Output :

```
Size of Pointer : 2 Bytes
```

C pointer arithmetic operations

Consider below table, We have declared some of the variables and also assumed some address for declared variables.

Data Type	Initial Address	Operation	Address after Operations	Required Bytes
int	4000	++	4002	2
int	4000	- -	3998	2
char	4000	++	4001	1
char	4000	- -	3999	1
float	4000	++	4004	4
float	4000	- -	3996	4
long	4000	++	4004	4
long	4000	- -	3996	4

We can see address of an variable after performing arithmetic operations.

Expression	Result
Address + Number	Address
Address – Number	Address
Address – Address	Number
Address + Address	Illegal

Above table clearly shows that we can add or subtract address and integer number to get valid address. We can even subtract two addresses but we cannot add two addresses

Incrementing Pointer :

1. Incrementing Pointer is generally used in array because we have contiguous memory in array and we know the contents of next memory location.
2. Incrementing Pointer Variable Depends Upon data type of the Pointer variable

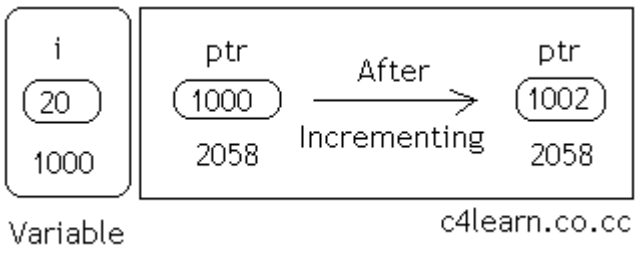
Formula : (After incrementing)

```
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 :

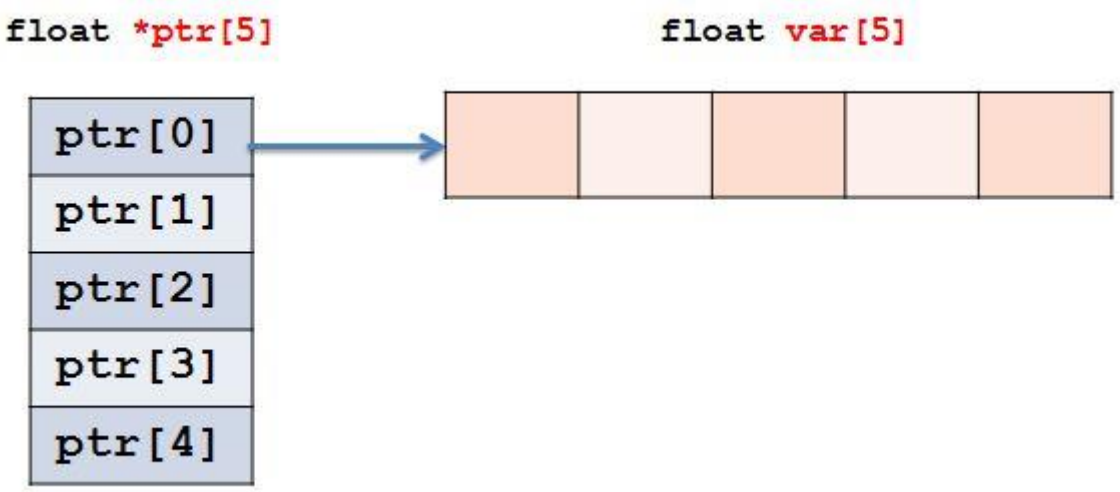
```
New Value of ptr : 1004
```

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

C pointer addition

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
```

C comparing two pointer variables

Comparison between two Pointers :

