

# Pointer In C Programming:

## Address Of Operator Or Referencing Operator:

It is a referencing unary operator. Which is used for to get the address of a variable. The symbol of Address Of Operator is (&). The syntax to use this operator is "&variable\_name".

## Indirection Of Operator Or Dereferencing Operator:

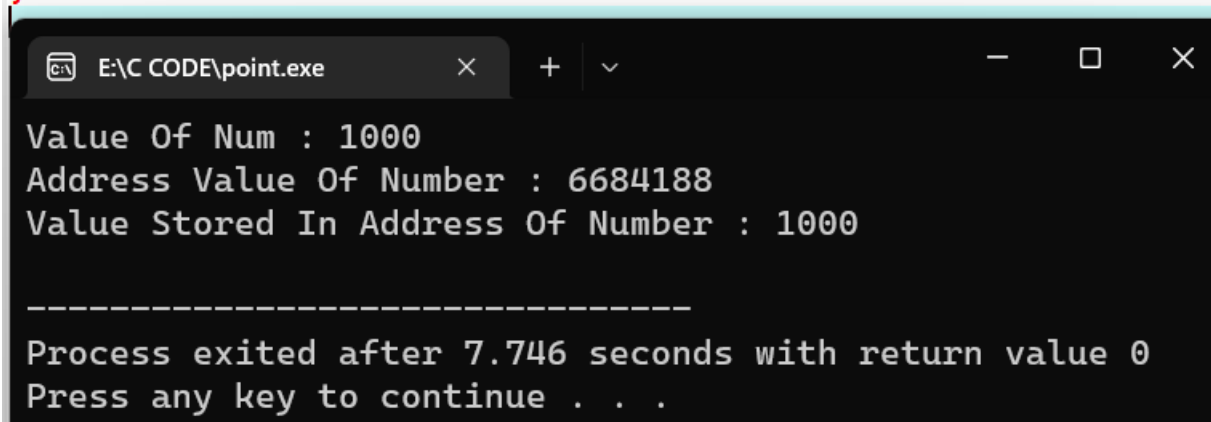
It is a dereferencing unary operator. Which is used to get the value which is stored in that particular address. The symbol of Indirection Operator is (\*). The syntax to use this operator is "\*address\_name".

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

int main()
{
    int num = 1000;

    printf("Value Of Num : %d\n",num);
    printf("Address Value Of Number : %d\n",&num);
    printf("Value Stored In Address Of Number : %d\n",&num);

    getch();
}
```



```
E:\C CODE\point.exe
Value Of Num : 1000
Address Value Of Number : 6684188
Value Stored In Address Of Number : 1000

