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 *Зинин Владислав Владимирович, 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.

**Описание программы**

Исходный код лежит в 14 файлах:

1. main.cpp - основная программа, взаимодействие с пользователем посредством команд из меню

2. include/figure.h - описание абстрактного класса фигур

3. include/point.h - описание класса точки

4. include/TVector.inl - реализация функций контейнера первого уровня (в моем случае вектора)

5. include/TVector.h – реализация класса контейнера первого уровня (в моем случае вектора)

6. include/rhombus.h - описание класса ромба, наследующегося от figures

7. include/point.cpp - реализация класса точки

8. include/TVectorItem.inl – реализация функций вспомогательного класса для контейнера

9. include/TVectorItem.h – описание вспомогательного класса для контейнера

10. include/rhombus.cpp: реализация класса ромба, наследующегося от figure

11. include/titerator.h – реализация класса Iterator

12. include/TQueue.h – реализация класса контейнера второго уровня

13. include/tallocation\_block.h – описание класса аллокатора

14. include/ tallocation\_block.cpp – реализация функций класса аллокатора

**Дневник отладки**

Во время выполнения лабораторной были некие трудности с реализацией линейного списка и 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**

**figure.h**

|  |
| --- |
| #ifndef FIGURE\_H |
|  |

|  |
| --- |
| #define FIGURE\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "point.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Figure |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| virtual ~Figure(){}; |
|  |

|  |
| --- |
| virtual double Area() = 0; |
|  |

|  |
| --- |
| virtual size\_t VertexesNumber() = 0; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //FIGURE\_H

**TVector.h**

|  |
| --- |
| #ifndef TVECTOR\_H |
|  |

|  |
| --- |
| #define TVECTOR\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVectorItem.h" |
|  |

|  |
| --- |
| #include "rhombus.h" |
|  |

|  |
| --- |
| #include <memory> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> class TVector |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| /\*-----init-----\*/ |
|  |

|  |
| --- |
| TVector(); |
|  |

|  |
| --- |
| /\*-----void-----\*/ |
|  |

|  |
| --- |
| void Remove(size\_t idx); |
|  |

|  |
| --- |
| void Resize(const size\_t new\_size); |
|  |

|  |
| --- |
| void InsertLast(std::shared\_ptr<Rhombus> &&rhomb); |
|  |

|  |
| --- |
| void RemoveLast(); |
|  |

|  |
| --- |
| /\*-----Rhombus-----\*/ |
|  |

|  |
| --- |
| const Rhombus& Last(); |
|  |

|  |
| --- |
| /\*-----bool-----\*/ |
|  |

|  |
| --- |
| bool Empty(); |
|  |

|  |
| --- |
| /\*-----size\_t-----\*/ |
|  |

|  |
| --- |
| size\_t Length(); |
|  |

|  |
| --- |
| /\*----operator-----\*/ |
|  |

|  |
| --- |
| Rhombus& operator[] (const size\_t idx); |
|  |

|  |
| --- |
| template <class B> friend std::ostream& operator<<(std::ostream& os, TVector<B> &obj); |
|  |

|  |
| --- |
| /\*-----destructor-----\*/ |
|  |

|  |
| --- |
| ~TVector(); |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| size\_t size; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> first; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include "TVector.inl" |
|  |

|  |
| --- |
|  |
|  |

#endif//TVECTOR\_H

**TVector.inl**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVector.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*----init----\*/ |
|  |

|  |
| --- |
| template <class T> TVector<T>::TVector() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| size = 0; |
|  |

|  |
| --- |
| std::cout << "TVector created" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*----bool----\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> bool TVector<T>::Empty() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return size == 0?1:0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> void TVector<T>::InsertLast(std::shared\_ptr<Rhombus> &&rhomb) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> value (new TVectorItem<T>(rhomb)); |
|  |

|  |
| --- |
| if(size == 0) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| this->first = value; |
|  |

|  |
| --- |
| this->first->next = nullptr; |
|  |

|  |
| --- |
| this->first = value; |
|  |

|  |
| --- |
| size++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> end = this->first; |
|  |

|  |
| --- |
| while(end->next != nullptr) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| end = end->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| end->next = value; |
|  |

