C++ Pointers Research and Report

Tulgar Cem Güngör

January 4, 2025

1 Introduction

In this report, I explore C++ pointers, their types, and how they are utilized in C++ programming. I will discuss various smart pointers, memory management, garbage collection mechanisms, and the pros and cons of raw pointers.

2 Pointers

2.1 Raw Pointer

A raw pointer is a basic variable that stores the memory address of another variable. It is very essential that we manage our memory. Every variable is located in the memory with its address. To get the memory address of a variable, we use the `&` operator.
Raw pointers in C++ can be used as follows:

- int* ptr = &value; – *both pointer and value can be changed.*
- const int* ptr = &value; – *pointer's pointed value can't be changed but pointer can point to a new object.*
- int const* ptr = &value; – *same as* const int* ptr = &value;.
- const int const* ptr = &value; – *both pointer and value can't be changed.*
- int* const ptr = &value; – *value can be changed, but pointer cannot.*

2.2 Void Pointer (Generic Pointer)

The `void pointer` is used with no specified data types. It can point to any type of data. Void pointers can not be dereferenced, that is, we cannot reach pointed value. When we want to take the pointed value by the pointer, we need to make type casting according to the variable's type.

2.3 Nullptr

It was introduced in C++11. Before C++11, `NULL` was used to defining a pointer which didn't point any object. However, `NULL` could cause some problems because it is defined as `0`. Therefore, editor may assume `NULL` as zero and it may lead to some problems. Because of this ambiguous situation, `nullptr` which defined as `std::nullptr_t` has been started to use.

2.4 Garbage Collection Mechanism

There are four parts in the memory. Two of them are stack and heap. Stack is more safe, but heap is more useful than stack. Therefore, complex applications require the use of a heap. Also, for using heap, pointers or STL containers must be used. In the heap, allocated parts of the code with new keyword must be de-allocated with delete command when their tasks were ended. In C#, `garbage collection mechanism` do that automatically; however, in the C++, code authors have to do that manually with using delete command.
Manual memory management can lead to several issues:

- Forgetting to deallocate memory can lead to memory leaks. The object which isn't destroyed after it is used occupies unnecessary place at the memory and this situation is named with `Memory Leak`.
- As a result of trying to reach an index that isn't exist in the array, `Buffer Overflow` problem occurs. To avoid this issue, STL Containers, like std::array or std::vector, must be used.
- When a raw pointer is created but isn't pointed any object, this pointer is named with `Wild Pointer` because this pointer points arbitrary memory location and this may cause a problem.
- After a pointer deallocated, function that a pointer points its parameter ended or a scope where the pointer points an object ended; if the deallocated object is called by the pointer, the program gives error because this is start to being `Dangling Pointer` after it deallocated.

On one hand, garbage coolection prevents these issues. On the other hand, it causes some slowings in the program. It is very effective way that using the garbage collection nevertheless. However, in the C++, there isn't garbage collection mechanism. Instead of garbage collection, we can use smart pointers in the C++.

2.5 Smart Pointers

Smart pointers help manage memory allocation and deallocation automatically, reducing the risk of memory leaks and other data problems. They include:

2.5.1 auto_ptr

Auto pointer was created to prevent some data problems such as memory leaks. However, it removed with C++17 because of some reasons. Firstly, there is no any particular function to copy an auto_ptr. When we copy an auto_ptr, other becomes nullptr and this may cause some problems in the program. Secondly, auto_ptr can not be used with STL Containers, such as std::vector or std::list because auto_ptr can not copy. As a result, other smart pointers such as shared_ptr or unique_ptr are introduced in C++11 to solve this problems.

2.5.2 sharedl_ptr

Used for shared ownership of an object. It keeps track of the number of shared_ptr objects pointing to the same object. When the last one is destroyed, the object is deallocated.

2.5.3 unique_ptr

Ensures that only one pointer owns the object. Ownership can be transferred using move(). For giving parameter to a function we need to use move() function while we are using unique_prt. Furthermore, after give a parameter we can't use unique_prt because we have changed the referance. However, we can give parameter directly and also we can use shared_ptr even we give a parameter to a function.

2.5.4 weak_ptr

It doesn't affect to that it points an object. It is generally used with shared_ptr. If there aren't any shared_ptr to point an object, weak_ptr is automatically destroyed. Each unique_ptr have their own ownership, that is, they don't share their ownerships; however, shared_ptr isn't like this. shared_ptr can share their ownerships with other shared pointers. weak_ptr hasn't any ownership like raw pointer. Weak pointer is necessery at the circular referance situations. At the circular links, there are two object whicha point each other with shared_ptr. In this case, these objects couldn't be deleted and it leads to memory leak.

2.6 Constructor & Destructor

`Constructors` are special member functions used to initialize objects of a class, while `destructors` are used to free resources when an object is destroyed. The destructor is important for preventing memory leaks.

3 Conclusion

Pointers are an essential part of C++ programming. They allow efficient memory management and enable complex data structures. However, raw pointers can lead to several issues, including memory leaks, dangling pointers, and buffer overflows. Smart pointers, on the other hand, provide safer alternatives by managing memory automatically.

```
--------------MY SMART POINTER--------------

MyClass Constructed has launched.
SmartPointer Constructed has launched.
SmartPointer: 0x5bee1e8512c0
SmartPointer Value: 10
SmartPointer Value: 10
MyCLass Destructer has launched.
SmartPointer Destructer has launched.

--------------SHARED POINTER--------------

Shared Pointer: 0x5bee1e8512d0
Shared Pointer Value: 20
Shared Pointer Referance Number: 3
Shared Pointer New Referance Number: 2
Moved Shared Pointer Value: 25
Shared Pointer Last Referance Number: 1

--------------UNIQUE POINTER--------------

Unique Pointer 1: 0x5bee1e8512c0
Unique Pointer 1 Value: 30
Moved Unique Pointer: 0x5bee1e8512c0
Moved Unique Pointer Value: 30

--------------WEAK POINTER--------------

Weak Pointer: 0x5bee1e8512d0
werakPtr1 is moved.
New Weak Pointer Value: 40
```