-----
Process exited after 7.746 seconds with return value 0
Press any key to continue . . .
```

"&num" is just a way to represent address of variable num. "&num" is treated as a constant. "&num" is not a variable.

A pointer is a variable which contains address of another variable, object or a function. A pointer is declared much like any other variable except an asterisk(\*) is placed between the type and the name of the variable to denote it is a pointer. Ordinary variable size is depends on the datatype but in case of pointer variable the size is not depends on the datatype.

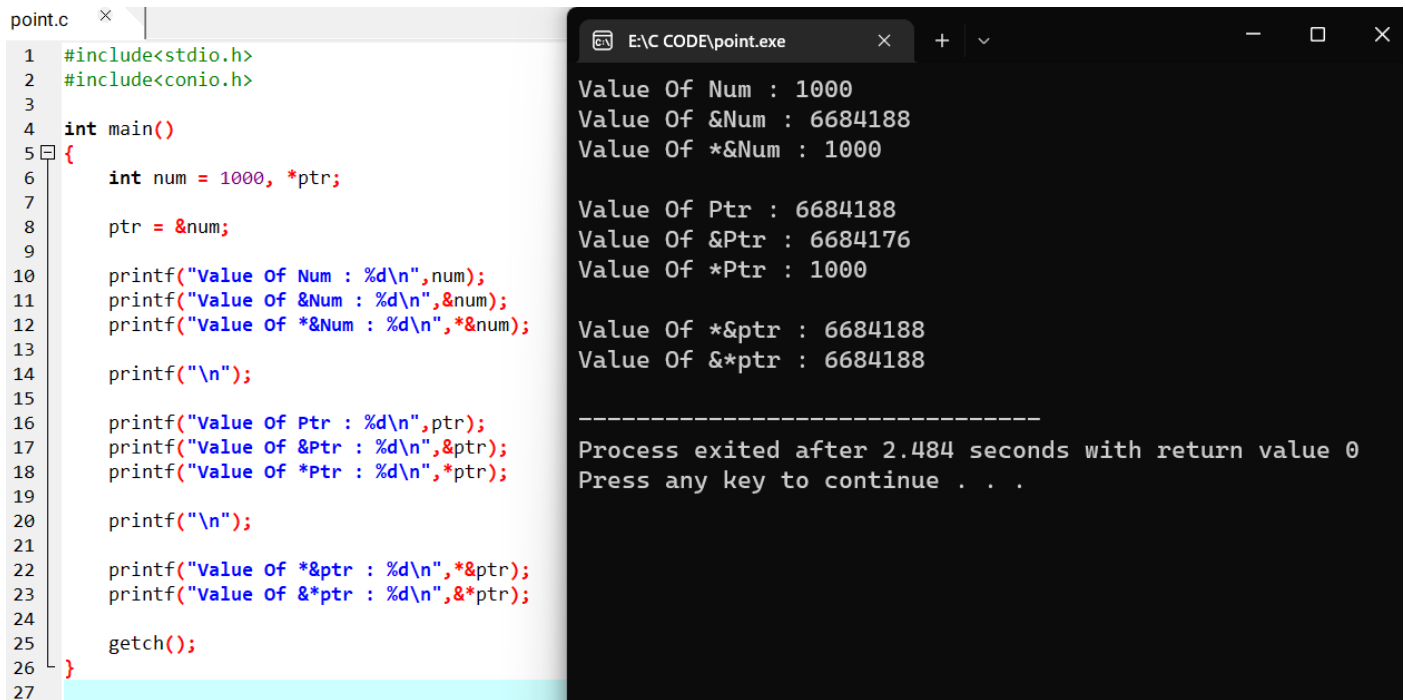
In C programming language there are 4 segments of memory. "CODE SEGMENT", "DATA SEGMENT", "HEAP SEGMENT", "STACK SEGMENT".

In CODE SEGMENT the program equivalent machine instruction are stored. In DATA SEGMENT where all the global variables of your program re siders. In HEAP SEGMENT programmer can have memory allocated from heap at run time of the program, the allocation will not be

sequential. In STACK SEGMENT function local variables are stored in here. This memory also used to hold function arguments and return values.

pointer are variables talks such as dynamic memory allocation cannot be performed without using pointers. every variable is a memory location and every memory location has its addressed using (&) operator which denotes an address in memory.

pointer is a variable whose value is the address of an another variable that direct address of memory location. like any variable of constant, you must declare a pointer before you can use it to store any variable address.



The image shows a C program named 'point.c' and its execution output. The program defines a variable 'num' with the value 1000 and a pointer 'ptr' that points to 'num'. It then prints the values of 'num', '&num', '\*&num', 'ptr', '&ptr', '\*ptr', '\*&ptr', and '&\*ptr'.

```
point.c  x
1  #include<stdio.h>
2  #include<conio.h>
3
4  int main()
5  {
6      int num = 1000, *ptr;
7
8      ptr = &num;
9
10     printf("Value Of Num : %d\n",num);
11     printf("Value Of &Num : %d\n",&num);
12     printf("Value Of *&Num : %d\n",&num);
13
14     printf("\n");
15
16     printf("Value Of Ptr : %d\n",ptr);
17     printf("Value Of &Ptr : %d\n",&ptr);
18     printf("Value Of *Ptr : %d\n",*ptr);
19
20     printf("\n");
21
22     printf("Value Of *&ptr : %d\n",&*ptr);
23     printf("Value Of *&ptr : %d\n",&*ptr);
24
25     getch();
26 }
27
```

The execution output shows the following values:

```
E:\C CODE\point.exe
Value Of Num : 1000
Value Of &Num : 6684188
Value Of *&Num : 1000

Value Of Ptr : 6684188
Value Of &Ptr : 6684176
Value Of *Ptr : 1000

Value Of *&ptr : 6684188
Value Of *&ptr : 6684188

-----
Process exited after 2.484 seconds with return value 0
Press any key to continue . . .
```

pointer holds an address which is a numeric value. therefore. you can perform arithmetic operations on a pointer just as you can a numeric value. there are four arithmetic operators that can be used on pointers "++, --, +, -". the arithmetic operators on pointer variable is dependent upon the data type for which the pointer variable is declared. A pointer to pointer is a form of multiple indirection, or a chain of pointers.

If a pointer is uninitialized then that pointer will act as a wild pointer. Null pointer is a type of pointer concept where it always a good practice to assign a null value to a pointer variable in case you do not have exact address to be assigned. a pointer that is assigned null is called a null pointer.

near pointer(16 bit) up to 65,535 address space. huge pointer(32 bit) up to 1 mb. Far pointer(32 bit) up to 1mb, faster access than huge pointer. void pointer is a type of pointer which has no associated data type with it. it can point to any data of any data type can be typecasted to any type. we cannot dereferenced a void pointer. malloc and calloc function returns a void pointer. due to this reason they can allocate a memory for any type of data.