|  |
| --- |
| value->next = nullptr; |
|  |

|  |
| --- |
| size++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> void TVector<T>::Resize(const size\_t new\_size) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(size == new\_size || new\_size < 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else if(new\_size > size) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| size\_t iter = new\_size - size; |
|  |

|  |
| --- |
| for(int i = 0; i < iter; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| InsertLast(std::shared\_ptr<Rhombus>(new Rhombus())); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else{ |
|  |

|  |
| --- |
| size\_t iter = new\_size; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> end = this->first; |
|  |

|  |
| --- |
| for(int i = 0; i < iter; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| end = end->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| end->next = nullptr; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| size = new\_size; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> void TVector<T>::RemoveLast() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(size == 0) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "List is empty" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(size == 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| size--; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> del = this->first; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> del = this->first; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> save; |
|  |

|  |
| --- |
| while(del->next != nullptr) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| save = del; |
|  |

|  |
| --- |
| del = del->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| size--; |
|  |

|  |
| --- |
| save->next = nullptr; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> void TVector<T>::Remove(size\_t idx) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(idx < 1 || idx > size) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Invalid erase!" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> del; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> prev\_del; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> next\_del = this->first; |
|  |

|  |
| --- |
| size--; |
|  |

|  |
| --- |
| if(idx == 1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| del = this->first; |
|  |

|  |
| --- |
| next\_del = next\_del->next; |
|  |

|  |
| --- |
| this->first = next\_del; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| for(int i = 1; i < idx; ++i) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| prev\_del = next\_del; |
|  |

|  |
| --- |
| next\_del = next\_del->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| del = next\_del; |
|  |

|  |
| --- |
| next\_del = next\_del->next; |
|  |

|  |
| --- |
| prev\_del->next = next\_del; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*-----Rhombus-----\*/ |
|  |

|  |
| --- |
| template <class T> const Rhombus& TVector<T>::Last() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> node = this->first; |
|  |

|  |
| --- |
| while(node->next != nullptr) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| node = node->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return \*node->rhomb; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*----destructor---\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> TVector<T>::~TVector() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "TVector deleted" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*----size\_t---\*/ |
|  |

|  |
| --- |
| template <class A> size\_t TVector<A>::Length() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return size; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| /\*----operator---\*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> Rhombus& TVector<T>::operator[](const size\_t idx) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> idx\_rhomb = this->first; |
|  |

|  |
| --- |
| for(int i = 1; i < idx; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| idx\_rhomb = idx\_rhomb->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return \*idx\_rhomb->rhomb; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> std::ostream& operator<<(std::ostream& os, TVector<T>& obj) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(obj.size == 0) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << "TList is empty" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << "Print rhombus" << std::endl; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> print = obj.first; |
|  |

|  |
| --- |
| os << '['; |
|  |

|  |
| --- |
| for(int i = 0; i < obj.size - 1; i++) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << print->rhomb->Area() << " " << "," << " "; |
|  |

|  |
| --- |
| print = print->next; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| os << print->rhomb->Area() << ']'; |
|  |

|  |
| --- |
| os << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return os; |
|  |

}

**TVectorItem.h**

|  |
| --- |
| #ifndef TVECTORITEM\_H |
|  |

|  |
| --- |
| #define TVECTORITEM\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "rhombus.h" |
|  |

|  |
| --- |
| #include <memory> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> class TVectorItem |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| TVectorItem(std::shared\_ptr<Rhombus>& rhomb); |
|  |

|  |
| --- |
| template <class B> friend std::ostream& operator<<(std::ostream& os, TVectorItem<B> &obj); |
|  |

|  |
| --- |
| ~TVectorItem(); |
|  |

|  |
| --- |
| std::shared\_ptr<T> rhomb; |
|  |

|  |
| --- |
| std::shared\_ptr<TVectorItem<T>> next; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
| #include "TVectorItem.inl" |
|  |

#endif //TVECTORITEM\_H

**TVectorItem.inl**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVectorItem.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> TVectorItem<T>::TVectorItem(std::shared\_ptr<Rhombus>& rhomb) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| this->rhomb = rhomb; |
|  |

|  |
| --- |
| this->next = nullptr; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class B> std::ostream& operator<<(std::ostream& os, TVectorItem<B> &obj) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| os << obj.rhomb << " "; |
|  |

|  |
| --- |
| return os; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class T> TVectorItem<T>::~TVectorItem() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "TVectorItem deleted" << std::endl; |
|  |