1. **Pointer comparison is Valid** only if the **two pointers are Pointing to same array**
2. All Relational Operators can be used for comparing pointers of **same type**
3. **All Equality and Inequality Operators** can be used with all Pointer types
4. 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 :

```c
#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]

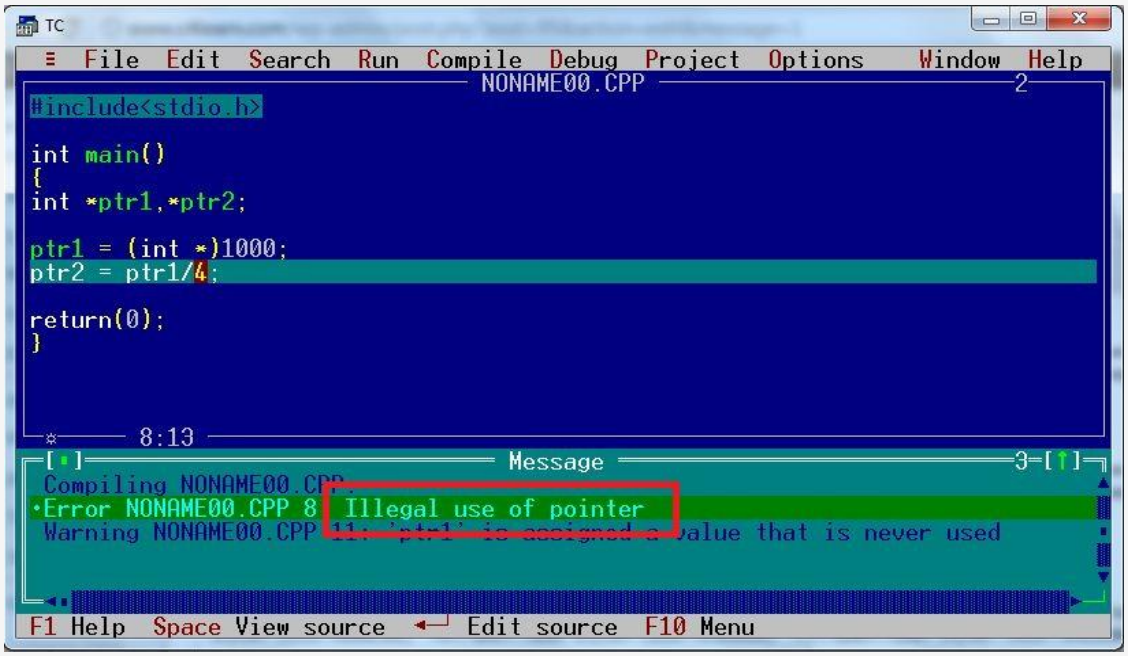
Following operations on pointers :

>	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 :



C pointer operation rules

Rules for using Pointer :

1. Pointer Variable Can be Assigned the address of another Variable

```
int main() {
    int num = 10;
    int *ptr;
    ptr = &num;
return(0);
}
```

2. Pointer Variable Can be Assigned the value of another Pointer Variable

```
int *sptr,*tptr;
```

```
sptr = &num;


tptr = sptr;
```

3. Pointer Variable Can be initialized with zero or NULL value.

```
int *sptr,*tptr;
sptr = 0;
tptr = NULL;
```

4. Pointer variable Can be Perform Pre/Post fix Increment/Decrement Operation

```
int arr[20];
int *ptr;
ptr = &arr;
ptr++;
```

5. Integer value can be added or Subtracted from Pointer variable

```
int arr[20];
int *ptr;
ptr = &arr;
ptr = ptr + 2; //arr[2] will be accessed
```

6. When two Pointer points to same array then one Pointer variable can be Subtracted from another
7. Two Pointers pointing to objects of same data type then they can be compared using the Relational Operator
8. Value cannot be assigned to arbitrary address

```
int *ptr;
ptr = 100; //Illegal
```

9. Pointer Variable cannot be multiplied by Constant

```
int *ptr;
ptr = ptr * 6; //Illegal
```

C pointer invalid operations

One can perform different arithmetic operations on Pointer such as increment,decrement but still we have some more arithmetic operations that cannot be performed on pointer –

1. Addition of two addresses.
2. Multiplying two addresses.
3. Division of two addresses.
4. Modulo operation on pointer.
5. Cannot perform bitwise AND,OR,XOR operations on pointer.
6. Cannot perform NOT operation or negation operation.

1. Addition of Two Pointers :

```
#include<stdio.h>

int main()
{

int var = 10;
int *ptr1 = &i;
int *ptr2 = (int *)2000;

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

return 0;
}
```

Output :

```
Compile Error
```

2. Multiplication of Pointer and number :

```
#include<stdio.h>

int main()
{

int var = 10;
```

```
int *ptr1 = &i;
int *ptr2 = (int *)2000;

printf("%d",ptr1*var);

return 0;
}
```

Output :

```
Compile Error
```

3. Multiplication of Two Pointers :

```
#include<stdio.h>

int main()
{

int var = 10;
int *ptr1 = &i;
int *ptr2 = (int *)2000;

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

return 0;
}
```

Output :