```

point.c x
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main()
5 {
6     int num1 = 1000, num2 = 2000, *ptr1, *ptr2;
7
8     ptr1 = &num1;
9     ptr2 = &num2;
10
11     printf("Address Of &Num1 : %d\n",&num1);
12     printf("Address Of &Num2 : %d\n",&num2);
13
14     printf("\n");
15
16     printf("Value Of Ptr1 : %d\n",ptr1);
17     printf("Ptr1 + 1 : %d\n\n",(ptr1 + 1));
18
19     printf("Value Of Ptr2 : %d\n",ptr2);
20     printf("Ptr2 - 1 : %d\n",(ptr2 - 1));
21
22     getch();
23 }
24

```

```

E:\C CODE\point.exe
Address Of &Num1 : 6684172
Address Of &Num2 : 6684168

Value Of Ptr1 : 6684172
Ptr1 + 1 : 6684176

Value Of Ptr2 : 6684168
Ptr2 - 1 : 6684164

-----
Process exited after 12.79 seconds with return value 0
Press any key to continue . . .

```

```

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

int main()
{
    int num = 1000, *ptr1, **ptr2, ***ptr3;

    ptr1 = &num;
    ptr2 = &ptr1;
    ptr3 = &ptr2;

    printf("Value Of Num : %d\n",num);
    printf("Value Of &Num : %d\n",&num);
    printf("Value Of *&Num : %d\n",&num);

    printf("\n");

    printf("Value Of *Ptr1 : %d\n",*ptr1);
    printf("Value Of &Ptr1 : %d\n",&ptr1);
    printf("Value Of Ptr1 : %d\n",ptr1);
    printf("Value Of *&Ptr1 : %d\n",&ptr1);
    printf("Value Of *&Ptr1 : %d\n",&ptr1);

    printf("\n");

    printf("Value Of **Ptr2 : %d\n",**ptr2);
    printf("Value Of &Ptr2 : %d\n",&ptr2);
    printf("Value Of Ptr2 : %d\n",ptr2);
    printf("Value Of *&Ptr2 : %d\n",&ptr2);
    printf("Value Of *&Ptr2 : %d\n",&ptr2);

    printf("\n");

    printf("Value Of ***Ptr3 : %d\n",***ptr3);
    printf("Value Of &Ptr3 : %d\n",&ptr3);
    printf("Value Of Ptr3 : %d\n",ptr3);
    printf("Value Of *&Ptr3 : %d\n",&ptr3);
    printf("Value Of *&Ptr3 : %d\n",&ptr3);

    getch();
}

```

```

E:\C CODE\point.exe
Value Of Num : 1000
Value Of &Num : 6684188
Value Of *&Num : 1000

Value Of *Ptr1 : 1000
Value Of &Ptr1 : 6684176
Value Of Ptr1 : 6684188
Value Of *&Ptr1 : 6684188
Value Of *&Ptr1 : 6684188

Value Of **Ptr2 : 1000
Value Of &Ptr2 : 6684168
Value Of Ptr2 : 6684176
Value Of *&Ptr2 : 6684176
Value Of *&Ptr2 : 6684188

Value Of ***Ptr3 : 1000
Value Of &Ptr3 : 6684160
Value Of Ptr3 : 6684168
Value Of *&Ptr3 : 6684168
Value Of *&Ptr3 : 6684188

-----

```

## Wild Pointer:

Uninitialized pointers are known as wild pointers because they point to some arbitrary memory location and may cause a program to crash or behave badly. If a pointer p points to a known variable then it's not a wild pointer. If a pointer is uninitialized then that pointer will act as a wild pointer.

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

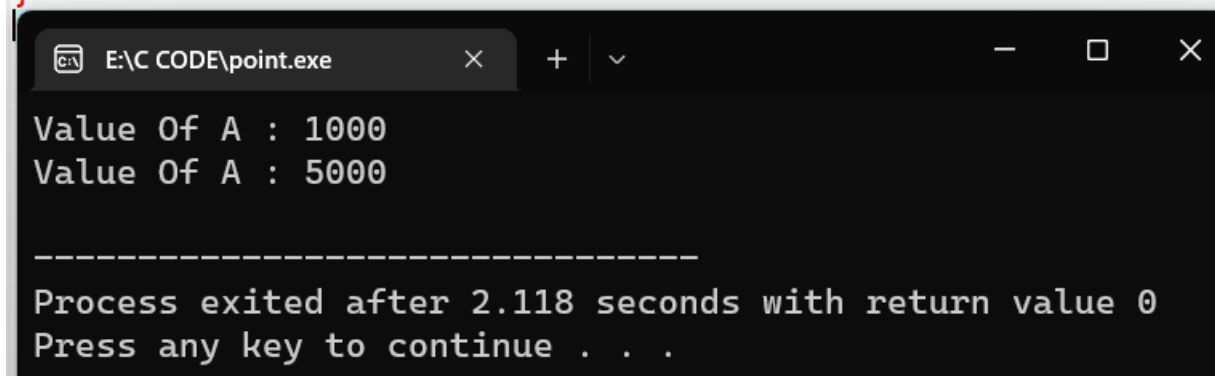
int main() //wild pointer
{
    int *ptr, a = 1000;

    printf("Value of A : %d\n",a);

    ptr = &a;
    *ptr = 5000;

    printf("Value of A : %d\n",a);

    getch();
}
```



```
E:\C CODE\point.exe
Value Of A : 1000
Value Of A : 5000

-----
Process exited after 2.118 seconds with return value 0
Press any key to continue . . .
```

## Null Pointer:

Null pointer is a type of pointer concept where it always a good practice to assign a null value to a pointer variable in case you do not have exact address to be assigned. a pointer that is assigned null is called a null pointer.