}

**Main.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "TVector.h" |
|  |

|  |
| --- |
| #include <vector> |
|  |

|  |
| --- |
| #include "tallocation\_block.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int main() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TVector<Rhombus> list; |
|  |

|  |
| --- |
| /\*-----Test push\_front---\*/ |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,2), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,3), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,4), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,5), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,6), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| list.InsertLast(std::shared\_ptr<Rhombus>(new Rhombus(Point(1,7), Point(3,4), Point(5,6), Point(7,8)))); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| /\*-----Test pop\_front---\*/ |
|  |

|  |
| --- |
| list.RemoveLast(); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| list.RemoveLast(); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| /\*-----Test erase---\*/ |
|  |

|  |
| --- |
| list.Resize(2); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| std::cout << "--------------" << std::endl; |
|  |

|  |
| --- |
| std::cout << list.Length() << std::endl; |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| std::cout << list[2] << std::endl; |
|  |

|  |
| --- |
| list.Resize(4); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| list.Resize(4); |
|  |

|  |
| --- |
| std::cout << list << std::endl; |
|  |

|  |
| --- |
| for (auto i : list) { |
|  |

|  |
| --- |
| std::cout << \*i << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| TAllocationBlock allocator(sizeof(int), 10); |
|  |

|  |
| --- |
| int \*a1 = nullptr; |
|  |

|  |
| --- |
| int \*a2 = nullptr; |
|  |

|  |
| --- |
| int \*a3 = nullptr; |
|  |

|  |
| --- |
| int \*a4 = nullptr; |
|  |

|  |
| --- |
| int \*a5 = nullptr; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| a1 = (int \*) allocator.allocate(); |
|  |

|  |
| --- |
| \*a1 = 1; |
|  |

|  |
| --- |
| std::cout << "a1 pointer value:" << \*a1 << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| a2 = (int \*) allocator.allocate(); |
|  |

|  |
| --- |
| \*a2 = 2; |
|  |

|  |
| --- |
| std::cout << "a2 pointer value:" << \*a2 << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| a3 = (int \*) allocator.allocate(); |
|  |

|  |
| --- |
| \*a3 = 3; |
|  |

|  |
| --- |
| std::cout << "a3 pointer value:" << \*a3 << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| allocator.deallocate(a1); |
|  |

|  |
| --- |
| allocator.deallocate(a3); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| a4 = (int \*) allocator.allocate(); |
|  |

|  |
| --- |
| \*a4 = 4; |
|  |

|  |
| --- |
| std::cout << "a4 pointer value:" << \*a4 << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| a5 = (int \*) allocator.allocate(); |
|  |

|  |
| --- |
| \*a5 = 5; |
|  |

|  |
| --- |
| std::cout << "a5 pointer value:" << \*a5 << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::cout << "a1 pointer value:" << \*a1 << std::endl; |
|  |

|  |
| --- |
| std::cout << "a2 pointer value:" << \*a2 << std::endl; |
|  |

|  |
| --- |
| std::cout << "a3 pointer value:" << \*a3 << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| allocator.deallocate(a2); |
|  |

|  |
| --- |
| allocator.deallocate(a4); |
|  |

|  |
| --- |
| allocator.deallocate(a5); |
|  |

|  |
| --- |
| return 0; |
|  |

}

**Point.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "point.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| 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\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| 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; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double get\_x(Point &other) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return other.x\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double get\_y(Point &other) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return other.y\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void Point::set\_x(Point &other, double x) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| other.x\_ = x; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void Point::set\_y(Point &other, double y) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| other.y\_ = y; |
|  |