```
Compile Error
```

Similarly we cannot perform following operations –

---

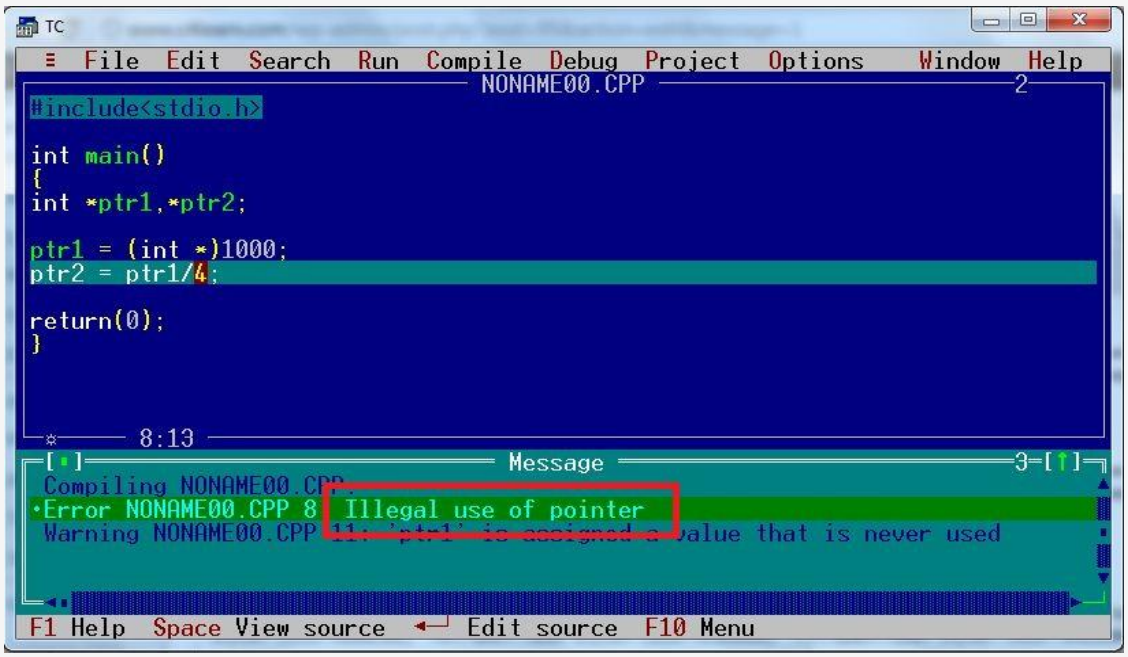
4. Modulo Operation

```
int *ptr1 = (int *)1000;
int *ptr2 = (int *)2000;
int var = ptr2 % ptr1;
```

5. Division of pointer

```
#include<stdio.h>

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

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

return(0);
}
```

Output :



6.Cannot perform bitwise OR

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

```
int main(){

int i=5,j=10;
int *p=&i;
int *q=&j;

printf("%d",p|q);

return 0;
}
```

7.Negation Operation on Pointer

```
#include<stdio.h>

int main(){

int i=5;
int *ptr=&i;

printf("%d",~ptr);

return 0;
}
```

Summary at a glance :

```
Address + Address = Illegal
Address * Address = Illegal
Address / Address = Illegal
Address % Address = Illegal
Address & Address = Illegal
Address | Address = Illegal
```

```
Address ^ Address = Illegal
~Address          = Illegal
```

C precedence of * and & operator

.

Precedence of Value at Operator :

In C Star(*) operator is used for two purposes –

- Multiplication
- In Pointer for De-referencing

Some important points about De-Reference Operator :

1. Value at Operator (*) is unary operator when use in pointer.
2. Multiplication Operator (*) is binary operator when used for multiplication.

3. Value at operator '*' in pointer concept is also called as "**De-reference**" Operator Or "**Value at Operator**"

Precedence of ' * ' and ' & ' Operator

1. Both are **Unary Operators**
2. They have **Equal Precedence**
3. They Associate from **Right –> Left**

C double pointer

Pointer to Pointer in C Programming

Declaration : Double Pointer

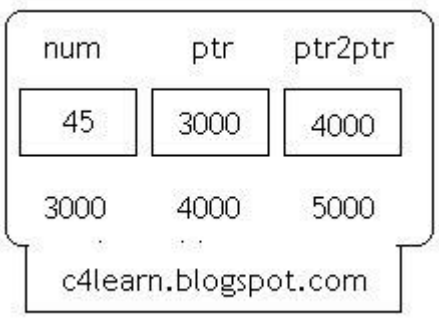
```
int  **ptr2ptr;
```

Consider the Following Example :

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

What is Pointer to Pointer ?

1. Double (**) is used to denote the **double Pointer**
2. Pointer Stores the address of the Variable
3. Double Pointer **Stores the address of the Pointer Variable**



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

Notes :

1. Conceptually we can have Triple ….. n pointers
2. Example : *****n,****b can be another example

Live Example :

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

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

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

return(0);
}
```