A Null Pointer is a pointer that does not point to any memory location. It stores the base address of the segment. The null pointer basically stores the Null value while void is the type of the pointer.

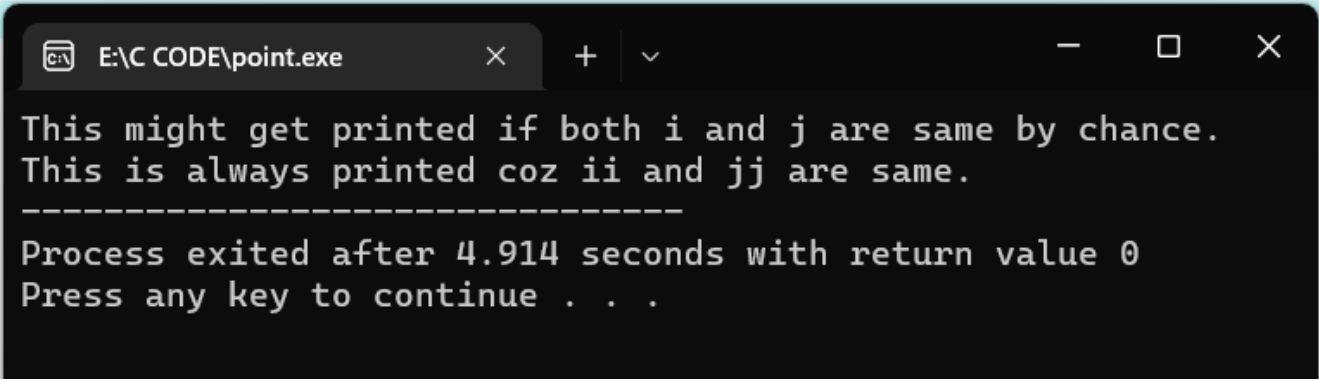
If we do not have any address which is to be assigned to the pointer, then it is known as a null pointer. When a NULL value is assigned to the pointer, then it is considered as a Null pointer.

```

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

int main() //null pointer
{
    int *i = NULL, *j = NULL;
    int *ii = NULL, *jj = NULL;
    if(i == j)
    {
        printf("This might get printed if both i and j are same by chance.\n");
    }
    if(ii == jj)
    {
        printf("This is always printed coz ii and jj are same.");
    }
    getch();
}

```



```

E:\C CODE\point.exe
This might get printed if both i and j are same by chance.
This is always printed coz ii and jj are same.
-----
Process exited after 4.914 seconds with return value 0
Press any key to continue . . .

```

## Near, Huge & Far Pointer:

near pointer(16 bit) up to 65,535 address space. In general, we can consider the near point as it is used to store the address, which has a maximum size of 16 bits only. Using the near pointer, we cannot store the address with a size greater than 16 bits. However, we can store all other smaller addresses that are within the 16 bits limit.

It is a pointer that works within the range of the kb data segment of memory. It cannot access addresses beyond the given data segment. A near pointer can be incremented or decremented in the address range by using an arithmetic operator.

Huge pointer(32 bit) up to 1 mb. Like the far pointer, the huge pointer also has the same size as 32 bits, but it can even access the bits stored or located outside the segment. However, the far pointer is quite similar to the Huge pointer, but it still has some advantages over the far pointer.

A pointer that can point to any segment in the memory is known as a huge pointer. A huge pointer has a size of 4 bytes or 32-bits, and it can access up to 64K size in memory. The huge pointer can be incremented without suffering with segment work round.

Far pointer(32 bit) up to 1mb, faster access than huge pointer. In general, Far pointer is typically considered as a pointer of 32 bits size. However, it can also access the information stored outside the computer's memory from the current segment. Although to use this type of

pointer, we usually need to allocate the sector register to store the data address in the current segment.

When the pointer is incremented or decremented, only the offset part is changed. It is such a type of pointer that stores both offset and segment address to which the pointer is differencing. A far pointer address ranges from 0 to 1MB. It can access all 16 segments.

## Void Pointer:

void pointer is a type of pointer which has no associated data type with it. it can point to any data of any data type can be typecasted to any type. we cannot dereferenced a void pointer. malloc and calloc function returns a void pointer. due to this reason they can allocate a memory for any type of data.

