**数据结构第一周实验报告**



2018 软件工程创新班 邰思童 2018204493 / 2019/11/11

**实验题目**

线性表

**实验环境**

Windows 10

g++ (tdm64-2) 4.8.1

std=c++11

**实验内容**

实现一个可以动态增长的线性表，支持随机访问以及顺序查找、在任意位置插入元素等操作。

利用c++面向对象思想，将整个线性表封装成一个类，同时利用c++的提供的类模板以及函数模板特性使得线性表中可以存放任意类型的元素。

在实现随机访问功能时，利用了c++的运算符重载特性，重载 [] 运算符，使其与普通数组的使用无异。



考虑到C风格数组不进行越界检查，在实现过程中增加了这一功能，当试图越界访问时，会抛出 std::out\_of\_range 异常，便于捕获与错误调试。

**算法流程**

 动态增长部分

私有变量 capacity 记录线性表的大小， size 记录当前线性表中的元素个数，当发现 size > capacity 时，重新分配一块大小为原本容量两倍的空间，将原始数据拷贝到新的空间中，释放内存。此时线性表的长度动态增加至原始的两倍。



此方法为类的私有方法。（私有成员函数）

 随机访问部分

内部由数组实现，因此重载了 [] 运算符，直接用 a[i] 的方式访问第 i 个元素，在 i 越界的情况下会引发 std::out\_of\_range 异常



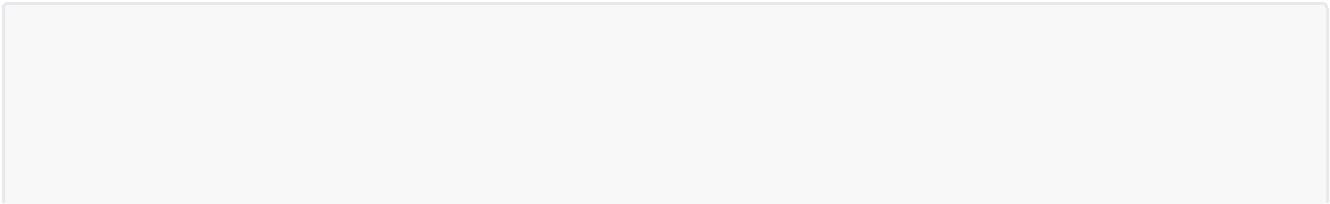
 任意位置插入部分

若空间不足则按照动态增长的规则增长至原先的两倍。

顺序寻找到需要的位置，将后面的元素按顺序后移一位，空出插入元素的位置。

时间复杂度。

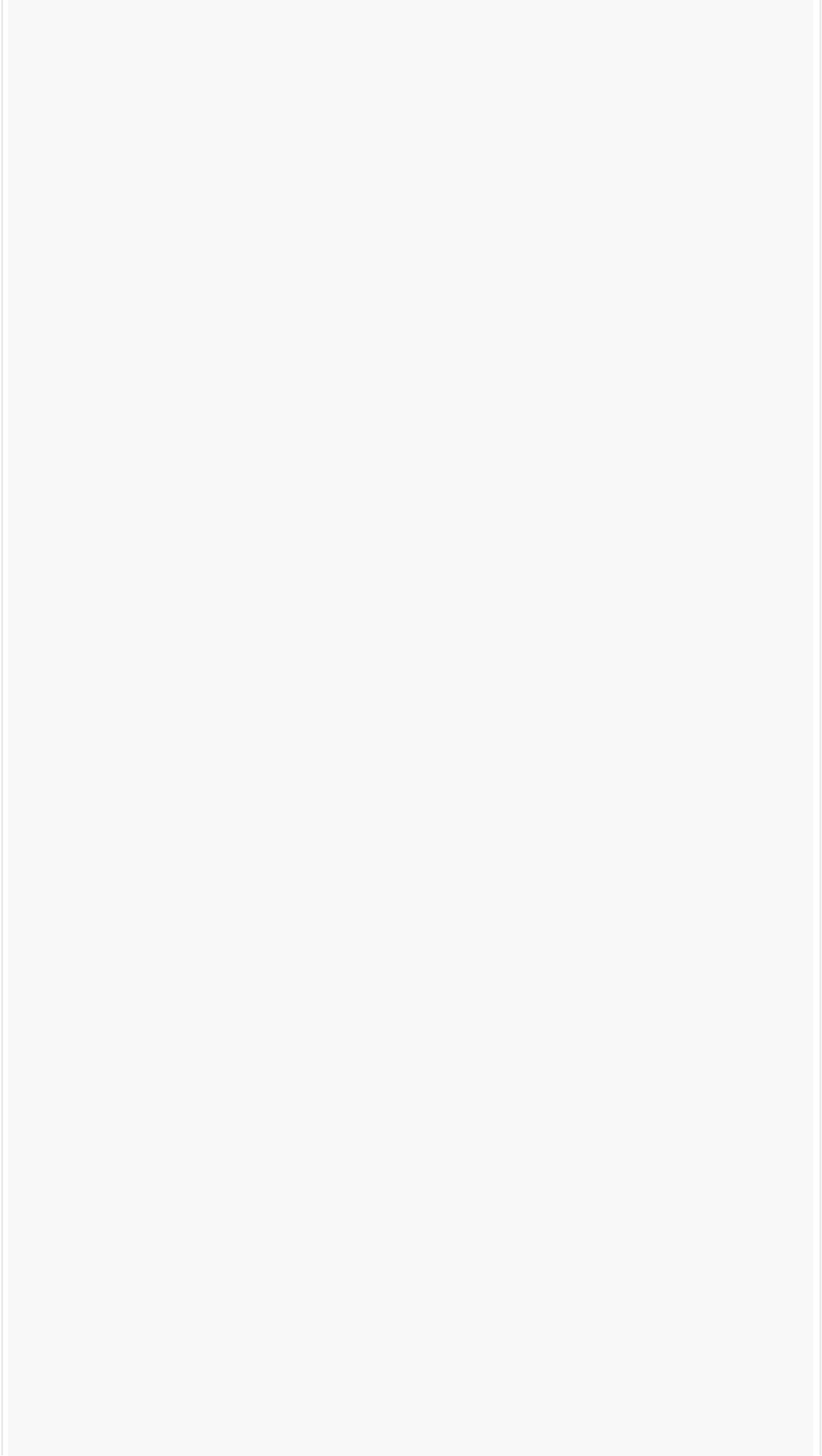
**程序代码**



//liearList.h

#ifndef LINEARLIST\_H\_

#define LINEARLIST\_H\_