}

**Point.h**

|  |
| --- |
| #ifndef POINT\_H |
|  |

|  |
| --- |
| #define POINT\_H |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Point |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| Point(); |
|  |

|  |
| --- |
| Point(double x, double y); |
|  |

|  |
| --- |
| Point(std::istream &is); |
|  |

|  |
| --- |
| double dist(Point &other); |
|  |

|  |
| --- |
| friend double get\_x(Point &other); |
|  |

|  |
| --- |
| friend double get\_y(Point &other); |
|  |

|  |
| --- |
| void set\_x(Point &other, double x); |
|  |

|  |
| --- |
| void set\_y(Point &other, double y); |
|  |

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

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| double x\_, y\_; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //POINT\_H

**Rhombus.cpp**

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "rhombus.h" |
|  |

|  |
| --- |
| #include <math.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::Rhombus() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| a.set\_x(a, 1); |
|  |

|  |
| --- |
| a.set\_y(a, 1); |
|  |

|  |
| --- |
| b.set\_x(b, 2); |
|  |

|  |
| --- |
| b.set\_y(b, 2); |
|  |

|  |
| --- |
| c.set\_x(c, 0); |
|  |

|  |
| --- |
| c.set\_y(c, 3); |
|  |

|  |
| --- |
| d.set\_x(d, -1); |
|  |

|  |
| --- |
| d.set\_y(d, -1); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::Rhombus(std::istream &is) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| is >> a; |
|  |

|  |
| --- |
| is >> b; |
|  |

|  |
| --- |
| is >> c; |
|  |

|  |
| --- |
| is >> d; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::Rhombus(Point pa, Point pb, Point pc, Point pd): a(pa), b(pb), c(pc), d(pd) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Rhombus created" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // void Rhombus::Print(std::ostream &os) |
|  |

|  |
| --- |
| // { |
|  |

|  |
| --- |
| // os << "Rhombus" << std::endl; |
|  |

|  |
| --- |
| // os << a << ',' << b << ',' << c << ',' << d << std::endl; |
|  |

|  |
| --- |
| // } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| double Rhombus::Area() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return 0.5 \* fabs(get\_x(a)\*get\_y(b) + get\_x(b)\*get\_y(c) + get\_x(c)\*get\_y(d) + get\_x(d)\*get\_y(a) - get\_x(b)\*get\_y(a) - get\_x(c)\*get\_y(b) - get\_x(d)\*get\_y(c) - get\_x(a)\*get\_y(d)); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Rhombus::~Rhombus() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Rhombus deleted" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| size\_t Rhombus::VertexesNumber() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| return 4; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::ostream& operator<<(std::ostream& os, Rhombus& p) { |
|  |

|  |
| --- |
| os << p.a << p.b << p.c << p.d; |
|  |

|  |
| --- |
| return os; |
|  |

}

**Rhombus.h**

|  |
| --- |
| #ifndef RHOMBUX\_H |
|  |

|  |
| --- |
| #define RHOMBUX\_H |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include "point.h" |
|  |

|  |
| --- |
| #include "figure.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Rhombus : public Figure |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| Rhombus(); |
|  |

|  |
| --- |
| Rhombus(std::istream &is); |
|  |

|  |
| --- |
| Rhombus(Point a, Point b, Point c, Point d); |
|  |

|  |
| --- |
| double Area(); |
|  |

|  |
| --- |
| size\_t VertexesNumber(); |
|  |

|  |
| --- |
| friend std::ostream& operator<<(std::ostream& os, Rhombus& p); |
|  |

|  |
| --- |
| virtual ~Rhombus(); |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| Point a, b, c, d; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif //RHOMBUX\_H

**Titerot.h**

|  |
| --- |
| #ifndef TITERATOR\_H |
|  |

|  |
| --- |
| #define TITERATOR\_H |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <memory> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template <class node, class T> |
|  |

|  |
| --- |
| class TIterator { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| TIterator(std::shared\_ptr<node> n) { node\_ptr = n; } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::shared\_ptr<T> operator\*() { return node\_ptr->rhomb; } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| std::shared\_ptr<T> operator->() { return node\_ptr->next; } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void operator++() { node\_ptr = node\_ptr->next; } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TIterator operator++(int) { |
|  |

|  |
| --- |
| TIterator iter(\*this); |
|  |

|  |
| --- |
| ++(\*this); |
|  |

|  |
| --- |
| return iter; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bool operator==(TIterator const& i) { return node\_ptr == i.node\_ptr; } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bool operator!=(TIterator const& i) { return !(\*this == i); } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| std::shared\_ptr<node> node\_ptr; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif // TITERATOR\_H

