题目描述

用数组实现一个简单的 ArrayList 类，要求支持**模板泛型**。数组的初始大小为 10。

数组填满之后再增加元素时需要进行扩容，按照1.5倍扩容（使用 `oldCapacity + (oldCapacity / 2)` 或 `oldCapacity + (oldCapacity >> 1)`），数组容量不能减小。

具体地，你需要实现以下的方法，请确保函数名保持一致：

- `add(x)`: 在数组的末尾增加x，无返回值
- `remove(x)`: 删除**第一个**值为x的元素，如果数组中包含多个x，只删除第一个，后面元素往前移动，无返回值；有可能删除的数在数组中并不存在，此时什么也不用做

  > 注：无需考虑浮点数精度问题

- `get(x)`:输出索引位置为x的元素的值,如果该位置没有元素或者索引不合法，返回`std::nullopt`
- `getSize()`: 返回数组中的实际元素个数，返回值为int类型
- `getCapacity()`:返回数组的容量，返回值为int类型

编程要求

1. 你需要新建`ArrayList.h`，在其中编写ArrayList类。在本题中，你**只需要实现构造函数和简单的封装**，不需要使用继承等复杂的OO编程机制也不需要考虑五三原则。
2. 将下面的`Main.cpp`代码复制到CPPOJ的`Main.cpp`中，这些代码包含了题目的测试用例，你可以参考测试用例来完成你的ArrayList类，但**严禁修改**`Main.cpp`文件中的内容，一旦发现，**本题 0 分！**
3. 本题不允许使用 STL 中的容器（包括但不限于 `string`、`vector` 和 `list`），一旦发现，**本题 0 分！**
4. 本题不需要你来实现输入输出。

使用std::optional

在get函数中，我们要求使用`std::optional`来处理可能的没有元素或者索引不合法的情况。为了不给大家带来太多困难，做出如下提示：

```
std::optional<T> get(int index) {
    if (索引不合法) {
        return std::nullopt;  // 返回nullopt表示索引无效
    }
    // 正常返回index位置的元素即可
}
```

框架代码与测试代码

ArrayList.h

```
#ifndef ARRAYLIST_H
#define ARRAYLIST_H

#include <optional>
#include <cstring>
#include <iostream>

// TODO: 在这里编写你的ArrayList类

#endif // ARRAYLIST_H
```

