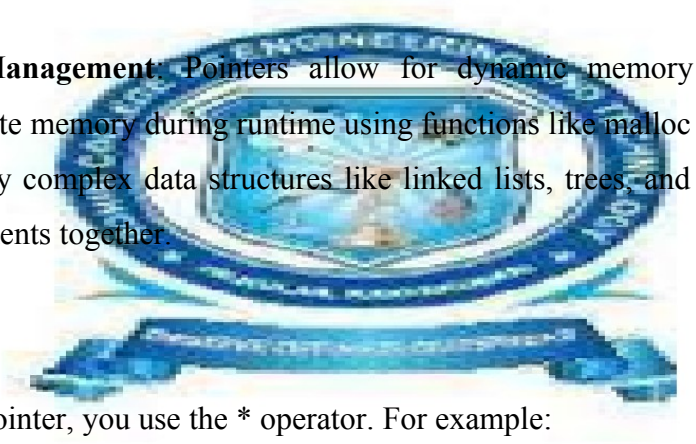
4.1.POINTERS:

A pointer is a variable that stores the memory address of another variable. Instead of holding a data value directly, a pointer holds the location where that value is stored.

C pointer is the derived data type that is used to store the address of another variable and can also be used to access and manipulate the variable's data stored at that location. The pointers are considered as derived data types.

a) Why Use Pointers?

1. **Efficiency**: Pointers can be more efficient for certain operations, especially when dealing with large data structures. Instead of copying entire structures, you can pass around pointers.
2. **Dynamic Memory Management**: Pointers allow for dynamic memory allocation, meaning you can allocate memory during runtime using functions like malloc or new.
3. **Data Structures**: Many complex data structures like linked lists, trees, and graphs rely on pointers to link elements together.



b) Basic Pointer Syntax

1. Declaration: To declare a pointer, you use the * operator. For example:

```
int *p;       // p is a pointer to an int
```

Example of Valid Pointer Variable Declarations

Take a look at some of the valid pointer declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
```

```
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

2.Initialization: You can initialize a pointer to the address of a variable using the address-of operator &:

```
int a = 10;
p = &a;      // p now holds the address of a
```

Example

Here is an example of pointer initialization –

```
int x
int *
```

Here, **x** is an integer variable, **ptr** is an integer pointer. The pointer **ptr** is being initialized with **x**.

c) Referencing and Dereferencing Pointers

A pointer references a location in memory. Obtaining the value stored at that location is known as **dereferencing the pointer**.

In C, it is important to understand the purpose of the following two operators in the context of pointer mechanism −

The & Operator − It is also known as the "Address-of operator". It is used for Referencing which means taking the address of an existing variable (using **&**) to set a pointer variable.

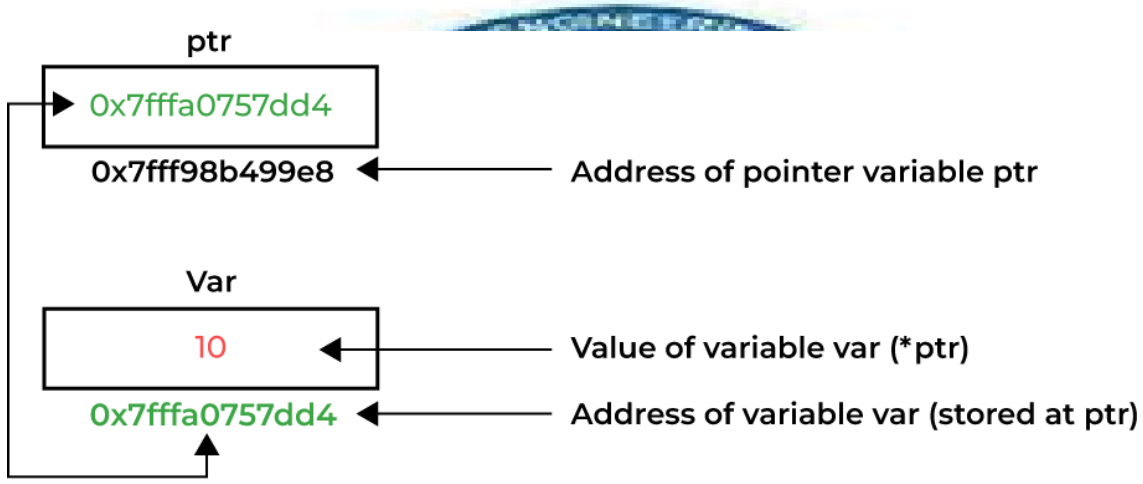
The * Operator − It is also known as the "dereference operator". **Dereferencing** a pointer is carried out using the *** operator** to get the value from the memory address that is pointed by the pointer.