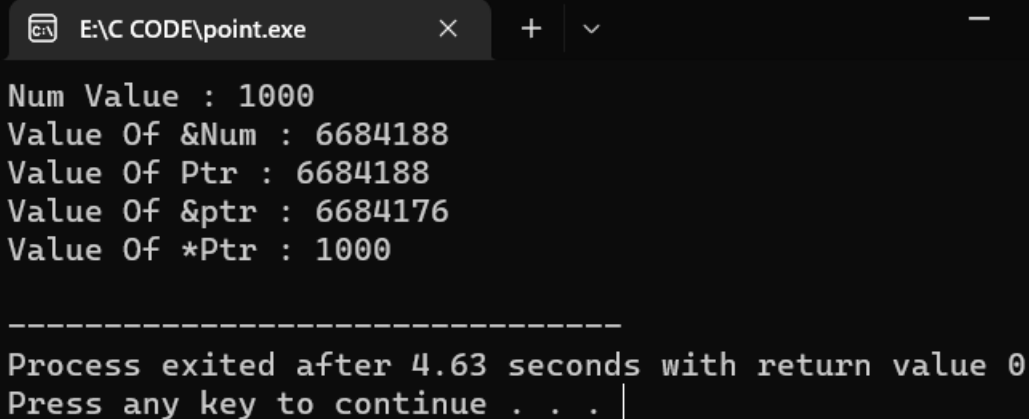
```
#include<stdio.h>
#include<conio.h>

int main()
{
    int num = 1000;

    void *ptr = &num;

    printf("Num Value : %d\n",num);
    printf("Value Of &Num : %d\n",&num);
    printf("Value Of Ptr : %d\n",ptr);
    printf("Value Of &ptr : %d\n",&ptr);
    printf("Value Of *Ptr : %d\n",*(int*)ptr);

    getch();
}
```



```
E:\C CODE\point.exe  x  +  v  -  □  X

Num Value : 1000
Value Of &Num : 6684188
Value Of Ptr : 6684188
Value Of &ptr : 6684176
Value Of *Ptr : 1000

-----
Process exited after 4.63 seconds with return value 0
Press any key to continue . . . |
```

## Dangling Pointer:

Dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer. In this case, the pointer is pointing to the memory, which is de-allocated. The dangling pointer can point to the memory, which contains either the program code or the code of the operating system. If we assign the value to this pointer, then it overwrites the value of the program code or operating system instructions; in such cases, the program will show the undesirable result or may even crash.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int *ptr = (int*)malloc(sizeof(int));
    int num = 5000;

    ptr = &num;
    printf("*ptr value : %d\n",*ptr);
    printf("ptr value : %d\n",ptr);

    free(ptr);

    printf("dangling *ptr value : %d\n",*ptr); //not showing
    getch();
}
```

E:\C CODE\point.exe

```
*ptr value : 5000
ptr value : 6684180
```

```
-----
Process exited after 0.5361 seconds with return value 3
221226356
Press any key to continue . . . |
```

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int *fuction()
{
    int num = 1000;
    return &num;
}
int main()
{
    int *ptr = fuction();

    printf("value of ptr : %d\n",ptr);
    printf("Value Of *ptr : %d\n",*ptr);

    getch();
}

```

```

E:\C CODE\point.exe
value of ptr : 0

-----
Process exited after 0.7383 seconds with return value 322122
5477
Press any key to continue . . . |

```

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int *fuction()
{
    static int num = 1000;
    return &num;
}
int main()
{
    int *ptr = fuction();

    printf("value of ptr : %d\n",ptr);
    printf("Value Of *ptr : %d\n",*ptr);