#include <exception>

#include <iostream>

#include <stdexcept>

#include <algorithm>

template<typename T>

class linearList {

private:

* \*arr;

int capacity, size;

void extend();

public:

linearList(int capacity);

~linearList();

T& operator[](int index);

void insert(int index, T data);

void push\_back(T data);

void clear();

int find(T data);

int size();

};

template<typename T>

void linearList<T>::extend() {

T \*newArr = new T[capacity \* 2];

capacity \*= 2;

for (int i = 0; i <= size; i++) {

newArr[i] = arr[i];

}

delete[] arr;

arr = newArr;

}

template<typename T>

linearList<T>::linearList(int capacity) {

this->size = 0;

this->capacity = capacity;

arr = new T[capacity];

}

template<typename T>

linearList<T>::~linearList() {

delete[] arr;

}

template<typename T>

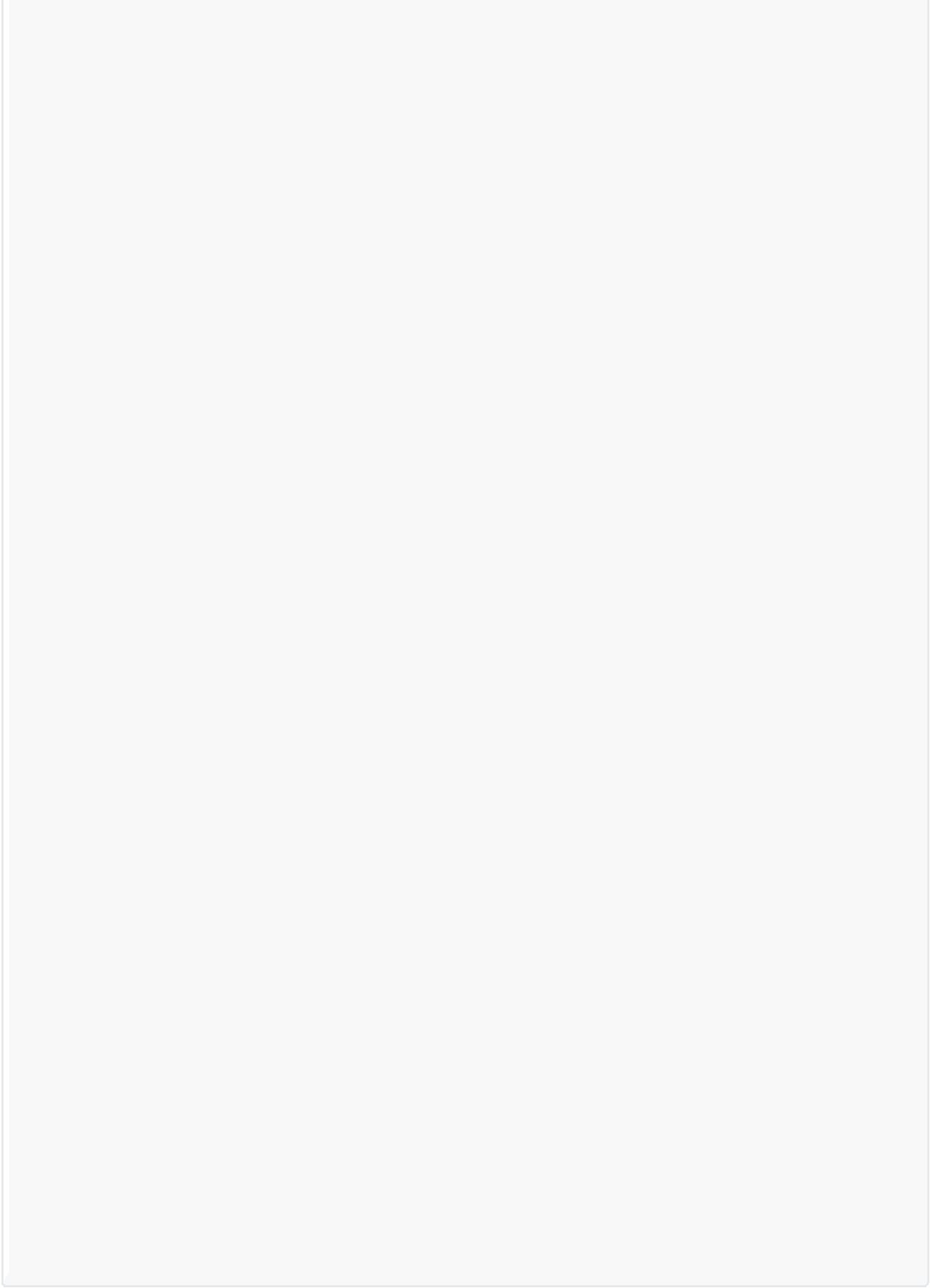
T& linearList<T>::operator[](int index) {