Pointers are used to pass parameters by reference. This is useful if a programmer wants a function's modifications to a parameter to be visible to the function's caller. This is also useful for returning multiple values from a function.

Dereferencing: To access or modify the value at the address stored in a pointer, you use the dereference operator *:

```
int value = *p; // value is now 10
*p = 20; // a is now 20
```

Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. We use the same **(*) dereferencing operator** that we used in the pointer declaration.



How to use pointers

We can use pointers with any type of variable such as integer, float, string, etc. You can also use pointers with derived data types such as array, structure, union, etc.

Example

In the below example, we are using pointers for getting values of different types of variables.

```c
#include <stdio.h>

int main() {
  int x = 10;
  float y = 1.3f;
  char z = 'p';

  // Pointer declaration and initialization
  int * ptr_x = & x;
  float * ptr_y = & y;
  char * ptr_z = & z;

  // Printing the values
  printf("Value of x = %d\n", * ptr_x);
  printf("Value of y = %f\n", * ptr_y);
  printf("Value of z = %c\n", * ptr_z);

  return 0;
}
```

Output

Value of x = 10

Value of y = 1.300000

Value of z = p

PASSING ARGUMENTS BY ADDRESS

Basics of Argument Passing

1. **Pass by Value**: When you pass an argument by value, the function receives a copy of the variable. Any changes made to this variable inside the function do not affect the original variable.
2. **Pass by Address**: When you pass an argument by address (or by reference), you provide the function with the memory address of the variable. This allows the function to access and modify the original variable.

How It Works

1. **Using Pointers**: In languages like C/C++, you use pointers to pass by address. A pointer is a variable that stores the memory address of another variable.
2. **Function Definition**: When defining a function that takes an argument by address, you use a pointer type for that argument.

```
void modifyValue(int *ptr) {
    *ptr = 10;  // Dereference the pointer to modify the original value
}
```

3. **Calling the Function**: When calling the function, you pass the address of the variable using the address-of operator (&).

```
int main() { int num = 5;
modifyValue(&num);  //
Passthe address of num
printf("%d\n", num);  //
Output will be 10
return 0;
}
```

Advantages of Passing by Address

1. **Efficiency**: Passing large data structures (like arrays or structs) by address can be more efficient than passing by value because it avoids copying large amounts of data.
2. **Modification**: Functions can modify the original variable, which can be useful for returning multiple values from a function or changing the state of an object.
3. **Dynamic Memory Management**: It allows functions to manage dynamic memory by passing pointers, which can be allocated and freed inside the function.
4. Passing the pointers to the function means the memory location of the variables is passed to the parameters in the function, and then the

operations are performed. The function definition accepts these addresses using pointers, addresses are stored using pointers.

Program to swap two numbers by using pass by reference method

// C program to swap two values using pass by reference

```
#include <stdio.h>
  void swap(int* a, int* b)
{int temp; temp = *a;
  *a = *b;
  *b = temp;
}
  int main()
{
  int a = 10, b = 20;
  printf("Values before swap
        function are: %d, %d\n", a,
        b);
  swap(&a, &b);
  printf("Values after swap
        function are: %d, %d", a,
        b);
  return 0;
}
```

Output

```
Values before swap function are: 10, 20

Values after swap function are: 20, 10
```