МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №8 по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Клитная Анастасия Викторовна, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Цель работы:**  
Целью лабораторной работы является:   
  
Закрепление навыков по работе с памятью в C++;   
Создание аллокаторов памяти для динамических структур данных.  
  
Задание:  
  
Используя структуру данных, разработанную для лабораторной работы №5, спроектировать и разработать аллокатор памяти для динамической структуры данных.   
Цель построения аллокатора – минимизация вызова операции malloc. Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти. Аллокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 2-го уровня, согласно варианту задания). Для вызова аллокатора должны быть переопределены оператор new и delete у классов-фигур.   
  
Нельзя использовать:   
  
Стандартные контейнеры std.   
  
Программа должна позволять:   
  
Вводить произвольное количество фигур и добавлять их в контейнер;   
Распечатывать содержимое контейнера;   
Удалять фигуры из контейнера.

**Дневник отладки**

Во время выполнения лабораторной были некие трудности с реализацией аллокатора, позже они были полностью ликвидирован.

**Недочёты**  
Недочётов не было обнаружено.

**Выводы**

Лабораторная работа №8 позволила мне реализовать свой класс аллокаторов, полностью прочувствовать процесс выделения памяти на низкоуровневых языках программирования. Лабораторная прошла успешно.

**Исходный код**

HListItem.cpp

#include <iostream>

#include "HListItem.h"

template <class T> HListItem<T>::HListItem(const std::shared\_ptr<Octagon> &octagon) {

this->octagon = octagon;

this->next = nullptr;

}

template <class A> std::ostream& operator<<(std::ostream& os,HListItem<A> &obj) {

os << "[" << obj.octagon << "]" << std::endl;

return os;

}

template <class T> HListItem<T>::~HListItem() {

}

HListItem.h

#ifndef HLISTITEM\_H

#define HLISTITEM\_H

#include <iostream>

#include "octagon.h"

#include <memory>

template <class T> class HListItem {

public:

HListItem(const std::shared\_ptr<Octagon> &octagon);

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

~HListItem();

std::shared\_ptr<T> octagon;

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

};

#include "HListItem.cpp"

#endif

TAllocatorBlock.h

#ifndef TALLOCATORBLOCK\_H

#define TALLOCATORBLOCK\_H

#include "TLinkedList.h"

#include <memory>

class TAllocatorBlock {

public:

TAllocatorBlock(const size\_t& size, const size\_t count){

this->size = size;

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

unused\_blocks.Insert(malloc(size));

}

}

void\* Allocate(const size\_t& size){

if(size != this->size){

std::cout << "Error during allocation\n";

}

if(unused\_blocks.Length()){

for(int i = 0; i < 5; ++i){

unused\_blocks.Insert(malloc(size));

}

}

void\* tmp = unused\_blocks.GetItem(1);

used\_blocks.Insert(unused\_blocks.GetItem(1));

unused\_blocks.Remove(0);

return tmp;

}

void Deallocate(void\* ptr){

unused\_blocks.Insert(ptr);

}

~TAllocatorBlock(){

while(used\_blocks.size()){

try{

free(used\_blocks.GetItem(1);)

used\_blocks.Remove(0);

} catch(...){

used\_blocks.Remove(0);

}

}

while(unused\_blocks.size()){

try{

free(unused\_blocks.GetItem(1);

unused\_blocks.Remove(0);

} catch(...){

unused\_blocks.Remove(0);

}

}

}

private:

size\_t size;

TLinkedList <void\*> used\_blocks;

TLinkedList <void\*> unused\_blocks;

};

#endif

## ****TLinkedList.cpp****

#include <iostream>

#include "TLinkedList.h"

template <class T> TLinkedList<T>::TLinkedList() {

size\_of\_list = 0;

std::shared\_ptr<HListItem<T>> front;

std::shared\_ptr<HListItem<T>> back;

std::cout << "Octagon List created" << std::endl;

}

template <class T> TLinkedList<T>::TLinkedList(const std::shared\_ptr<TLinkedList> &other){

front = other->front;

back = other->back;