if (index < 0 || index >= capacity) throw std::out\_of\_range("out of range"); else {

size = std::max(size, index);

return arr[index];

}

}

template<typename T>

void linearList<T>::insert(int index, T data) { if (size >= capacity) extend();

for (int i = size + 1; i > index; i--) {

arr[i] = arr[i - 1];

}

arr[index] = data;

if (index <= size) size++;

else size = index;

}

template<typename T>

void linearList<T>::push\_back(T data) {

if (size >= capacity) extend();

arr[++size] = data;

}

template<typename T>

int linearList<T>::find(T data) {

for (int i = 0; i < size; i++) {

if(arr[i] == data) return i;

}

return -1;

}

template<typename T>

void linearList<T>::clear() {

for (int i = 0; i <= size; i++) {

arr[i] = 0;

}

size = 0;

}

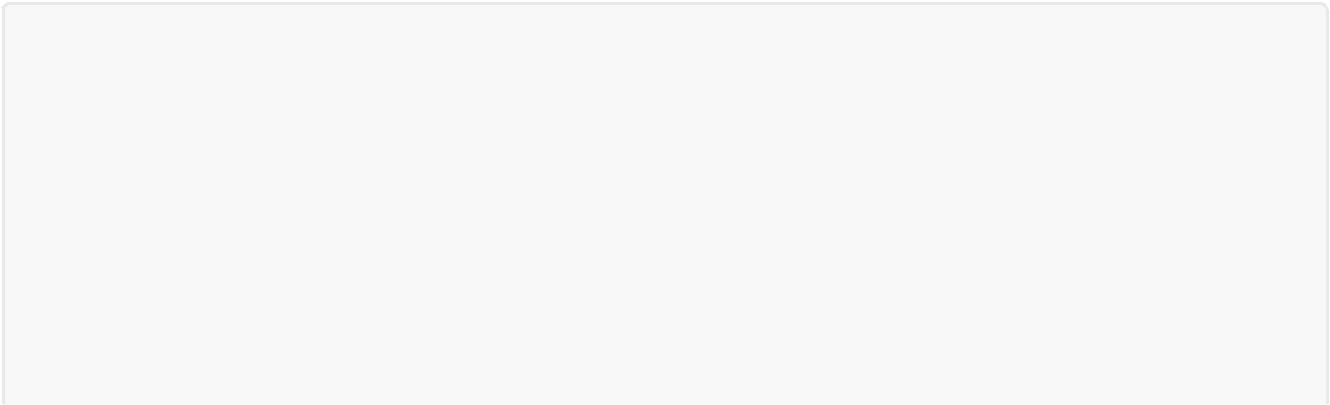
template<typename T>

int linearList<T>::size() {

return size;

}

#endif



//main.cpp

#include <iostream>

#include "linearList.h"

int main() {

linearList<int> a(10);

for (int i = 0; i < 10; i++) {

a[i] = i;

