Unit – IV

Pointers: Concept of a Pointer, Declaring and Initializing Pointer Variables, Pointer Expressions and Address Arithmetic, Null Pointers, Generic Pointers, Pointers as Function arguments, Pointers and Arrays, Pointers and Strings, Pointer to Pointer, Dynamic Memory Allocation, Dangling Pointer, Command line Arguments.

Pointer Introduction:-

- One of the powerful features of C is ability to access the memoryvariables by their memory address.
- This can be done by using Pointers. The real power of C lies in the properuse of Pointers.
- A pointer is a variable that can store an address of a variable (i.e., 112300).We say that a pointer points to a variable that is stored at thataddress.
- A pointer itself usually occupies 4 bytes of memory (then it can addresscells from 0 to 232-1).

Advantages of Pointers:-

1. A pointer enables us to access a variable that is defined out side the function.
2. Pointers are more efficient in handling the data tables.
3. Pointers reduce the length and complexity of a program.
4. They increase the execution speed.

Definition :-

A variable that holds a physical memory address is called a pointer variable orPointer

Declaration :

Datatype * Variable-name;

```
Eg:-    int *ad; /* pointer to int */
        char *s; /* pointer to char */
        float *fp; /* pointer to float
        */

        char **s; /* pointer to variable that is a pointer to char */
```

Dangling Pointers in C

The most common bugs related to pointers and memory management is dangling/wild pointers. Sometimes the programmer fails to initialize the pointer with a valid address, then this type of initialized pointer is known as a dangling pointer in C.

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. If the memory is re-allocated to some other process, then we dereference the dangling pointer will cause the segmentation faults.

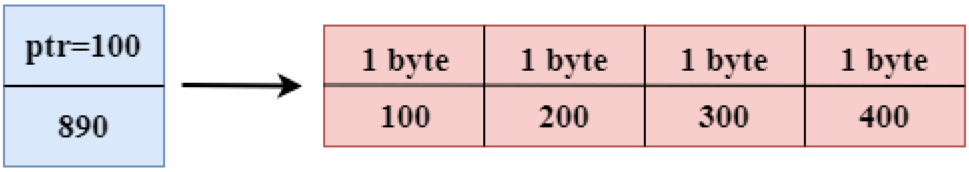
Let's understand the dangling pointer through some C programs.

Using free() function to de-allocate the memory.

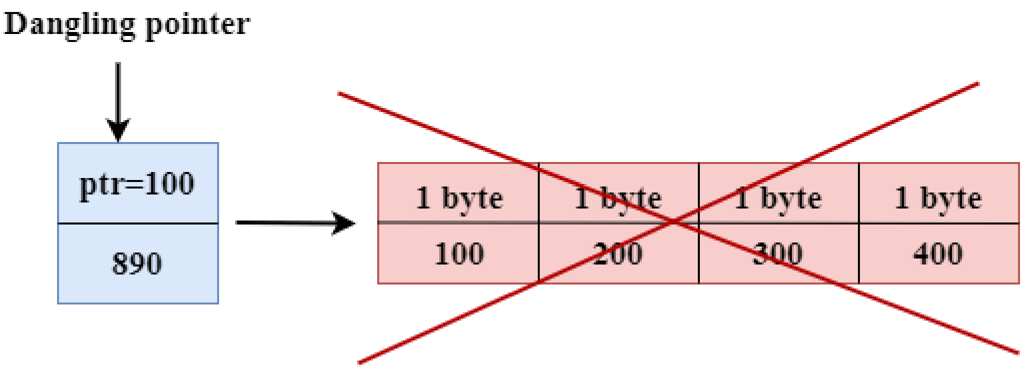
```c
#include <stdio.h>
int main()
{
  int *ptr=(int *)malloc(sizeof(int));
  int a=560;
  ptr=&a;
  free(ptr);
  return 0;
}
```

In the above code, we have created two variables, i.e., *ptr and a where 'ptr' is a pointer and 'a' is a integer variable. The *ptr is a pointer variable which is created with the help of **malloc()** function. As we know that malloc() function returns void, so we use int * to convert void pointer into int pointer.

