**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: аллокаторы

Студент: Тимофеев Алексей Владимирович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr). Опционально использование std::unique\_ptr;
2. В качестве параметра шаблона коллекция должна принимать тип данных;
3. Коллекция должна содержать метод доступа:

* Стек – pop, push, top;
* Очередь – pop, push, top;
* Список, Динамический массив – доступ к элементу по оператору [];

1. Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти – является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь);
2. Коллекция должна использовать аллокатор для выделения и освобождения памяти для своих элементов.
3. Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально – vector).
4. Реализовать программу, которая:

* Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор;
* Позволяет удалять элемент из коллекции по номеру элемента;
* Выводит на экран введенные фигуры c помощью std::for\_each;

|  |  |  |  |
| --- | --- | --- | --- |
| 14 | 5-угольник | Список | Список |

1. Описание программы

Для хранения фигуры создается шаблонный класс Pentagon , который хранит в себе радиус описанной окружности (R) и координаты центра этой окружности. Для хранения коллекции используется умный указатель. При инициализации класса list(список) выделяется память под элементы Pentagon. Также реализован класс iterator, который использует умные указатели. Для того, чтобы итератор был совместим со стандартными алгоритмами (которые мы используем в данной программе: std::for\_each, std::count\_if), необходимо перегрузить операторы - ==, !=, ++, TListItem\*, +. В классе list реализованы методы begin() и end(), которые возвращают итератор. Метод Push отвечает за вставку, в нем можно выбрать позицию на которую будет вставлен элемент. Метод Pop отвечает за удаление по индексу в списке. Выводится список с помощью перегруженного оператора << . Также я перегрузил оператор new и delete для TListItem с использованием своего аллокатор. Аллокатор выделяет блок памяти равный размеру типа моих данных умноженному на 100 и добавляет его в список блоков памяти.

1. Набор тестов

coord to center: 0 0 radius: 7 index = 0

coord to center: 50 50 radius: 8 index = 1

coord to center: -50 -50 radius: 8 index = 2

coord to center: 100 100 radius: 10 index = 1

1. Результаты выполнения тестов

dude@DESKTOP-9IO9OQQ:/mnt/d/Ycheba2kurs/OOp/OOPlab/laba6$ ./lab6

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 0 0

Enter radius: 7

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 50 50

Enter radius: 8

Enter index = 1

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: -50 -50

Enter radius: 8

Enter index = 2

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

1

Enter coord to center: 100 100

Enter radius: 10

Enter index = 1

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

3

idx: 0 (-6.77075,1.77676) (6.09803,-3.43715) (-5.0259,4.87241) (3.62457,-5.98853) (-1.98584,6.71241)

idx: 1 (90.3275,102.538) (108.711,95.0898) (92.8201,106.961) (105.178,91.445) (97.1631,109.589)

idx: 2 (42.262,52.0306) (56.9692,46.0718) (44.2561,55.5685) (54.1424,43.156) (47.7305,57.6713)

idx: 3 (-57.738,-47.9694) (-43.0308,-53.9282) (-55.7439,-44.4315) (-45.8576,-56.844) (-52.2695,-42.3287)

Choose an operation:

1) Add pentagon

2) Delete figure from list

3) Print list

4) Find the number of objects which set area less than the established one

0) Exit

0

1. Листинг программы

TListItem.h

#ifndef TLISTITEM\_H

#define TLISTITEM\_H

#include <memory>

#include <iostream>

#include "pentagon.h"

#include "TAllocationBlock.h"

template <class T>

class TListItem{

private:

std::shared\_ptr<T> item;

std::shared\_ptr<TListItem<T>> next;

std::shared\_ptr<TListItem<T>> prev;

static TAllocationBlock listitem\_allocator;

public:

TListItem(const std::shared\_ptr<T> &obj);

std::shared\_ptr<T> GetFigure() const;

std::shared\_ptr<TListItem<T>> GetNext();

std::shared\_ptr<TListItem<T>> GetPrev();

void SetNext(std::shared\_ptr<TListItem<T>> item);

void SetPrev(std::shared\_ptr<TListItem<T>> item);

template <class A> friend std::ostream& operator<<(std::ostream &os, const TListItem<A> &obj);

void \*operator new(size\_t size);

void operator delete(void \*ptr);

virtual ~TListItem(){};

};

template <class T> TAllocationBlock

TListItem<T>::listitem\_allocator(sizeof(TListItem<T>), 100);

template <class T>

TListItem<T>::TListItem(const std::shared\_ptr<T> &obj){

this->item = obj;

this->next = nullptr;

this->prev = nullptr;

}

template <class T>

std::shared\_ptr<T> TListItem<T>::GetFigure() const{

return this->item;

}

template <class T>

std::shared\_ptr<TListItem<T>> TListItem<T>::GetNext(){

return this->next;

}

template <class T>

std::shared\_ptr<TListItem<T>> TListItem<T>::GetPrev(){

return this->prev;

}

template <class T>

void TListItem<T>::SetNext(std::shared\_ptr<TListItem<T>> item){

this->next = item;

}

template <class T>

void TListItem<T>::SetPrev(std::shared\_ptr<TListItem<T>> item){

this->prev = item;

}

template <class T>

std::ostream& operator<<(std::ostream &os, const TListItem<T> &obj){

os << obj.item << std::endl;

return os;

}

template <class T>

void \*TListItem<T>::operator new(size\_t size){

return listitem\_allocator.Allocate();

}

template <class T>

void TListItem<T>::operator delete(void \*ptr){

listitem\_allocator.Deallocate(ptr);

}

#endif

TAllocationBlock.h

#ifndef TALLOCATIONBLOCK\_H

#define TALLOCATIONBLOCK\_H

#include <iostream>

#include <list>

#include <cstdlib>

typedef unsigned char Byte;

class TAllocationBlock{

private:

Byte \*\_used\_blocks;

std::list <void \*>\_free\_blocks;

public:

TAllocationBlock(int32\_t size, int32\_t count);

void \*Allocate();

void Deallocate(void \*ptr);

bool Empty();

int32\_t Size();

virtual ~TAllocationBlock();

};

#endif /\* TALLOCATIONBLOCK\_H \*/

TAllocationBlock.cpp

#include "TAllocationBlock.h"

TAllocationBlock::TAllocationBlock(int32\_t size, int32\_t count){

\_used\_blocks = (Byte \*)malloc(size \* count);

for(int32\_t i = 0; i < count; ++i) {

void \*ptr = (void \*)malloc(sizeof(void \*));

ptr = \_used\_blocks + i \* size;

\_free\_blocks.push\_back(ptr);

}

}

void \*TAllocationBlock::Allocate(){

if(!\_free\_blocks.empty()) {

return \_free\_blocks.front();

} else {

throw std::bad\_alloc();

}

}

void TAllocationBlock::Deallocate(void \*ptr){

\_free\_blocks.push\_back(ptr);

}

bool TAllocationBlock::Empty(){

return \_free\_blocks.empty();

}

int32\_t TAllocationBlock::Size(){

return \_free\_blocks.size();

}

TAllocationBlock::~TAllocationBlock(){

while(!\_free\_blocks.empty()){

\_free\_blocks.pop\_front();

}

free(\_used\_blocks);

}

1. Вывод

В данной лабораторной работе я научился реализовывать аллокаторы и добавил собственное выделение памяти в предыдущую лабораторную работу для этого я перегрузил операцию new и delete.

Список литературы

1. cppreference.com [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/container/list (дата обращения: 03.12.20)
2. Метапрограммирование в C++ [Электронный ресурс]. URL: https://habr.com/ru/post/38622/(дата обращения: 02.12.20)