}

for (int i = 0; i < a.getSize(); i++) {

std::cout << a[i] << (i == a.getSize() - 1 ? "\n" : " ");

}

std::cout << "size = " << a.getSize() << std::endl;

try {

a[100] = 10;

} catch (std::out\_of\_range) {

std::cout << "out of range" << std::endl;

}

a.insert(2, 100);

for (int i = 0; i < a.getSize(); i++) {

std::cout << a[i] << (i == a.getSize() - 1 ? "\n" : " ");

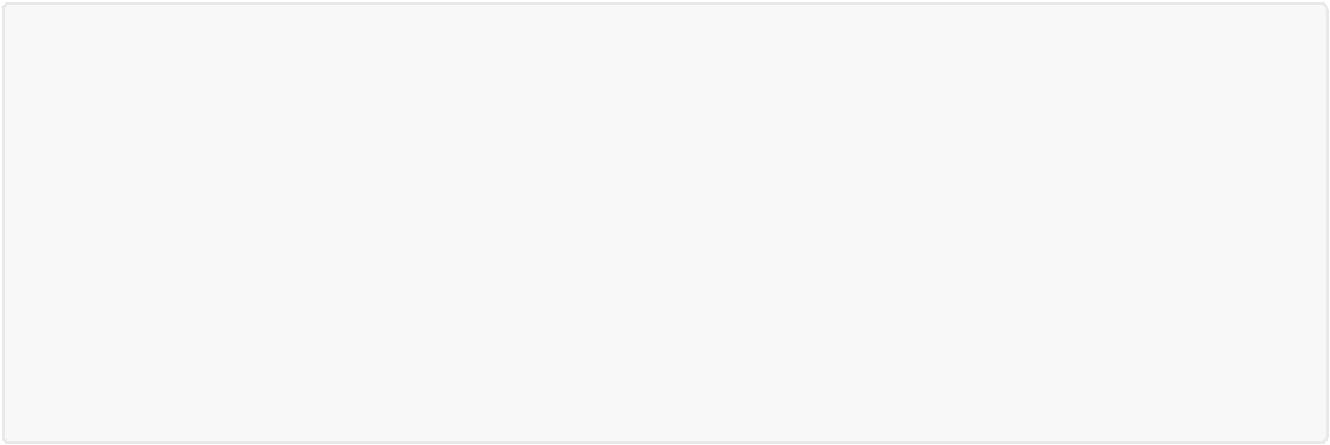
}

std::cout << "size = " << a.getSize() << std::endl;

return 0;

}

**运行结果**



E:\Code\**课程**\DataStructure

λ g++ main.cpp -o main -g -std=c++11

E:\Code\**课程**\DataStructure

λ main

0 1 2 3 4 5 6 7 8 9

size = 10

out of range

0 1 100 2 3 4 5 6 7 8 9

size = 11

**结果分析**

 第一行和第二行测试数据访问。

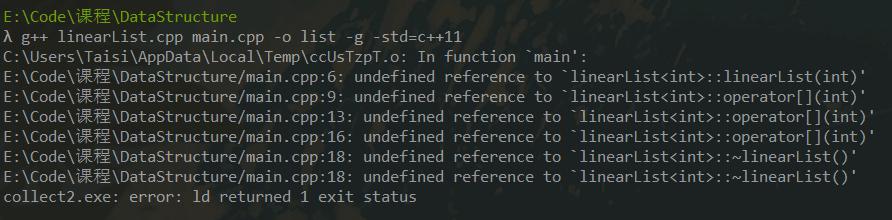
 第三行测试越界处理。

 第四行和第五行测试任意位置插入以及长度动态增长。

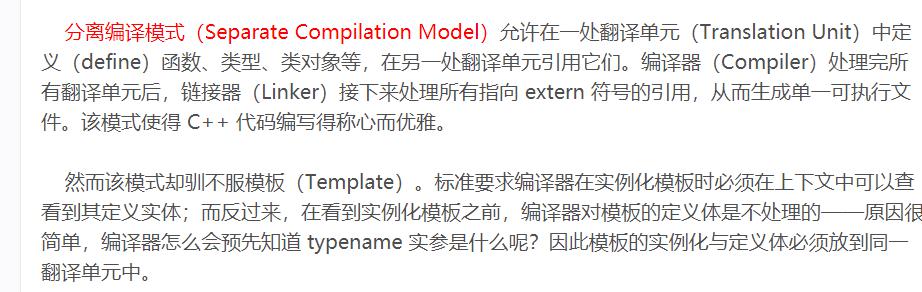
**注意事项**

 异常的使用时机？

 模板类的成员函数的定义不能与声明分开编译，否则连接器会如下报错



原因如下



 动态分配的内存记得手动释放