МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №8

по курсу “Объектно-ориентированное программирование» 1 семестр,

2021/22 уч. год

Студент: *Колпакова Диана Саргаевна, группа М8О-208Б-20*
Преподаватель: *Дорохов Евгений Павлович*

Задание

1. Используя структуру данных, разработанную для лабораторной работы No5, 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.

   Алокатор 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.

   Вариант 9:

Фигура №1	**Имя класса**	**Фигура №2**	**Имя класса**	**Фигура №3**	**Имя класса**
Треугольник	Triangle	Квадрат	Square	Прямоугольник	Rectangle

Описание программы

Исходный код лежит в 14 файлах:

1. main.cpp: часть программы, отвечающая за взаимодействие с пользователем через консоль. В ней происходит инициализация объектов и вызов функций работы с ними, заполнение стандартного контейнера вектор введенными объектами и печать его содержимого;
2. point.h: описание класса Point точек A(a1, a2);
3. point.cpp: реализация класса Point;

4. figure.h: описание абстрактного класса-родителя Figure;
5. figure.cpp: реализация класса Figure;
6. triangle.h: описание класса Triangle треугольников, заданных по трем точкам, наследника Figure;
7. triangle.cpp: реализация класса Triangle;
8. tlinkedlist.h
9. tlinkedlist.cpp
10. tlinkedlistitem.h
11. tlinkedlistitem.cpp
12. titterator.h
13. titterator.cpp
14. TVector.h
15. TVector.cpp
16. CMakeLists.txt

Также используется файл CMakeLists.txt с конфигурацией CMake для автоматизации сборки программы.

Дневник отладки

Проблем не возникло

Вывод

Выполняя лабораторную работу, я начала разбираться в устройстве алокаторов и работе с блоками памяти в C++. Хотя и в начале использовать аллокаторы было непривычно, в этой лабораторной мне удалось поработать с памятью напрямую. И мой аллокатор помог избежать лишних системных вызовов для выделения небольшего блока памяти.

Исходный код

main.cpp:

```
// OOP, Lab 3 variant 9, Diana Kolpakova
// Triangle, TLinkedList, shared_ptr
```

```cpp
#include <iostream>

#include "figure.h"
#include "triangle.h"
#include "tlinkedlist.h"

using namespace std;

int main()
{
    cout.setf(ios_base::boolalpha);
    cout << "oop_exercise_3 (c) Diana Kolpakova" << endl;
    cout << "Triangles, TLinkedList, shared_ptr" << endl;

    shared_ptr<TLinkedList> pList(new TLinkedList());

    for (;;)
    {
        cout << endl;
        cout << "Select an action for the linked list of triangles" << endl;
        cout << "1) Is the list empty?" << endl;
        cout << "2) Get number of triangles in the list" << endl;
        cout << "3) Show the first triangle from the list" << endl;
        cout << "4) Show the last triangle from the list" << endl;
        cout << "5) Show the triangle at a specified position in the list" <<
endl;
        cout << "6) Show areas of all triangles in the list" << endl;
        cout << "7) Add a new triangle to the beginning of the list" << endl;
        cout << "8) Add a new triangle to the end of the list" << endl;
        cout << "9) Add a new triangle to a specified position in the list" <<
endl;
        cout << "a) Remove the first triangle from the list" << endl;
        cout << "b) Remove the last triangle from the list" << endl;
        cout << "c) Remove the triangle at a specified position in the list" <<
endl;
        cout << "d) Remove all triangles from the list" << endl;
        cout << "l) Show all triangles from the list" << endl;
        cout << "x) End the program" << endl;

        try
        {
            shared_ptr<Triangle> pTriangle;
            size_t position;

            char ch;
            cin >> ch;
            switch (ch)
            {
                case '1':
                    cout << "Is the list empty: " << pList->Empty() << endl;
                    break;
                case '2':
                    cout << "Length of the list: " << pList->Length() << endl;
                    break;
                case '3':
                    pTriangle = pList->First();
```

```
        cout << *pTriangle << endl;
        break;
    case '4':
        pTriangle = pList->Last();
        cout << *pTriangle << endl;
        break;
    case '5':
        cout << "Enter position in the list:";
        cin >> position;
        pTriangle = pList->GetItem(position);
        cout << *pTriangle << endl;
        break;
    case '6':
        cout << "Triangle areas:" << endl;
        if (pList->Empty())
        {
            cout << "Empty list" << endl;
        }
        else
        {
            cout << *pList << endl;
        }
        break;
    case '7':
        pTriangle = shared_ptr<Triangle>(new Triangle());
        cout << "Enter 3 points of triangle (6 numbers):";
        cin >> *(pTriangle);
        pList->InsertFirst(pTriangle);
        cout << *pTriangle << endl;
        break;
    case '8':
        pTriangle = shared_ptr<Triangle>(new Triangle());
        cout << "Enter 3 points of triangle (6 numbers):";
        cin >> *(pTriangle);
        pList->InsertLast(pTriangle);
        cout << *pTriangle << endl;
        break;
    case '9':
        cout << "Enter 3 points of triangle (6 numbers):";
        pTriangle = shared_ptr<Triangle>(new Triangle());
        cin >> *(pTriangle);
        cout << "Enter position in the list:";
        cin >> position;
        pList->Insert(pTriangle, position);
        cout << *pTriangle << endl;
        break;
    case 'a':
    case 'A':
        pList->RemoveFirst();
        cout << "Removed the first triangle" << endl;
        break;
    case 'b':
    case 'B':
        pList->RemoveLast();
        cout << "Removed the last triangle" << endl;
        break;
    case 'c':
```

```
                case 'C':
                    cout << "Enter position in the list:";
                    cin >> position;
                    pList->Remove(position);
                    cout << "Removed the triangle at specified position" <<
endl;
                    break;
                case 'd':
                case 'D':
                    pList->Clear();
                    cout << "Removed all" << endl;
                    break;
                case 'l':
                case 'L':
                    cout << "Triangles:" << endl;
                    if (pList->Empty())
                    {
                        cout << "Empty list" << endl;
                    }
                    else
                    {
                        for (size_t i = 0; i < pList->Length(); i++)
                        {
                            pTriangle = pList->GetItem(i);
                            cout << "#" << i << " " << *pTriangle << endl;
                        }
                    }
                    break;
                case 'q':
                case 'Q':
                case 'x':
                case 'X':
                    cout << "Exiting" << endl;
                    return 0;
                default:
                    cout << "Error: invalid action selected" << endl;
                    break;
            }
        }
        catch (exception& ex)
        {
            cout << "Exception: " << ex.what() << endl;
        }
    }
}
```