    getch();
}

```

```

E:\C CODE\point.exe
value of ptr : 4206608
Value Of *ptr : 1000

-----
Process exited after 1.974 seconds with return value 0
Press any key to continue . . . |

```



## **Static Memory Allocation & Dynamic Memory Allocation:**

`void* malloc(size_t n); void * malloc(int num);` it returns the pointer to 'n' bytes of uninitialized memory of null. This function allocates an array of num bytes and leave them initialized.

`void* calloc(size_t n, size_t size); void * calloc(int num, int size);` it return pointer to memory for an array of 'n' objects of the specialized 'size'. The memory is initialized zero. This function allocates an array of num elements each of whose size in bytes will be size.

`void *realloc(void *ptr, size_t size); void *realloc(void *address, int newsize);` it resize the memory pointed to by 'ptr' to 'size' bytes. returns the address of the newly allocated memory. The function re-allocates memory extending up to newsize.

`void free(void * address);` this function release a block of memory block specified by address.

## **Memory Map Of A Typical Program In C Programming:**

In every c program consists of 4 segments of memory. "code segment, data segment, heap segment, stack segment"

In code segment the program equivalent machine instruction are stored. In data segment where all the global variables of your program resides. In heap segment programmer can have memory allocated from heap at run time of the program. the allocation will not be sequential. In stack segment function local variables are stored in here. this memory also used to hold function arguments and return values.

## **How Program Uses Memory In C Programming:**

Heap Memory is a part of the main memory. It is unorganized and treated as a resource when you require the use of it if not release. Heap memory can't be used directly with the help of a pointer.

Stack Memory it stores temporary variables created by a function. In stack, variables are declared, stored and initialized during runtime. It follows the first in last out method that means whatever element is going to store last is going to delete first when it's not in use.

Code Section whenever the program is executed it will be brought into the main memory. This program will get stored the code section. Based upon the program it will decide whether to utilize the stack or heap sections.

## **Static Memory Allocation:**

In this memory allocation whenever the program executes it fixes the size that the program is going to take, and it can't be changed further. So the exact memory requirements must be known before allocation and deallocation of memory will be done by the compiler automatically. When everything is done at compile time before the run time it is called static memory allocation. Memory allocated during compile time is called static memory. The memory allocated is fixed and cannot be increased or decreased during run time.

1. Allocation and deallocation are done by the compiler.
2. It uses a data structures stack for static memory allocation.
3. Variables get allocated permanently. No reusability.
4. Execution is faster than dynamic memory allocation.
5. Memory is allocated before runtime. It is less efficient.

If you are allocating memory for an array during compile time then you have to fix the size at the time of declaration. Size is fixed and user cannot increase or decrease the size of the array at run time. If the values stored by the user in the array at run time is less than the size specified then there will be wastage of memory. If the values stored by the user in the array at run time is more than the size specified then the program may crash or misbehave.

## **Dynamic Memory Allocation In C Programming:**

The process of allocating memory at the time of execution is called dynamic memory allocation. The mechanism by which storage/memory/cells can be allocated to variables during the run time is called Dynamic Memory Allocation.

When we do not know how much amount of memory would be needed for the program beforehand. When we want data structures without any upper limit of memory space.