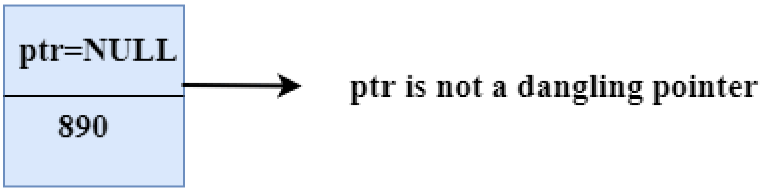
The statement **int *ptr=(int *)malloc(sizeof(int));** will allocate the memory with 4 bytes shown in the below image:



The statement **free(ptr)** de-allocates the memory as shown in the below image with a cross sign, and 'ptr' pointer becomes dangling as it is pointing to the de-allocated memory.



If we assign the NULL value to the 'ptr', then 'ptr' will not point to the deleted memory. Therefore, we can say that ptr is not a dangling pointer, as shown in the below image:



Variable goes out of the scope

When the variable goes out of the scope then the pointer pointing to the variable becomes a **dangling pointer.**

- A pointer is a variable that contains an address which is a location ofanother variable in memory.

Consider the Statement

p=&i;

Here „&‟ is called address of a variable.

‘p’ contains the address of a variable i.

The operator & returns the memory address of variable on which it isoperated, this is called Referencing.

The * operator is called an indirection operator or dereferencing operatorwhich is used to display the contents of the Pointer Variable.

Consider the following Statements :int
*p,x;

x =5;

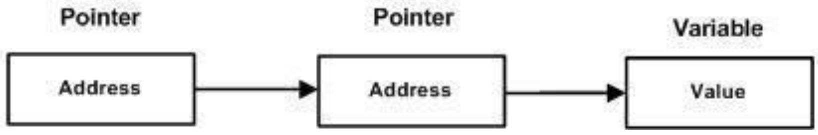
p= &x;

Assume that x is stored at the memory address 2000. Then the output for thefollowing printf statements is :

	Output
Printf(“The Value of x is %d”,x);	5
Printf(“The Address of x is %u”,&x);	2000
Printf(“The Address of x is %u”,p);	2000
Printf(“The Value of x is %d”,*p);	5
Printf(“The Value of x is %d”,*(&x));	5

- **Pointers to pointers:-**
  - A pointer to a pointer is a form of multiple indirection, or a chain ofpointers. Normally, a pointer contains the address of a variable.
  - When we define a pointer to a pointer, the first pointer contains the address of the

second pointer, which points to the location that containsthe actual value as shown below.



- A variable that is a pointer to a pointer must be declared assuch.

- This is done by placing an additional asterisk in front of itsname. For example, the following declaration declares a pointer to a pointer of type int –

int **var;

□ **Compatibility pointer:-**

The rules for assigning one pointer to another are tighter than therules for numeric types.

For example, you can assign an int value to a double variable withoutusing a type conversion, but you can't do the same for pointers to these two types. Let's see a simple C program to exemplify this.

```
*

 * ptr_compatibility.c -- program illustrates concept of pointer
 * compatibility
 */

#include <stdio.h>



int main(void)

{


    x = n;      /* implicit type conversion */
    pld = pi;   /* compile-time error: assigning pointer-to-int to */

                /* pointer-to-long-double */


}
```

□ **L value and R value:-**

L-value: "l-value" refers to memory location which identifies an object. l-value may appear as either left hand or right hand side of anassignment operator(=). l-value often represents as identifier.

Expressions referring to modifiable locations are called "**modifiable l-values**". A modifiable l-value cannot have an array type, an incompletetype, or a type with the **const** attribute

In C, the concept was renamed as **"locator value"**, and referred toexpressions that locate (designate) objects.

The l-value is one of the following:

1. he name of the variable of any type i.e, an identifier of integral,floating, pointer, structure, or union type.
2. A subscript ([ ]) expression that does not evaluate to an array.
3. A unary-indirection (*) expression that does not refer to an array
4. An l-value expression in parentheses.
5. A **const** object (a nonmodifiable l-value).
6. The result of indirection through a pointer, provided that it isn’t afunction pointer.
7. The result of member access through pointer(-> or .)

R-value: r-value” refers to data value that is stored at some addressin memory. A r-value is an expression that can’t have a value assigned to it which means r-value can appear on right but not on lefthand side of an assignment operator(=).