**TQueue.h**

|  |
| --- |
| #ifndef DATA\_TQUEUE\_H |
|  |

|  |
| --- |
| #define DATA\_TQUEUE\_H |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template<typename T> |
|  |

|  |
| --- |
| class TQueue { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| TQueue() { |
|  |

|  |
| --- |
| arr\_ = new T[1]; |
|  |

|  |
| --- |
| capacity\_ = 1; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TQueue(TQueue &other) { |
|  |

|  |
| --- |
| if (this != &other) { |
|  |

|  |
| --- |
| delete[] arr\_; |
|  |

|  |
| --- |
| arr\_ = other.arr\_; |
|  |

|  |
| --- |
| size\_ = other.size\_; |
|  |

|  |
| --- |
| capacity\_ = other.capacity\_; |
|  |

|  |
| --- |
| other.arr\_ = nullptr; |
|  |

|  |
| --- |
| other.size\_ = other.capacity\_ = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TQueue(TQueue &&other) noexcept { |
|  |

|  |
| --- |
| if (this != &other) { |
|  |

|  |
| --- |
| delete[] arr\_; |
|  |

|  |
| --- |
| arr\_ = other.arr\_; |
|  |

|  |
| --- |
| size\_ = other.size\_; |
|  |

|  |
| --- |
| capacity\_ = other.capacity\_; |
|  |

|  |
| --- |
| other.arr\_ = nullptr; |
|  |

|  |
| --- |
| other.size\_ = other.capacity\_ = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TQueue &operator=(TQueue &other) { |
|  |

|  |
| --- |
| if (this != &other) { |
|  |

|  |
| --- |
| delete[] arr\_; |
|  |

|  |
| --- |
| arr\_ = other.arr\_; |
|  |

|  |
| --- |
| size\_ = other.size\_; |
|  |

|  |
| --- |
| capacity\_ = other.capacity\_; |
|  |

|  |
| --- |
| other.arr\_ = nullptr; |
|  |

|  |
| --- |
| other.size\_ = other.capacity\_ = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return \*this; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TQueue &operator=(TQueue &&other) noexcept { |
|  |

|  |
| --- |
| if (this != &other) { |
|  |

|  |
| --- |
| delete[] arr\_; |
|  |

|  |
| --- |
| arr\_ = other.arr\_; |
|  |

|  |
| --- |
| size\_ = other.size\_; |
|  |

|  |
| --- |
| capacity\_ = other.capacity\_; |
|  |

|  |
| --- |
| other.arr\_ = nullptr; |
|  |

|  |
| --- |
| other.size\_ = other.capacity\_ = 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return \*this; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ~TQueue() { |
|  |

|  |
| --- |
| delete[] arr\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| [[nodiscard]] bool isEmpty() const { |
|  |

|  |
| --- |
| return size\_ == 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| [[nodiscard]] size\_t size() const { |
|  |

|  |
| --- |
| return size\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| [[nodiscard]] size\_t capacity() const { |
|  |

|  |
| --- |
| return capacity\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void push\_back(const T &value) { |
|  |

|  |
| --- |
| if (size\_ >= capacity\_) addMemory(); |
|  |

|  |
| --- |
| arr\_[size\_++] = value; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void pop() { |
|  |

|  |
| --- |
| --size\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| T &back() { |
|  |

|  |
| --- |
| return arr\_[size\_ - 1]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void remove(size\_t index) { |
|  |

|  |
| --- |
| for (size\_t i = index + 1; i < size\_; ++i) { |
|  |

|  |
| --- |
| arr\_[i - 1] = arr\_[i]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| --size\_; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| T \*begin() { |
|  |

|  |
| --- |
| return &arr\_[0]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| const T \*begin() const { |
|  |

|  |
| --- |
| return &arr\_[0]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| T \*end() { |
|  |

|  |
| --- |
| return &arr\_[size\_]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| const T \*end() const { |
|  |

|  |
| --- |
| return &arr\_[size\_]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| T &operator[](size\_t index) { |
|  |

|  |
| --- |
| return arr\_[index]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| const T &operator[](size\_t index) const { |
|  |