When you want to use your memory space more efficiently. Dynamically created lists insertions and deletions can be done very easily just by the manipulation of addresses whereas in case of statically allocated memory insertions and deletions lead to more movements and wastage of memory. When you want you to use the concept of structures and linked list in programming, dynamic memory allocation is a must.

Heap is the segment memory where dynamic memory allocation takes place. Unlike stack where memory is allocated or deallocated in a define order. Heap is an area of memory where memory is allocated or deallocated without any order or randomly.

There are certain built-in functions that can help in allocating or deallocating some memory space at runtime. Pointer plays an important role in dynamic memory allocation. Allocation memory can only be accessed through pointers. Built In Function That Are Used In Dynamic Memory Allocation: malloc(), calloc(), realloc(), free()

## **Malloc() Function In Dynamic Memory Allocation:**

`void* malloc(size_t n); void * malloc(int num);` it returns the pointer to 'n' bytes of uninitialized memory of null. This function allocates an array of num bytes and leave them initialized.

malloc is a built in function declared in the header file `<stdlib.h>`. malloc is the name for 'memory allocation' is used to dynamically allocate a single large block of contiguous memory according to the size specified.

`(void* )malloc(size_t size)` malloc function simply allocates a memory block according to the size specified in the heap and on success it returns a pointer pointing to the first byte of the allocated memory else return null. "size\_t" is defined in `<stdlib.h>` as unsigned int.

Malloc doesn't have an idea of what it is pointing to. It merely allocates memory requested by the user without knowing the type of data to be stored inside the memory. The void pointer can be typecasted to an appropriate type.

`int *ptr = (int* )malloc(4);` malloc allocates 4 bytes of memory in the heap and the address of the first byte is stored in the pointer 'ptr'.

## **Calloc() Function In Dynamic Memory Allocation:**

calloc() function is used to dynamically allocate multiple blocks of memory. calloc() needs two arguments instead of instead of just one.

`void* calloc(size_t n, size_t size); void * calloc(int num, int size);` it return pointer to memory for an array of 'n' objects of the specialized 'size'. The memory is initialized zero. This function allocates an array of num elements each of whose size in bytes will be size.

`void *calloc(size_t n, size_t size);` size\_t n = number of block, size\_t size = size of each block.

Memory allocated by calloc is initialized to zero. It is not the case with malloc. Memory allocated by malloc is initialized with some garbage value.

malloc and calloc both return null when sufficient memory is not available in the heap. calloc stands for clear allocation, malloc stands for memory allocation.

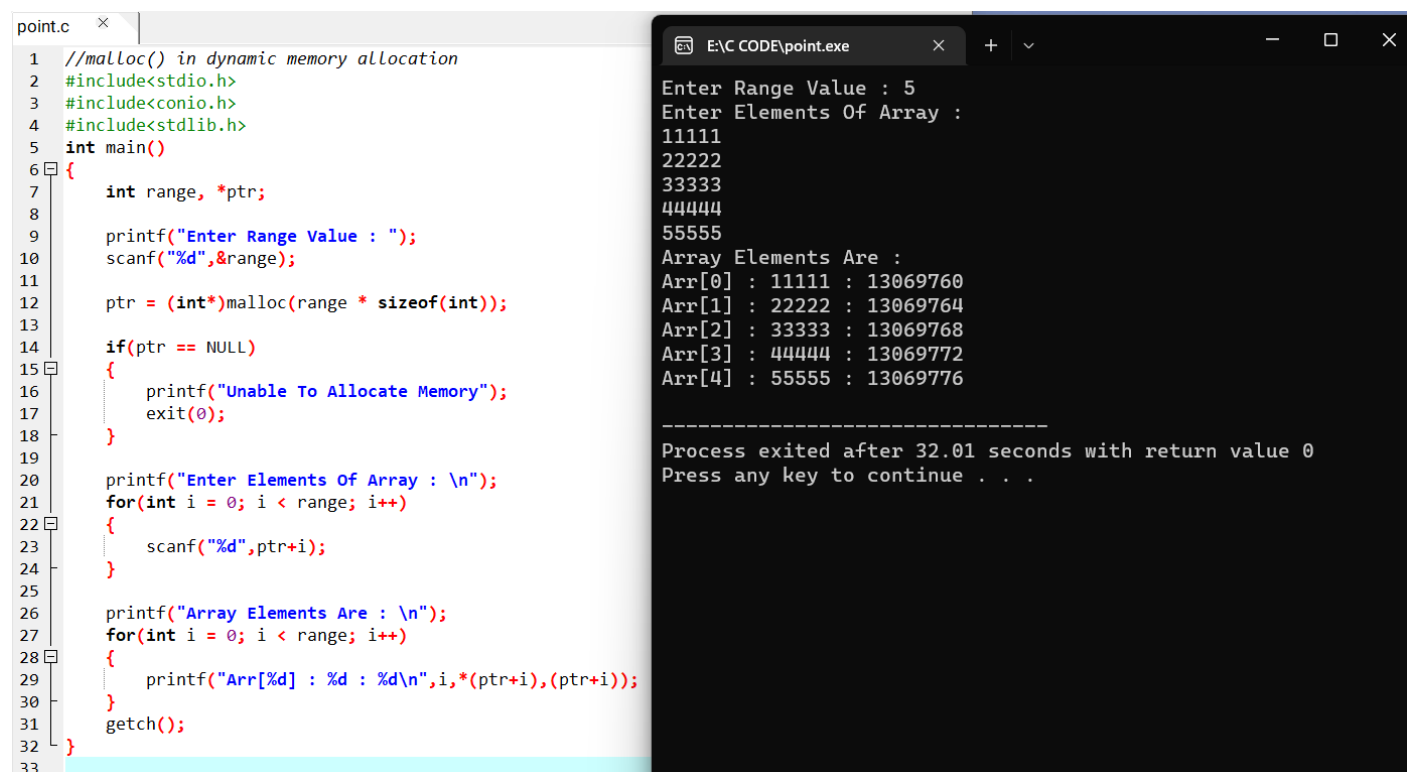
## **realloc() function in dynamic memory allocation:**

`void *realloc(void *ptr, size_t size); void *realloc(void *address, int newsize);` it resize the memory pointed to by 'ptr' to 'size' bytes. returns the address of the newly allocated memory. The function re-allocates memory extending up to newsize. realloc() function is used to change the size of the memory block without losing the old data. realloc means the re allocation. `void *realloc(void *ptr, size_t newsize)` void \*ptr = pointer to previously allocated memory. on failure realloc returns null.

## Free() function in dynamic memory allocation:

void free(void \* address); this function release a block of memory block specified by address. free() function is used to release the dynamically allocated memory in heap. void free(ptr);

the memory allocated in heap will not be released automatically after using the memory. the space remains there and can't be used. It is the programmer responsibility to release the memory after use.



The image shows a C program named 'point.c' and its execution output. The program uses malloc to allocate memory and free to release it. The output shows the program running successfully, allocating memory, and printing the array elements.