}

template <class T> size\_t TLinkedList<T>::Length() {

return size\_of\_list;

}

template <class T> bool TLinkedList<T>::Empty() {

return size\_of\_list;

}

template <class T> std::shared\_ptr<Octagon>& TLinkedList<T>::GetItem(size\_t idx){

int k = 0;

std::shared\_ptr<HListItem<T>> obj = front;

while (k != idx){

k++;

obj = obj->next;

}

return obj->octagon;

}

template <class T> std::shared\_ptr<Octagon>& TLinkedList<T>::First() {

return front->octagon;

}

template <class T> std::shared\_ptr<Octagon>& TLinkedList<T>::Last() {

return back->octagon;

}

template <class T> void TLinkedList<T>::InsertLast(const std::shared\_ptr<Octagon> &&octagon) {

std::shared\_ptr<HListItem<T>> obj (new HListItem<T>(octagon));

if(size\_of\_list == 0) {

front = obj;

back = obj;

size\_of\_list++;

return;

}

back->next = obj;

back = obj;

obj->next = nullptr;

size\_of\_list++;

}

template <class T> void TLinkedList<T>::RemoveLast() {

if (size\_of\_list == 0) {

std::cout << "Octagon does not pop\_back, because the Octagon List is empty" << std:: endl;

} else {

if (front == back) {

RemoveFirst();

size\_of\_list--;

return;

}

std::shared\_ptr<HListItem<T>> prev\_del = front;

while (prev\_del->next != back) {

prev\_del = prev\_del->next;

}

prev\_del->next = nullptr;

back = prev\_del;

size\_of\_list--;

}

}

template <class T> void TLinkedList<T>::InsertFirst(const std::shared\_ptr<Octagon> &&octagon) {

std::shared\_ptr<HListItem<T>> obj (new HListItem<T>(octagon));

if(size\_of\_list == 0) {

front = obj;

back = obj;

} else {

obj->next = front;

front = obj;

}

size\_of\_list++;

}

template <class T> void TLinkedList<T>::RemoveFirst() {

if (size\_of\_list == 0) {

std::cout << "Octagon does not pop\_front, because the Octagon List is empty" << std:: endl;

} else {

std::shared\_ptr<HListItem<T>> del = front;

front = del->next;

size\_of\_list--;

}

}

template <class T> void TLinkedList<T>::Insert(const std::shared\_ptr<Octagon> &&octagon, size\_t position) {

if (position <0) {

std::cout << "Position < zero" << std::endl;

} else if (position > size\_of\_list) {

std::cout << " Position > size\_of\_list" << std::endl;

} else {

std::shared\_ptr<HListItem<T>> obj (new HListItem<T>(octagon));

if (position == 0) {

front = obj;

back = obj;

} else {

int k = 0;

std::shared\_ptr<HListItem<T>> prev\_insert = front;

std::shared\_ptr<HListItem<T>> next\_insert;

while(k+1 != position) {

k++;

prev\_insert = prev\_insert->next;

}

next\_insert = prev\_insert->next;

prev\_insert->next = obj;

obj->next = next\_insert;

}

size\_of\_list++;

}

}

template <class T> void TLinkedList<T>::Remove(size\_t position) {

if (position > size\_of\_list ) {

std:: cout << "Position " << position << " > " << "size " << size\_of\_list << " Not correct erase" << std::endl;

} else if (position < 0) {

std::cout << "Position < 0" << std::endl;

} else {

if (position == 0) {

RemoveFirst();

} else {

int k = 0;

std::shared\_ptr<HListItem<T>> prev\_erase = front;

std::shared\_ptr<HListItem<T>> next\_erase;

std::shared\_ptr<HListItem<T>> del;F

while( k+1 != position) {

k++;

prev\_erase = prev\_erase->next;

}

next\_erase = prev\_erase->next;

del = prev\_erase->next;

next\_erase = del->next;

prev\_erase->next = next\_erase;

}

size\_of\_list--;