Output :

```
45
```

C pointer to array element

Pointer to array in C : Pointer pointing to the array of elements

---

1. Array and Pointer are backbones of the C Programming language
2. Pointer and array , both are closely related
3. We know that array name without subscript points to first element in an array

Example : One dimensional array

```
int a[10];
```

Here a , a[0] both are identical

Example : Two dimensional Array

```
int a[10][10];
```

Here a , a[0][0] both are identical

Let

1. x is an array ,
2. i is subscript variable

$$x[i] = *(x + i)$$
$$= *(i + x)$$
$$= i[x]$$

Proof of the above Formula :

1 . For first element of array **i** = 0

$$x[0] = *(x + 0)$$
$$= *x$$

Thus we have concluded that , first element of array is equivalent to *x .

2 . Thus *(x+i) will gives us value of ith element.

Summary : Consider following code

```
main()
{
int x[] = {1,2,3,4,5};
int *ptr,i ;
ptr = x
for(i=0;i<5;i++)
   {
   printf("nAddress : %u",&x[i]);
   printf("nElement : %d",x[i]);
   printf("nElement : %u",*(x+i));
   printf("nElement : %d",i[x]);
   printf("nElement : %d",*ptr);
   }
}
```

Output :

```
   Address : 7600
   Element : 1
   Element : 1
   Element : 1
   Element : 1
   Address : 7602
   Element : 2
   Element : 2
   Element : 2
   Element : 2
   Address : 7604
   Element : 3
   Element : 3
   Element : 3
   Element : 3
   Address : 7606
   Element : 4
   Element : 4
   Element : 4
   Element : 4
   Address : 7608
   Element : 5
```

```
Element : 5
Element : 5
Element : 5
```

Following 4 are considered as Equivalent	
x[i]	*(x + i)
i[x]	*ptr

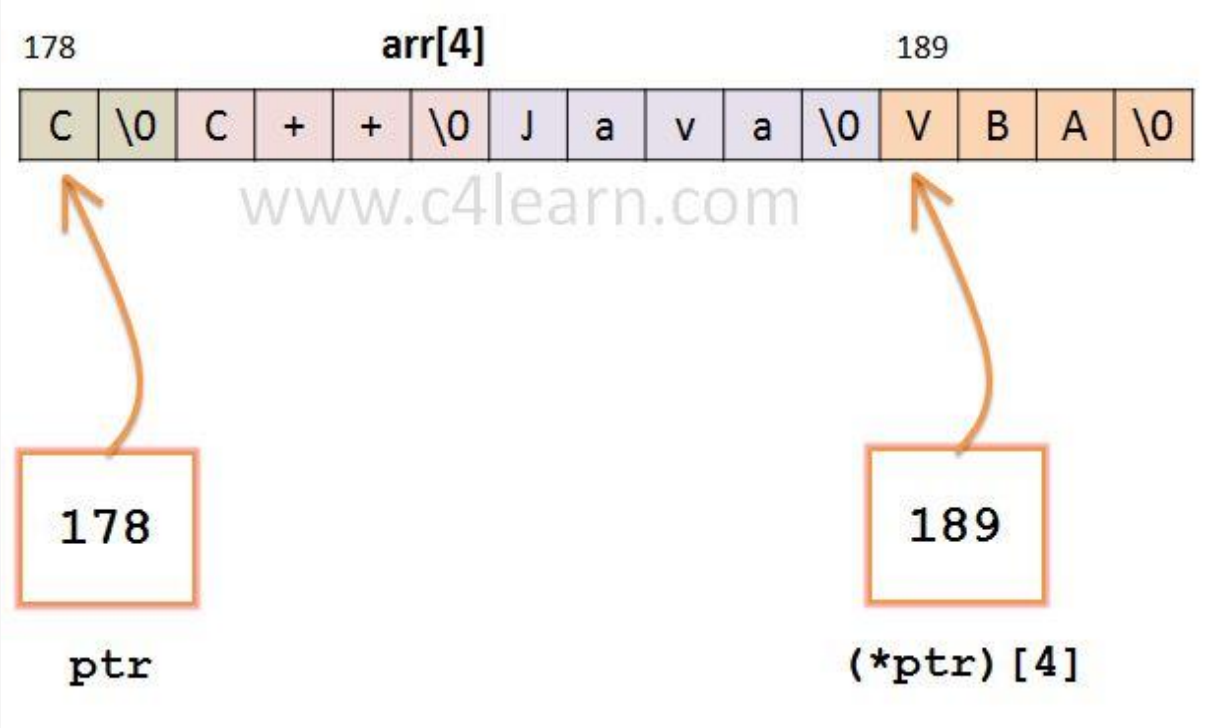
2-D Array can be accessed –

x [i][j] = *(x[i] + j)

= *(* (x + i) + j)

C pointer to array of string

Pointer to array of string : A pointer which pointing to an array which content is string, is known as pointer to array of strings.



