A Comparative Study of Pointers

Yifei Guan
Nanjing University
Nanjing, China
231220044@smail.nju.edu.cn

Abstract

Pointers, as an important programming concept, have different implementations and applications across various programming languages. This paper aims to compare and analyze the usage of pointers in programming languages such as C/C++, Go, Java and Python.

ACM Reference Format:
Yifei Guan. 2023. A Comparative Study of Pointers. In *Proceedings of (PLP Workshop '23).* ACM, New York, NY, USA, 2 pages. https://doi.org/404.404

1 Introduction

Pointers are a type of variable that stores a memory address and can be used to access and manipulate data stored at that address. In programming, pointers offer several advantages, such as direct memory access, dynamic memory allocation, and passing data between functions. However, due to the flexibility and complexity of pointers, they can also pose security issues, such as dangling pointers and memory leaks. Different programming languages have different ways of handling pointers and rules regarding their usage. In this paper, we will delve into the differences and usage of pointers in various programming languages. We will explore the characteristics, advantages, and limitations of pointers in different languages. Through comparative analysis, we can better understand how different programming languages handle pointers and provide developers with a more comprehensive perspective and guidance to make informed choices in practical applications. In conclusion, this paper aims to explore the differences in pointers among different programming languages and compare and analyze them. By studying various languages, we can gain a better understanding of the concept and usage of pointers and provide developers with guidance and directions for consideration.

PLP Workshop '23, Dec 23, 2023, Nanjing,Jiangsu

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/404.404

2 Features

In the following section, I will present examples illustrating the applications of pointers in different programming languages. By analyzing these examples, we can examine the unique features and distinctions of pointers across these languages.

2.1 Pointers in C/C++

In C/C++, a pointer is a variable type that stores a memory address. Through pointers, we can directly access and manipulate the data stored at that address. The declaration of a pointer requires specifying the data type it points to, ensuring that the pointer correctly dereferences and operates on the data. Pointers can also be used to pass data between functions. By passing a pointer as a parameter to a function, we can achieve modification of the original data instead of just passing a copy. This is highly efficient when dealing with large amounts of data or when in-place modification is required. However, the use of pointers requires caution because incorrect pointer operations can lead to program crashes or unpredictable results. Common pointer issues include dangling pointers (pointing to invalid memory addresses) and memory leaks (failure to properly deallocate allocated memory).

```
int *ptr;  // Declare a pointer to an
    integer variable
int num = 10;
int *ptr = &num;  // Initialize the
    pointer ptr with the address of num
*ptr = 20;// After modification, the
    value of num is: 20
int *ptr = new int;  // Dynamically
    allocate memory space for an integer
     variable
*ptr = 10;  // Store a value in the
    memory pointed to by the pointer
delete ptr;  // Release the dynamically
    allocated memory space
```

2.2 Pointers in Go

Unlike some other programming languages, pointers in Go have similar usage to regular variables. We can use pointers to access and modify the value of variables, and we can also pass pointers as arguments to functions, and so on.

It's important to note that in Go language, pointers cannot be used for pointer arithmetic (i.e., adding or subtracting). This limitation is in place to avoid illegal memory access and overflow issues.

```
var num int = 10
var ptr *int = &num
*ptr = 20// Modify the value of the
   variable pointed to by the pointer
fmt.Println(num)  // Output: 20
func updateNum(ptr *int) {
        // Modify the value of the
           variable pointed to by the
           pointer
        *ptr = 20
}
var num int = 10
updateNum(&num)
fmt.Println(num)  // Output: 20
```

2.3 References in Java

In Java, unlike languages such as C or C++, there is no direct concept of pointers. Memory management in Java is handled automatically by the garbage collector, so developers do not need to manage memory manually. However, Java has a similar mechanism to pointers, called references.In Java, references are an essential concept used for working with objects. Unlike primitive types (such as int, char, etc.), which store the actual value, references store the address of an object in memory. In Java, when you create an object using the new keyword, memory is allocated on the heap to hold the object's data. The reference variable holds the memory address where the object is stored, allowing you to access and manipulate the object's properties and methods.

```
class Person {
        String name;
        void sayHello() {
                System.out.println("
                   Hello, my name is "
                   + name);
        }
}
public class Main {
        public static void main(String[]
            args) {
                Person person = new
                   Person();  //
                   Declare and create a
                    reference to a
                   Person object
                person.name = "John";
                   // Set the value of
                   the object's
                   attribute
                person.sayHello();  //
                   Call the object's
                   method
        }
}
```

2.4 References in Python

Python, like Java, does not have a direct concept of pointers, but it has references. When we create an object in Python, it allocates a block of memory to store the object and returns a reference to that object. Variables store references to objects, not the objects themselves. Therefore, we can manipulate objects through their references. Unlike in C/C++, references in Python do not need to be explicitly declared or released. When there are no references pointing to an object, the garbage collector will reclaim the memory occupied by that object. It is important to note that in Python, the type of an object can be changed. For example, we can convert an integer object into a string object. This means that the same object can have multiple references and different references can point to objects of different types. This dynamic feature is one of the characteristics of the Python language.

```
def update_list(lst):
lst[0] = 100
my_list = [1, 2, 3]
update_list(my_list)
print(my_list)  # output: [100, 2, 3]
x = [1, 2, 3]
y = x
y.append(4)
print(x)  # output: [1, 2, 3, 4]
x = 10
print(type(x))  # output: <class 'int'>
x = 'Hello'
print(type(x))  # output: <class 'str'>
```

3 Conclusion

Overall, C/C++ provides the lowest-level memory access and manipulation capabilities, but requires programmers to manage memory manually. Go language maintains flexibility while increasing code readability and maintainability by limiting pointer operations and introducing receiver pointers. Java and Python provide higher-level abstractions, automatically manage memory, and offer greater security. Choosing the appropriate programming language and using pointers or references appropriately depends on specific needs and application scenarios.