}

}

template <class T> void TLinkedList<T>::Clear() {

std::shared\_ptr<HListItem<T>> del = front;

std::shared\_ptr<HListItem<T>> prev\_del;

if(size\_of\_list !=0 ) {

while(del->next != nullptr) {

prev\_del = del;

del = del->next;

}

size\_of\_list = 0;

// std::cout << "HListItem deleted" << std::endl;

}

size\_of\_list = 0;

std::shared\_ptr<HListItem<T>> front;

std::shared\_ptr<HListItem<T>> back;

}

template <class T> std::ostream& operator<<(std::ostream& os, TLinkedList<T>& hl) {

if (hl.size\_of\_list == 0) {

os << "The octagon list is empty, so there is nothing to output" << std::endl;

} else {

os << "Print Octgon List" << std::endl;

std::shared\_ptr<HListItem<T>> obj = hl.front;

while(obj != nullptr) {

if (obj->next != nullptr) {

os << obj->octagon << " " << "," << " ";

obj = obj->next;

} else {

os << obj->octagon;

obj = obj->next;

}

}

os << std::endl;

}

return os;

}

template <class T> TLinkedList<T>::~TLinkedList() {

std::shared\_ptr<HListItem<T>> del = front;

std::shared\_ptr<HListItem<T>> prev\_del;

if(size\_of\_list !=0 ) {

while(del->next != nullptr) {

prev\_del = del;

del = del->next;

}

size\_of\_list = 0;

std::cout << "Octagon List deleted" << std::endl;

}

}

## ****TLinkedList.h****

#ifndef HLIST\_H

#define HLIST\_H

#include <iostream>

#include "HListItem.h"

#include "octagon.h"

#include <memory>

template <class T> class TLinkedList {

public:

TLinkedList();

int size\_of\_list;

size\_t Length();

std::shared\_ptr<Pentagon>& First();

std::shared\_ptr<Pentagon>& Last();

std::shared\_ptr<Pentagon>& GetItem(size\_t idx);

bool Empty();

TLinkedList(const std::shared\_ptr<TLinkedList> &other);

void InsertFirst(const std::shared\_ptr<Octagon> &&octagon);

void InsertLast(const std::shared\_ptr<Octagon> &&octagon);

void RemoveLast();

void RemoveFirst();

void Insert(const std::shared\_ptr<Octagon> &&octagon, size\_t position);

void Remove(size\_t position);

void Clear();

template <class A> friend std::ostream& operator<<(std::ostream& os, TLinkedList<A>& list);

~TLinkedList();

private:

std::shared\_ptr<HListItem<T>> front;

std::shared\_ptr<HListItem<T>> back;

};

#include "TLinkedList.cpp"

#endif

Figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <memory>

#include "point.h"

class Figure {

public:

virtual double Area() = 0;

virtual void Print(std::ostream &os) = 0;

virtual size\_t VertexesNumber() = 0;

virtual ~Figure() {};

};

#endif

Main.cpp

#include <iostream>

#include "octagon.h"

#include "TLinkedList.h"

#include "TLinkedListItem.h"

#include "TAllocatorBlock.h"

int main () {

//lab1

Octagon a (std::cin);

std:: cout << "The area of your figure is : " << a.Area() << std:: endl;

Octagon b (std::cin);

std:: cout << "The area of your figure is : " << b.Area() << std:: endl;

Octagon c (std::cin);

std:: cout << "The area of your figure is : " << c.Area() << std:: endl;

//lab8

TLinkedList list;

std:: cout << "Is list empty? " << list.Empty() << std:: endl;

list.InsertFirst(a);

std:: cout << "And now, is tree empty? " << list.Empty() << std:: endl;

list.InsertLast(b);

list.InsertLast(c);

std:: cout << "The length of your list is: " << list.Length() << std:: endl;

std:: cout << list;

return 0;

}

Octagon.cpp

#include "octagon.h"