|  |
| --- |
| return arr\_[index]; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void addMemory() { |
|  |

|  |
| --- |
| capacity\_ \*= 2; |
|  |

|  |
| --- |
| T \*tmp = arr\_; |
|  |

|  |
| --- |
| arr\_ = new T[capacity\_]; |
|  |

|  |
| --- |
| for (size\_t i = 0; i < size\_; ++i) arr\_[i] = tmp[i]; |
|  |

|  |
| --- |
| delete[] tmp; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| T \*arr\_; |
|  |

|  |
| --- |
| size\_t size\_{}; |
|  |

|  |
| --- |
| size\_t capacity\_{}; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| template<typename T> |
|  |

|  |
| --- |
| inline std::ostream &operator<<(std::ostream &os, const TQueue<T> &vec) { |
|  |

|  |
| --- |
| for (const T &val: vec) os << val << " "; |
|  |

|  |
| --- |
| return os; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

#endif

**Tallocation\_bloch.h**

|  |
| --- |
| #ifndef TALLOCATION\_BLOCK\_H |
|  |

|  |
| --- |
| #define TALLOCATION\_BLOCK\_H |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include "TQueue.h" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class TAllocationBlock { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| TAllocationBlock(size\_t size, size\_t count); |
|  |

|  |
| --- |
| void\* allocate(); |
|  |

|  |
| --- |
| void deallocate(void\* pointer); |
|  |

|  |
| --- |
| bool has\_free\_blocks(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| virtual ~TAllocationBlock(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| private: |
|  |

|  |
| --- |
| size\_t \_size; |
|  |

|  |
| --- |
| size\_t \_count; |
|  |

|  |
| --- |
| char\* \_used\_blocks; |
|  |

|  |
| --- |
| TQueue<void\*> vec\_free\_blocks; |
|  |

|  |
| --- |
| size\_t \_free\_count; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

#endif // TALLOCATION\_BLOCK\_H

**Tallocation\_bloch.cpp**

|  |
| --- |
| #include "tallocation\_block.h" |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TAllocationBlock::TAllocationBlock(size\_t size, size\_t count) |
|  |

|  |
| --- |
| : \_size(size), \_count(count) { |
|  |

|  |
| --- |
| \_used\_blocks = (char \*) malloc(\_size \* \_count); |
|  |

|  |
| --- |
| for (size\_t i = 0; i < \_count; ++i) { |
|  |

|  |
| --- |
| vec\_free\_blocks.push\_back(\_used\_blocks + i \* \_size); |
|  |

|  |
| --- |
| std::cout << i << " OK" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| \_free\_count = \_count; |
|  |

|  |
| --- |
| std::cout << "TAllocationBlock: Memory init" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void \*TAllocationBlock::allocate() { |
|  |

|  |
| --- |
| void \*result = nullptr; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (\_free\_count > 0) { |
|  |

|  |
| --- |
| std::cout << vec\_free\_blocks.size() << std::endl; |
|  |

|  |
| --- |
| result = vec\_free\_blocks.back(); |
|  |

|  |
| --- |
| vec\_free\_blocks.pop(); |
|  |

|  |
| --- |
| \_free\_count--; |
|  |

|  |
| --- |
| std::cout << "TAllocationBlock: Allocate " << (\_count - \_free\_count); |
|  |

|  |
| --- |
| std::cout << " of " << \_count << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| std::cout << "TAllocationBlock: No memory exception :-)" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return result; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void TAllocationBlock::deallocate(void \*pointer) { |
|  |

|  |
| --- |
| std::cout << "TAllocationBlock: Deallocate block " << std::endl; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| vec\_free\_blocks[\_free\_count] = pointer; |
|  |

|  |
| --- |
| \_free\_count++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bool TAllocationBlock::has\_free\_blocks() { |
|  |

|  |
| --- |
| return \_free\_count > 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| TAllocationBlock::~TAllocationBlock() { |
|  |

|  |
| --- |
| if (\_free\_count < \_count) { |
|  |

|  |
| --- |
| std::cout << "TAllocationBlock: Memory leak?" << std::endl; |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| std::cout << "TAllocationBlock: Memory freed" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| delete \_used\_blocks; |
|  |

}