point.h:

```
#pragma once
#include <iostream>

using namespace std;

class Point
{
private:
   double x;
```

```
   double y;

public:
   Point();
   Point(double x, double y);

   static double Distance (const Point& point1, const Point& point2);

   friend istream& operator>>(istream& is, Point& point);
   friend ostream& operator<<(ostream& os, Point& point);

   bool operator==(const Point& other);
};
```

point.cpp:

```
#include <cmath>
#include "point.h"

using namespace std;

Point::Point()
{
   this->x = 0.0;
   this->y = 0.0;
}

Point::Point(double x, double y)
{
   this->x = x;
   this->y = y;
}

double Point::Distance(const Point& point1, const Point& point2)
{
   double dx = point1.x - point2.y;
   double dy = point1.y - point2.y;
   double distance = sqrt(dx * dx + dy * dy);
   return distance;
}

bool Point::operator==(const Point& other)
{
   return (this->x == other.x)
      && (this->y == other.y);
}

istream& operator>>(istream& is, Point& point)
{
   is >> point.x >> point.y;
   return is;
}

ostream& operator<<(ostream& os, Point& point)
{
   os << "(" << point.x << ", " << point.y << ")";
```

```
    return os;
}
```

figure.h:

```
#pragma once
#include "point.h"

class Figure
{
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
};
```

triangle.h:

```
#pragma once
#include "figure.h"

class Triangle : public Figure
{
private:
    Point point1;
    Point point2;
    Point point3;

public:
    Triangle();
    Triangle(Point point1, Point point2, Point point3);
    Triangle(const Triangle& other);

    virtual size_t VertexesNumber() override;
    virtual double Area() override;

    friend istream& operator>>(istream& is, Triangle& triangle);
    friend ostream& operator<<(ostream& os, Triangle& triangle);

    Triangle& operator=(const Triangle& other);
    bool operator==(const Triangle& other);
};
```