#include <cmath>

Octagon::Octagon() {

a.X() == 0.0; a.Y() == 0.0;

b.X() == 0.0; b.Y() == 0.0;

c.X() == 0.0; c.Y() == 0.0;

d.X() == 0.0; d.Y() == 0.0;

e.X() == 0.0; e.Y() == 0.0;

f.X() == 0.0; f.Y() == 0.0;

g.X() == 0.0; g.Y() == 0.0;

h.X() == 0.0; h.Y() == 0.0;

}

Octagon::Octagon(std::istream &InputStream)

{

InputStream >> a;

InputStream >> b;

InputStream >> c;

InputStream >> d;

InputStream >> e;

InputStream >> f;

InputStream >> g;

InputStream >> h;

std:: cout << "Octagon that you wanted to create has been created" << std:: endl;

}

void Octagon::Print(std::ostream &OutputStream) {

OutputStream << "Octagon: ";

OutputStream << a << " " << b << " " << c << " " << d << " " << e <<" " << f <<" " << g <<" " << h << std:: endl;

}

size\_t Octagon::VertexesNumber() {

size\_t number = 8;

return number;

}

double Octagon:: Area() {

double q = abs(point\_a.X() \* point\_b.Y() + point\_b.X() \* point\_c.Y() + poiny\_c.X() \* point\_d.Y() + point\_d.X() \* point\_e.Y() + point\_e.X() \* point\_f.Y() + point\_f.X() \* point\_g.Y() + point\_g.X() \* point\_h.Y() + point\_h.X() \* point\_a.Y() - point\_b.X() \* point\_a.Y() - point\_c.X() \* point\_b.Y() - point\_d.X() \* point\_c.Y() - point\_e.X() \* point\_d.Y() - point\_f.X() \* point\_e.Y() - point\_g.X() \* point\_f.Y() - point\_h.X() \* point\_g.Y() - point\_a.X() \* point\_h.Y());

double s = q / 2;

return s;

}

double Octagon:: GetArea() {

return area;

}

Octagon::~Octagon() {

std:: cout << "My friend, your pentagon has been deleted" << std:: endl;

}

bool operator == (Octagon& p1, Octagon& p2){

if(p1.a == p2.a && p1.b == p2.b && p1.c == p2.c && p1.d == p2.d && p1.e == p2.e && p1.f == p2.f && p1.g == p2.g && p1.h == p2.h) {

return true;

}

return false;

}

std::ostream& operator << (std::ostream& os, Octagon& p){

os << "Octagon: ";

os << p.a << p.b << p.c << p.d << p.e << p.f << p.g << p.h;

os << std::endl;

return os;

}

Octagon.h

#ifndef OCTAGON\_H

#define OCTAGON\_H

#include "figure.h"

#include <iostream>

class Octagon : public Figure {

public:

Octagon(std::istream &InputStream);

Octagon();

double GetArea();

size\_t VertexesNumber();

double Area();

void Print(std::ostream &OutputStream);

friend bool operator == (Octagon& p1, Octagon& p2);

friend std::ostream& operator << (std::ostream& os, Octagon& p);

virtual ~Octagon();

double area;

private:

Point a;

Point b;

Point c;

Point d;

Point e;

Point f;

Point g;

Point h;

};

#endif

Point.cpp

#include "point.h"

#include <cmath>

Point::Point() : x(0.0), y(0.0) {}

Point::Point(double x, double y) : x(x), y(y) {}

Point::Point(std::istream &is) {

is >> x >> y;

}

double Point::X() {

return x;

};

double Point::Y() {

return y;

};

std::istream& operator>>(std::istream& is, Point& p) {

is >> p.x >> p.y;

return is;

}

std::ostream& operator<<(std::ostream& os, Point& p) {

os << "(" << p.x << ", " << p.y << ")";

return os;

}

bool operator == (Point &p1, Point& p2) {

return (p1.x == p2.x && p1.y == p2.y);

}

Point.h

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(std::istream &is);

Point(double x, double y);

friend bool operator == (Point& p1, Point& p2);

friend class Octagon;

double X();

double Y();

friend std::istream& operator>>(std::istream& is, Point& p);

friend std::ostream& operator<<(std::ostream& os, Point& p);

private:

double x;

double y;

};

#endif