Object Oriented Programming with C++ HW

2024 Spring Semester

21 CST H3Art

Introduction

```
class vector {
private:
  int *v;
  int size;

public:
  vector(int i) {
    size = i;
    v = new int[i];
  }
  ~vector() { delete[] v; }
};
```

1. Define the copy constructor of the class `vector`.
2. Overload the "`=`"(assignment), "`+`"(plus), "`[]`"(subscript) operators for the class `vector`.
   The subscript operator is used to access the ith element in the vector. For example, after overload the `[]` operator we can use following statements to assign values to the elements of a `vector v1`.

   ```
   vector v1(10);
   v1[0] = 100, v1[2] = 50;
   ```

Solution

1. To implement the **copy constructor** of the class `vector`, we need to:
   - Pass the copied object's **reference** to the constructor, it will not be modified so we define the parameter as `const`.
   - **Reallocate the memory space** for new object's pointer member.
   - **Duplicate the memory space** from the copied instance to the new instance.
2. To overload the `=` **operator** for class `vector`, we need to:
   - Pass the copied object's **reference** to the overloading function, it will not be modified so we define the parameter as `const`.
   - Check whether the **assigning object** and the **assigned object** are the **same object**, if same, we directly return the value where `this` pointer points to.
   - **Reallocate the memory space** for assigned object's pointer member and **duplicate the value** from the assigning object to the assigned object.
   - Finally **return** the assigned instance's **value or reference**.

3. To overload the `+` **operator** for class `vector`, we need to:
    - Pass the another one's **reference** to the overloading function, it also will not be modified so we define the parameter as `const`.
    - Check whether the `vector`s' **attribute** `v` **have the same size**, if not, throw out a exception to alert the user.
    - Create a new `vector` object to **store the addition result**, perform addition between `this` object's `v` and another object's `v` **element by element**.
    - Finally **return its value instead of reference**, because the temporary local `vector` variable will be cleared after the function executed.
4. To overload the `[]` **operator** for class `vector`, we need to:
    - Pass the **integer index** of which we want to access
    - Check the whether the **index is out of range**.
    - To achieve assigning value to the element, we need to **return a reference** of the accessing element instead of the value.

The complete code framework is as follows:

```
#include <iostream>
#include <stdexcept>

class vector {
private:
  int *v;
  int size;

public:
  vector(int i) {
    size = i;
    v = new int[i];
  }
  ~vector() { delete[] v; }

  // Copy constructor
  vector(const vector &other) {
    size = other.size;
    v = new int[size];
    for (int i = 0; i < size; i++) {
      v[i] = other.v[i];
    }
  }

  // Overload the operator =
  vector &operator=(const vector &other) {
    if (this == &other) {
      return *this; // Prevent the self-assignment
    }

    // Clear old values
    delete[] v;
```

```
    size = other.size;
    v = new int[size];

    for (int i = 0; i < size; i++) {
      v[i] = other.v[i];
    }

    return *this;
  }

  // Overload the operator +
  vector operator+(const vector &other) {
    if (size != other.size) {
      throw std::invalid_argument("Vectors must have the same size for addition");
    }

    vector result(size);

    for (int i = 0; i < size; i++) {
      result.v[i] = v[i] + other.v[i];
    }

    // Return the instance of result √
    // Return the reference of result -> Undefined behavior ×
    return result;
  }

  // Overload the operator []
  int &operator[](int index) {
    if (index < 0 || index >= size) {
      throw std::out_of_range("Index out of range");
    }

    // Return the reference for assignment
    return v[index];
  }

  // Operator << can only be overloaded as friend function
  friend std::ostream &operator<<(std::ostream &, const vector &);
};

// Overload the ostream operator << for convenient test
std::ostream &operator<<(std::ostream &os, const vector &vec) {
  os << "Size of vector: " << vec.size << '\n' << "Vector: {";
  int len = vec.size;
  for (int i = 0; i < len; i++) {
    if (i < len - 1)
      std::cout << vec.v[i] << ", ";
    else
      std::cout << vec.v[i] << "}";
  }
```

```
  return os;
}

int main(int argc, char *argv[]) {
  vector v1(3);
  // Test operator [] assigment
  v1[0] = 100, v1[1] = 114514, v1[2] = 1919810;
  // Test operator [] access
  std::cout << v1[1] << ' ' << v1[2] << std::endl;

  // Test copy constructor
  vector v2 = v1;
  // Test operator +
  vector v3 = v1 + v2;

  std::cout << v1 << '\n' << v2 << '\n' << v3 << std::endl;

  // Test error handling
  vector *v4 = new vector(4);
  try {
    std::cout << v3 + *v4;
  } catch (std::invalid_argument e) {
    std::cout << e.what() << std::endl;
  }
  delete v4;

  return 0;
}
```

The output of above code is:

```
114514 1919810
Size of vector: 3
Vector: {100, 114514, 1919810}
Size of vector: 3
Vector: {100, 114514, 1919810}
Size of vector: 3
Vector: {200, 229028, 3839620}
Vectors must have the same size for addition
```