```
    return 0;
}
```

Output:

```
Original pointer value: 0x7ffe88bfe9d0
After addition, pointer value: 0x7ffe88bfe9d8
After subtraction, pointer value: 0x7ffe88bfe9d4
```

In this example, ptr += 2 moves the pointer two positions forward, and ptr -= 1 moves the pointer one position backward.

2.4 Memory Allocation (Static and Dynamic)

Memory allocation refers to the process of reserving memory space for variables or objects during program execution. In C++, memory allocation can be categorized into two types: static memory allocation and dynamic memory allocation.

1. Static Memory Allocation:

In static memory allocation, memory is allocated at compile-time, and the size of memory is fixed throughout the program execution. Compile time refers to the period when the programming code (such as C++) is converted to the machine code (i.e. binary code).

Example:

```
int arr[5]; // Static allocation of an array of 5 integers
```

- Memory for arr is allocated on the stack.
- Memory size is fixed and determined during compile-time.
- Memory is automatically deallocated when the scope containing the variable ends.

2. Dynamic Memory Allocation:

In dynamic memory allocation, memory is allocated at runtime, and the size of memory can be determined during program execution. Runtime is the period of time when a program is actually running and occurs after compile time.

Dynamic Memory Allocation using new operator:

Single Object Allocation:

```
int *ptr = new int; // Allocates memory for a single integer dynamically
```

- Memory for ptr is allocated on the heap.
- Memory size can be determined during runtime.
- Memory needs to be explicitly deallocated using the delete operator to prevent memory leaks.

Array Allocation:

```
int *arr = new int[5]; // Allocates memory for an array of 5 integers dynamically
```

- Memory for arr is allocated on the heap.
- Size of the array is determined during runtime.
- Individual elements of the array can be accessed using pointer notation.

Deallocating Dynamically Allocated Memory:

Single Object Deallocation:

```
delete ptr; // Deallocates memory allocated for a single variable
```

Array Deallocation:

```
delete[] arr; // Deallocates memory allocated for an array
```

- The delete operator is used to deallocate memory previously allocated using the new operator.
- For arrays, the delete[] operator is used to deallocate memory allocated for the entire array.

Comparison:

- Static memory allocation is simpler and faster compared to dynamic memory allocation.
- Dynamic memory allocation provides flexibility in memory management but requires manual deallocation to prevent memory leaks.
- Static memory allocation is suitable for scenarios where memory size is known in advance, while dynamic memory allocation is preferred when the size of memory is determined during runtime or when memory needs to be allocated and deallocated dynamically.

2.5 Dynamic Memory Management using new and delete Operators

Dynamic memory management in C++ allows the allocation and deallocation of memory during runtime. This is useful for managing memory that cannot be determined at compile time. The new operator is used to allocate memory on the heap, and the delete operator is used to deallocate that memory.

1. Allocating Memory Dynamically:

Memory can be allocated dynamically using the new operator in C++.

Syntax:

```
data_type *pointer_name = new data_type;
```

Example:

```
int *ptr = new int; // Allocates memory for an integer dynamically
```

In the example above, memory is allocated on the heap to store an integer value, and the address of the allocated memory is assigned to the pointer ptr.

2. Releasing Memory using delete:

After dynamically allocating memory, it's essential to release it when it's no longer needed to prevent memory leaks.

Syntax:

```
delete pointer_name;
```

Example:

```
delete ptr; // Deallocates memory allocated for the integer pointed by ptr
```

In the example above, the delete operator is used to deallocate the memory allocated for the integer pointed to by ptr. After deletion, the memory is returned to the system for reuse.

Example:

```
#include <iostream>
using namespace std;

int main() {
    // Dynamic memory allocation
    int *ptr = new int; // Allocates memory for an integer dynamically
    *ptr = 10; // Assigns a value to the dynamically allocated memory

    // Accessing and printing the dynamically allocated value
    cout << "Dynamically allocated value: " << *ptr << endl;

    // Deallocating dynamically allocated memory
    delete ptr;

    return 0;
}
```

In this example:

- Memory for an integer is allocated dynamically using the new operator.
- The value 10 is assigned to the dynamically allocated memory.
- The value stored in the dynamically allocated memory is printed.
- Finally, the dynamically allocated memory is deallocated using the delete operator.

Notes:

- Dynamically allocated memory remains allocated until explicitly deallocated using the delete operator.
- Failure to deallocate dynamically allocated memory can lead to memory leaks, causing the program to consume more memory than necessary.
- It's essential to deallocate dynamically allocated memory when it's no longer needed to ensure efficient memory usage.

2.6 Pointer to an Object

In C++, pointers can also be used to point to objects of classes. This allows for dynamic allocation of objects and provide flexibility in memory management.

1. Creating Pointers to Objects:

Pointers to objects are declared similarly to pointers to primitive data types.