```
point.c
1 //malloc() in dynamic memory allocation
2 #include<stdio.h>
3 #include<conio.h>
4 #include<stdlib.h>
5 int main()
6 {
7     int range, *ptr;
8
9     printf("Enter Range Value : ");
10    scanf("%d",&range);
11
12    ptr = (int*)malloc(range * sizeof(int));
13
14    if(ptr == NULL)
15    {
16        printf("Unable To Allocate Memory");
17        exit(0);
18    }
19
20    printf("Enter Elements Of Array : \n");
21    for(int i = 0; i < range; i++)
22    {
23        scanf("%d",ptr+i);
24    }
25
26    printf("Array Elements Are : \n");
27    for(int i = 0; i < range; i++)
28    {
29        printf("Arr[%d] : %d : %d\n",i,*(ptr+i),(ptr+i));
30    }
31    getch();
32 }
33
```

```
E:\C CODE\point.exe
Enter Range Value : 5
Enter Elements Of Array :
11111
22222
33333
44444
55555
Array Elements Are :
Arr[0] : 11111 : 13069760
Arr[1] : 22222 : 13069764
Arr[2] : 33333 : 13069768
Arr[3] : 44444 : 13069772
Arr[4] : 55555 : 13069776

-----
Process exited after 32.01 seconds with return value 0
Press any key to continue . . .
```

```

point.c
1 //calloc() in dynamic memory allocation
2 #include<stdio.h>
3 #include<conio.h>
4 #include<stdlib.h>
5 int main()
6 {
7     int range, *ptr;
8
9     printf("Enter Range Value : ");
10    scanf("%d",&range);
11
12    ptr = (int*)calloc(range,sizeof(int));
13
14    if(ptr == NULL)
15    {
16        printf("Unable To Allocate Memory");
17        exit(0);
18    }
19
20    printf("Enter Elements Of Array : \n");
21    for(int i = 0; i < range; i++)
22    {
23        scanf("%d",ptr+i);
24    }
25
26    printf("Array Elements Are : \n");
27    for(int i = 0; i < range; i++)
28    {
29        printf("Arr[%d] : %d : %d\n",i,*(ptr+i),(ptr+i));
30    }
31    getch();
32 }

```

```

E:\C CODE\point.exe
Enter Range Value : 5
Enter Elements Of Array :
66666
77777
88888
99999
12345
Array Elements Are :
Arr[0] : 66666 : 12217792
Arr[1] : 77777 : 12217796
Arr[2] : 88888 : 12217800
Arr[3] : 99999 : 12217804
Arr[4] : 12345 : 12217808

-----
Process exited after 39 seconds with return value 0
Press any key to continue . . .

```

```

//realloc() in dynamic memory allocation
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int range, *ptr;

    printf("Enter Range Value : ");
    scanf("%d",&range);

    ptr = (int*)malloc(range*sizeof(int));

    if(ptr == NULL)
    {
        printf("Unable To Allocate Memory");
        exit(1);
    }

    printf("Enter Element Values : \n");
    for(int i = 0; i < range; i++)
    {
        scanf("%d",ptr+i);
    }

    int Range;

    printf("Enter New Range : ");
    scanf("%d",&Range);

    ptr = (int*)realloc(ptr, Range*sizeof(int));

    if(ptr == NULL)
    {
        printf("Unable To Allocate Memory");
        exit(1);
    }

    printf("Enter New Element Values : \n");
    for(int i = range; i < Range; i++)
    {
        scanf("%d",ptr+i);
    }

    printf("Elements Values Are : \n");

    for(int i = 0; i < Range; i++)
    {
        printf("Arr[%d] : %d : %d\n",i,*(ptr+i),(ptr+i));
    }

    getch();
}

```

```

E:\C CODE\point.exe
Enter Range Value : 3
Enter Element Values :
100
200
300
Enter New Range : 6
Enter New Element Values :
400
500
600
Elements Values Are :
Arr[0] : 100 : 11431360
Arr[1] : 200 : 11431364
Arr[2] : 300 : 11431368
Arr[3] : 400 : 11431372
Arr[4] : 500 : 11431376
Arr[5] : 600 : 11431380

-----
Process exited after 19.79 seconds with return value 0
Press any key to continue . . .

```

```

point.c
1 //free() in dynamic memory allocation
2 #include<stdio.h>
3 #include<conio.h>
4 #include<stdlib.h>
5 int main()
6 {
7     int range , *ptr;
8
9     printf("Enter Range Value : ");
10    scanf("%d",&range);
11
12    ptr = (int*)malloc(range * sizeof(int));
13
14    if(ptr == NULL)
15    {
16        printf("Unable To Allocate Memory");
17        exit(1);
18    }
19
20    printf("Enter Element Values : \n");
21    for(int i = 0; i < range; i++)
22    {
23        scanf("%d",ptr+i);
24    }
25
26    printf("Element Values Are : \n");
27    for(int i = 0; i < range; i++)
28    {
29        printf("Arr[%d] : %d : %d\n",i,*(ptr+i),(ptr+i));
30    }
31
32    free(ptr); //release the memory at the end
33    ptr = NULL;
34
35    printf("Ptr Value : %d",ptr);
36
37    getch();
38 }

```

```

E:\C CODE\point.exe
Enter Range Value : 5
Enter Element Values :
100
200
300
400
500
Element Values Are :
Arr[0] : 100 : 11824576
Arr[1] : 200 : 11824580
Arr[2] : 300 : 11824584
Arr[3] : 400 : 11824588
Arr[4] : 500 : 11824592
Ptr Value : 0
-----
Process exited after 10.54 seconds with return value 0
Press any key to continue . . .

```