In this example

1. ptr : It is pointer to array of string of size 4.
2. array[4] : It is an array and its content are string.

1 : Printing Address of the Character Array

```
#include<stdio.h>

int main()
{
int i;

char *arr[4] = {"C","C++","Java","VBA"};
char *(*ptr)[4] = &arr;

for(i=0;i<4;i++)
   printf("Address of String %d : %u\n",i+1,(*ptr)[i]);

return 0;
}
```

Output :

```
Address of String 1 = 178
Address of String 2 = 180
Address of String 3 = 184
Address of String 4 = 189
```

---

2. Printing Contents of character array

```
#include<stdio.h>

int main()
```

```
{
int i;

char *arr[4] = {"C","C++","Java","VBA"};
char *(*ptr)[4] = &arr;

for(i=0;i<4;i++)
   printf("String %d : %s\n",i+1,(*ptr)[i]);

return 0;
}
```

Output :

```
String 1 = C
String 2 = C++
String 3 = Java
String 4 = VBA
```

3. Pointer to String and Pre-increment Operator

```
#include<stdio.h>

int main()
{
int i;

char *arr[4] = {"C","C++","Java","VBA"};
char *(*ptr)[4] = &arr;

printf("%s",++(*ptr)[2]);

return 0;
```

```
}
```

Output:

```
ava
```

C function pointer

What is function Pointer ?

Function pointer : A pointer which keeps address of a function is known as function pointer.
[Visit : Complete Reference Document for Function Pointer]

Live Example :

```
#include<stdio.h>

void display();

int main()
 {
 void *(*ptr)();
 ptr = &display;
 (*ptr)();

 return(0);
 }

void display()
{
printf("Hello World");
}
```

Output :

```
Hello World
```

Explanation of C Snippet :

Consider normal program –

Now in the above program we have just called function display(), and we get output as “Hello World”.

Consider Scenario using pointer , We should follow following 3 steps to use pointer to call function –

1. Declare Pointer which is capable of storing address of function.
2. Initialize Pointer Variable
3. Call function using Pointer Variable.

Step 1 : Declaring Pointer

```
void *(*ptr)();
```

- We are simply declaring a double pointer.
- Write () symbol after "**Double Pointer**".
- void represents that , function is not returning any value.
- () represents that , function is not taking any parameter.

Above declaration tells us ….

Declare Pointer variable that can store address of function which does not return anything and doesn't take any parameter

Step 2 : Initializing Pointer

```
ptr = &display;
```

This statement will store address of function in pointer variable.

Step 3 : Calling a function

```
(*ptr)();
```

using (*ptr)() we can call function display();

```
(*ptr)() = (*ptr)();
         = (*&display)();
         = (display)();
         = display();
```

thus (*ptr)() is as good as calling function.

Requirement	Declaration of Function Pointer	Initialization of Function Pointer	Calling Function using Pointer

Return Type : None Parameter : None	**void** *(*ptr)();	ptr = &display;	(*ptr)();
Return Type : Integer Parameter : None	**int** *(*ptr)();	ptr = &display;	**int** result; result = (*ptr)();
Return Type : Float Parameter : None	**float** *(*ptr)();	ptr = &display;	**float** result; result = (*ptr)();
Return Type : Char Parameter : None	**char** *(*ptr)();	ptr = &display;	**char** result; result = (*ptr)();

Example 1 : Function having two Pointer Parameters and return type as Pointer

```
#include<stdio.h>

char * getName(int *,float *);

int main()
{
char *name;
int  num = 100;
float marks = 99.12;

char *(*ptr)(int*,float *);
ptr=&getName;
name = (*ptr)(&num,&marks);
```

```
printf("Name : %s",name);

return 0;
}
//---------------------------------
char *getName(int *ivar,float *fvar)
{
char *str="www.google.com";
str = str + (*ivar) + (int)(*fvar);
return(str);
}
```