Note: The unary & (address-of) operator requires an lvalue as itsoperand. That is, &n is a valid expression only if n is an lvalue.

- **Arrays:-**
    - An array is defined as the collection of similar type of dataitems stored at contiguous memory locations.
    - Arrays are the derived data type in C programming language which can store the primitive type of data suchas int, char, double, float, etc.
    - It also has the capability to store the collection of deriveddata types, such as pointers, structure, etc.
    - The array is the simplest data structure where each dataelement can be randomly accessed by using its index number.
- **Pointer Arithmetic in C:-**

Pointer Arithmetic in C

- We can perform arithmetic operations on the pointers likeaddition, subtraction, etc.
- However, as we know that pointer contains the address,the result of an arithmetic operation performed on the

pointer will also be a pointer if the other operand is oftype integer.

- In pointer-from-pointer subtraction, the result will be aninteger value.
- Following arithmetic operations are possible on thepointer in C language:
  - Increment
  - Decrement
  - Addition
  - Subtraction
  - Comparison

Incrementing Pointer in C:-

- If we increment a pointer by 1, the pointer will startpointing to the immediate next location.
- This is somewhat different from the general arithmetic since the value of the pointer will get increased by the sizeof the data type to which the pointer is pointing.

The Rule to increment the pointer is given below:

new_address= current_address + i * size_of(data type)

Where i is the number by which the pointer get increased.

Decrementing Pointer in C

- Like increment, we can decrement a pointer variable. Ifwe decrement a pointer, it will start pointing to the previous location.
  - The formula of decrementing the pointer is given below:

new_address= current_address - i * size_of(data type)

C Pointer Addition

We can add a value to the pointer variable. The formula ofadding value to pointer is given below:

new_address= current_address + (number * size_of(data type))

C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer willgive an address. The formula of subtracting value from the pointer variable is given below:

new_address= current_address - (number * size_of(data type))

- **Memory Allocation Function:-**
  - The concept of **dynamic memory allocation in c language** *enables the C programmer to allocate memoryat runtime*.
  - Dynamic memory allocation in c language is possible by 4functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Before learning above functions, let's understand the differencebetween static memory allocation and dynamic memory allocation.

static memory allocation	dynamic memory allocation
memory is allocated at compiletime.	memory is allocated at run time.

memory can't be increased whileexecuting program.	memory can be increased whileexecuting program.
used in array.	used in linked list.

Now let's have a quick look at the methods used for dynamic memoryallocation.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc()functions.
free()	frees the dynamically allocated memory.

- **Array of Pointers**:-

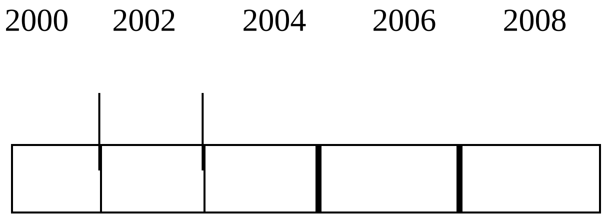
In computer programming, an array of pointers is an indexed set of variables in which the variables are pointers (a referenceto a location in memory).

- Pointers are an important tool in computer science forcreating, using, and destroying all types of data structures.
- An array of pointers is useful for the same reason that allarrays are useful: it allows you to numerically index a large set of variables.
- Below is an array of pointers in C that sets each pointer inone array to point to an integer in another and then print the values of the integers by dereferencing the pointers.

POINTERS WITH ARRAYS:-

When an array is declared, elements of array are stored in contiguous locations. The address of the first element of an array is called its base address.

Consider the array



a[0]   a[1]   a[2]   a[3]   a[4]

The name of the array is called its base address.

i.e., a and k& a[20] are equal

Now both a and a[0] points to location 2000. If we declare p as an integer pointer, then we can make the pointer P to point to the array a by followingassignment.