triangle.cpp:

```
#include "triangle.h"

using namespace std;

Triangle::Triangle()
{
    this->point1 = Point();
    this->point2 = Point();
    this->point3 = Point();
}

Triangle::Triangle(Point point1, Point point2, Point point3)
```

```
{
    this->point1 = point1;
    this->point2 = point2;
    this->point3 = point3;
}

Triangle::Triangle(const Triangle& other)
{
    this->point1 = other.point1;
    this->point2 = other.point2;
    this->point3 = other.point3;
}

size_t Triangle::VertexesNumber()
{
    return 3;
}

double Triangle::Area()
{
    double length12 = Point::Distance(point1, point2);
    double length23 = Point::Distance(point2, point3);
    double length31 = Point::Distance(point3, point1);
    double semiPerimeter = (length12 + length23 + length31) / 2.0;
    return sqrt(semiPerimeter * (semiPerimeter - length12) * (semiPerimeter -
length23) * (semiPerimeter - length31));
}

istream& operator>>(istream& is, Triangle& triangle)
{
    is >> triangle.point1 >> triangle.point2 >> triangle.point3;
    return is;
}

ostream& operator<<(ostream& os, Triangle& triangle)
{
    os << "Triangle: " << triangle.point1 << ", " << triangle.point2 << ", " <<
triangle.point3;
    return os;
}

Triangle& Triangle::operator=(const Triangle& other)
{
    this->point1 = other.point1;
    this->point2 = other.point2;
    this->point3 = other.point3;
    return *this;
}

bool Triangle::operator==(const Triangle& other)
{
    return (this->point1 == other.point1)
        && (this->point2 == other.point2)
        && (this->point3 == other.point3);
}
```

tlinkedlist.h:

```
#pragma once
#include "triangle.h"

class TLinkedList
{
private:
   struct Item
   {
      shared_ptr<Triangle> pTriangle;
      shared_ptr<Item> pNextItem;
   };

   size_t length;
   shared_ptr<Item> pFirstItem;
   shared_ptr<Item> pLastItem;

public:
   TLinkedList();
   TLinkedList(const TLinkedList& other);
   virtual ~TLinkedList();

   shared_ptr<Triangle> First();
   shared_ptr<Triangle> Last();
   shared_ptr<Triangle> GetItem(size_t position);

   void InsertFirst(shared_ptr<Triangle> pTriangle);
   void InsertLast(shared_ptr<Triangle> pTriangle);
   void Insert(shared_ptr<Triangle> pTriangle, size_t position);

   void RemoveFirst();
   void RemoveLast();
   void Remove(size_t position);

   void Clear();
   bool Empty();
   size_t Length();

   friend std::ostream& operator<<(std::ostream& os, const TLinkedList& list);
};
```