Main.cpp

```
// 不要修改Main.cpp
#include "ArrayList.h"
#include <cassert>
#include <functional>
#include <iostream>

// === TEST_CASES ===
void TEST_1();
void TEST_2();
void TEST_3();
void TEST_4();
void TEST_5();
void TEST_6();
void TEST_7();
void TEST_8();
void TEST_9();
void TEST_10();

#define REGISTER_TEST_CASE(name) {#name, name}

int main() {
  std::unordered_map<std::string, std::function<void()>>
      test_functions_by_name = {
          REGISTER_TEST_CASE(TEST_1), REGISTER_TEST_CASE(TEST_2),
          REGISTER_TEST_CASE(TEST_3), REGISTER_TEST_CASE(TEST_4),
          REGISTER_TEST_CASE(TEST_5), REGISTER_TEST_CASE(TEST_6),
          REGISTER_TEST_CASE(TEST_7), REGISTER_TEST_CASE(TEST_8),
          REGISTER_TEST_CASE(TEST_9), REGISTER_TEST_CASE(TEST_10),
      };

  std::string test_case_name;
  std::cin >> test_case_name;
  auto it = test_functions_by_name.find(test_case_name);
  assert(it != test_functions_by_name.end());
  auto fn = it->second;
  fn();
  return 0;
}

void TEST_1() { // 测试add函数
  ArrayList<int> a;
  a.add(1);
  a.add(2);
```

```
  a.add(3);

  assert(a.getSize() == 3);
  assert(a.getCapacity() == 10); // 容量为10

  assert(a.get(0).value() == 1);
  assert(a.get(1).value() == 2);
  assert(a.get(2).value() == 3);
}

void TEST_2() { // 测试add函数，支持泛型
  ArrayList<float> a;
  a.add(1.5);
  a.add(2.33);
  a.add(3.45);
  float sum = a.get(0).value() + a.get(1).value() * a.get(2).value();
  assert(sum > 9.5 && sum < 9.9);
  assert(a.getSize() == 3);
  a.add(3.55);
  assert(a.getSize() == 4);
}

void TEST_3() { // 测试remove函数
  ArrayList<int> a;
  a.add(10);
  a.add(20);
  a.add(30);
  a.remove(40);
  a.remove(20);
  assert(a.getSize() == 2);
  assert(a.get(0).value() == 10);
  assert(a.get(1).value() == 30);

  a.remove(10);
  assert(a.getSize() == 1);
  assert(a.get(0).value() == 30);

  a.remove(30);
  assert(a.getSize() == 0);
}

void TEST_4() { // 测试get函数
  ArrayList<int> a;
  a.add(100);
  a.add(200);

  // 有效索引
  assert(a.get(0).value() == 100);
  assert(a.get(1).value() == 200);

  // 无效索引
  assert(a.get(2) == std::nullopt);
  assert(a.get(-1) == std::nullopt);
}

void TEST_5() { // 测试扩容
  ArrayList<int> a;
  for (int i = 0; i < 100; ++i) {
```

```
    a.add(i);
  }
  assert(a.getSize() == 100);
  assert(a.getCapacity() == 109);

  for (int i = 0; i < 100; ++i) {
    assert(a.get(i).value() == i);
  }
  for (int i = 0; i < 100; ++i) {
    a.remove(i);
  }
  assert(a.getCapacity() == 109);
  assert(a.getSize() == 0);
  for (int i = 0; i < 109; ++i) {
    a.add(i);
  }
  assert(a.getCapacity() == 109);
  assert(a.getSize() == 109);
  a.add(10);
  assert(a.getCapacity() == 163);
  assert(a.getSize() == 110);
}

void TEST_6() {
  ArrayList<int> a;
  assert(a.getSize() == 0);
  assert(a.getCapacity() == 10); // 初始容量为10
  ArrayList<float> b;
  a.add(1);
  a.add(1);
  b.add(1.0);
  b.add(1.0);
  for (int i = 2; i <= 8; i++) {
    a.add(a.get(i - 2).value() + a.get(i - 1).value());
    b.add(b.get(i - 2).value() + b.get(i - 1).value());
  }
  assert(std::abs((float)a.get(8).value() - b.get(8).value()) < 1e-3);
  a.add(123);
  assert(a.getCapacity() == 10);
  a.add(123);
  assert(a.getCapacity() == 15);
  assert(a.getSize() == 11);
}

// === 部分通过标准输入输出隐藏的测试用例 ===
void TEST_7() {
  ArrayList<long long> a;
  int n, m;
  std::cin >> n;
  int x, y;
  for (int i = 0; i < n; i++) {
    std::cin >> x;
    a.add(x);
  }
  std::cout << a.getCapacity() << std::endl;
  assert(a.getSize() == n);
  std::cin >> m;
  for (int i = 0; i < m; i++) {
```

```
    std::cin >> x >> y;
    a.remove(x);
    std::cout << a.get(y).value() << std::endl;
  }
  assert(a.getSize() == n - m);
  assert(a.get(n - m) == std::nullopt);
}

void TEST_8() {
  ArrayList<std::string> a;
  ArrayList<int> b;
  int n, m, k;
  std::cin >> n >> m >> k;
  std::string s;
  for (int i = 0; i < n; i++) {
    std::cin >> s;
    a.add(s);
  }
  for (int i = 0; i < m; i++) {
    b.add(1);
    b.add(2);
  }
  for (int i = 0; i < k; i++) {
    b.remove(2);
  }
  std::string res = "";
  for (int i = 0; i < a.getSize(); i++) {
    res.append(a.get(i).value());
  }
  std::cout << res << std::endl;
  std::cout << b.getCapacity() << std::endl;
  std::cout << b.get(23).value() << std::endl;
}

void TEST_9() {
  ArrayList<float> a;
  int n, m;
  std::cin >> n >> m;
  a.add(99.99);
  std::cout << a.getCapacity() << std::endl;
  std::cout << a.getSize() << std::endl;
  for (int i = 0; i < n; i++) {
    a.add(1.0 * i);
  }
  std::cout << a.getCapacity() << std::endl;
  std::cout << a.getSize() << std::endl;
  for (int i = 0; i < m; i++) {
    a.add(1.0 * i);
  }
  std::cout << a.getCapacity() << std::endl;
  std::cout << a.getSize() << std::endl;
  ArrayList<std::string> b;
  b.add("hello");
  b.add("nihao");
  b.remove("hello");
  std::cout << b.get(0).value() << std::endl;
  assert(b.get(1) == std::nullopt);
}
```

```
void TEST_10() {
  ArrayList<int> a;
  int n, m;
  assert(a.getSize() == 0);
  assert(a.getCapacity() == 10); // 初始容量为10
  a.add(1);
  assert(a.getSize() == 1);
  assert(a.getCapacity() == 10);
  int x;
  std::cin >> n >> m;
  // 添加超出初始容量的元素，验证容量扩展
  for (int i = 0; i <= n; ++i) {
    std::cin >> x;
    a.add(x);
  }
  for (int i = 1; i <= m; i++) {
    std::cin >> x;
    if (a.get(x).has_value()) {
      std::cout << a.get(x).value() << std::endl;
    } else {
      std::cout << "Wrong!" << std::endl;
    }
  }
  a.remove(1);
  for (int i = 1; i <= m; i++) {
    std::cin >> x;
    if (a.get(x).has_value()) {
      std::cout << a.get(x).value() << std::endl;
    } else {
      std::cout << "Wrong!" << std::endl;
    }
  }
}
```

提示：在CPPOJ标准输入中输入`TEST_1`，即可运行对应的(TEST_1)单元测试。

注：单元测试中不会涉及ArrayList的拷贝和移动，也不涉及对析构函数的测试。