P = a;

We can access every value of array a by moving P from one element to another.i.e.,

P	points to 0th element
P+1	points to 1st element
P+2	points to 2nd element
P+3	points to 3rd element
P +4	points to 4th element

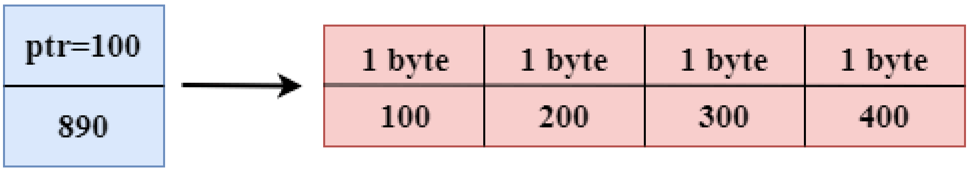
Let's understand the dangling pointer through some C programs.

Using free() function to de-allocate the memory.

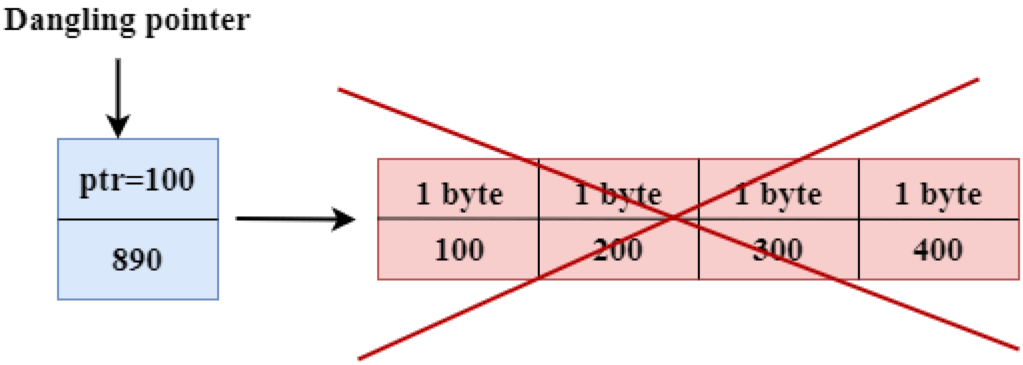
```
#include <stdio.h>
int main()
{
  int *ptr=(int *)malloc(sizeof(int));
  int a=560;
  ptr=&a;
  free(ptr);
  return 0;
}
```

In the above code, we have created two variables, i.e., *ptr and a where 'ptr' is a pointer and 'a' is a integer variable. The *ptr is a pointer variable which is created with the help of **malloc()** function. As we know that malloc() function returns void, so we use int * to convert void pointer into int pointer.

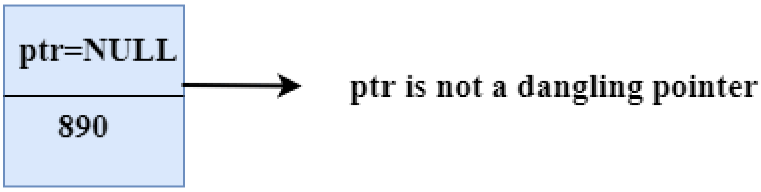
The statement **int *ptr=(int *)malloc(sizeof(int));** will allocate the memory with 4 bytes shown in the below image:



The statement **free(ptr)** de-allocates the memory as shown in the below image with a cross sign, and 'ptr' pointer becomes dangling as it is pointing to the de-allocated memory.



If we assign the NULL value to the 'ptr', then 'ptr' will not point to the deleted memory. Therefore, we can say that ptr is not a dangling pointer, as shown in the below image:



Variable goes out of the scope

When the variable goes out of the scope then the pointer pointing to the variable becomes a **dangling pointer.**

Command Line Arguments in C

The arguments passed from command line are called command line arguments. These arguments are handled by main() function.

To support command line argument, you need to change the structure of main() function as given below.

```
int main(int argc, char *argv[] )
```

Here, **argc** counts the number of arguments. It counts the file name as the first argument.

The **argv[]** contains the total number of arguments. The first argument is the file name always.

Example

Let's see the example of command line arguments where we are passing one argument with file name.

```
#include <stdio.h>

void main(int argc, char *argv[] )  {



  printf("Program name is: %s\n", argv[0]);



  if(argc < 2){

    printf("No argument passed through command line.\n");

  }

  else{

    printf("First argument is: %s\n", argv[1]);

  }

}
```

Output:

```
Program name is: program

First argument is: hello
```

If you pass many arguments, it will print only one.

1. ./program hello c how r u

Output:

```
Program name is: program

First argument is: hello
```