tlinkedlist.cpp:

```
#include "tlinkedlist.h"

TLinkedList::TLinkedList()
{
   pFirstItem = nullptr;
   pLastItem = nullptr;
   length = 0;
}

TLinkedList::TLinkedList(const TLinkedList& other)
{
   pFirstItem = nullptr;
   pLastItem = nullptr;
   length = 0;

   shared_ptr<Item> pCurrentItem = other.pFirstItem;
```

```
    while (pCurrentItem != nullptr)
    {
        InsertLast(pCurrentItem->pTriangle);
        pCurrentItem = pCurrentItem->pNextItem;
    }
}

shared_ptr<Triangle> TLinkedList::First()
{
    if (Empty())
        throw runtime_error("Cannon get the item from empty list");
    return pFirstItem->pTriangle;
}

shared_ptr<Triangle> TLinkedList::Last()
{
    if (Empty())
        throw runtime_error("Cannon get the item from empty list");
    return pLastItem->pTriangle;
}

void TLinkedList::InsertFirst(shared_ptr<Triangle> pTriangle)
{
    shared_ptr<Item> pNewItem(new Item());
    pNewItem->pTriangle = pTriangle;
    pNewItem->pNextItem = pFirstItem;

    pFirstItem = pNewItem;
    if (Empty())
        pLastItem = pNewItem;

    length++;
}

void TLinkedList::InsertLast(shared_ptr<Triangle> pTriangle)
{
    shared_ptr<Item> pNewItem(new Item());
    pNewItem->pTriangle = pTriangle;
    pNewItem->pNextItem = nullptr;

    if (pLastItem != nullptr)
        pLastItem->pNextItem = pNewItem;
    pLastItem = pNewItem;
    if (Empty())
        pFirstItem = pNewItem;

    length++;
}

void TLinkedList::Insert(shared_ptr<Triangle> pTriangle, size_t position)
{
    if (position == 0)
    {
        InsertFirst(pTriangle);
        return;
    }
    else if (position == length)
```

```
    {
        InsertLast(pTriangle);
        return;
    }
    else if (position > length)
        throw runtime_error("Specified poition is out of range");

        int i = 0;
    shared_ptr<Item> pCurrentItem = pFirstItem;
    shared_ptr<Item> pPreviousItem = nullptr;
    while (pCurrentItem != nullptr)
    {
        if (i == position)
            break;
        pPreviousItem = pCurrentItem;
        pCurrentItem = pCurrentItem->pNextItem;
        i++;
    }

    shared_ptr<Item> pNewItem(new Item());
    pNewItem->pTriangle = pTriangle;
    pNewItem->pNextItem = pCurrentItem;

    pPreviousItem->pNextItem = pNewItem;

    length++;
}

void TLinkedList::RemoveFirst()
{
    if (Empty())
        throw runtime_error("Cannon remove the item from empty list");
    shared_ptr<Item> pNextItem = pFirstItem->pNextItem;
    pFirstItem = pNextItem;
    length--;
    if (Empty())
        pLastItem = nullptr;
}

void TLinkedList::RemoveLast()
{
    if (Empty())
        throw runtime_error("Cannon remove the item from empty list");

    shared_ptr<Item> pCurrentItem = pFirstItem;
    shared_ptr<Item> pPreviousItem = nullptr;
    while (pCurrentItem != nullptr)
    {
        if (pCurrentItem == pLastItem)
            break;
        pPreviousItem = pCurrentItem;
        pCurrentItem = pCurrentItem->pNextItem;
    }

    if (pPreviousItem != nullptr)
        pPreviousItem->pNextItem = nullptr;
    pLastItem = pPreviousItem;
```

```
    length--;
    if (Empty())
        pFirstItem = nullptr;
}

void TLinkedList::Remove(size_t position)
{
    if (Empty())
        throw runtime_error("Cannon remove the item from empty list");
    if (position == 0)
    {
        RemoveFirst();
        return;
    }
    else if (position == length - 1)
    {
        RemoveLast();
        return;
    }
    else if (position >= length)
        throw runtime_error("Specified poition is out of range");

    int i = 0;
    shared_ptr<Item> pCurrentItem = pFirstItem;
    shared_ptr<Item> pPreviousItem = nullptr;
    while (pCurrentItem != nullptr)
    {
        if (i == position)
            break;
        pPreviousItem = pCurrentItem;
        pCurrentItem = pCurrentItem->pNextItem;
        i++;
    }

    pPreviousItem->pNextItem = pCurrentItem->pNextItem;
    length--;
}

shared_ptr<Triangle> TLinkedList::GetItem(size_t position)
{
    if (Empty())
        throw runtime_error("Cannon get the item from empty list");
    if (position >= length)
        throw runtime_error("Specified position is out of range");

    int i = 0;
    shared_ptr<Item> pCurrentItem = pFirstItem;
    while (pCurrentItem != nullptr)
    {
        if (i == position)
            return pCurrentItem->pTriangle;
        pCurrentItem = pCurrentItem->pNextItem;
        i++;
    }

    throw runtime_error("Something went wrong");
}
```

```
bool TLinkedList::Empty()
{
   return length == 0;
}

size_t TLinkedList::Length()
{
   return length;
}

void TLinkedList::Clear()
{
   shared_ptr<Item> pCurrentItem = pFirstItem;
   while (pCurrentItem != nullptr)
   {
      shared_ptr<Item> pNextItem = pCurrentItem->pNextItem;
      pCurrentItem = pNextItem;
   }
   pFirstItem = nullptr;
   pLastItem = nullptr;
   length = 0;
}

TLinkedList::~TLinkedList()
{
   Clear();
}

std::ostream& operator<<(std::ostream& os, const TLinkedList& list)
{
   shared_ptr<TLinkedList::Item> pCurrentItem = list.pFirstItem;
   while (pCurrentItem != nullptr)
   {
      os << pCurrentItem->pTriangle->Area();
      if (pCurrentItem != list.pLastItem)
         os << " -> ";
      pCurrentItem = pCurrentItem->pNextItem;
   }
   return os;
}
```

TLikedListItem.h:

```
#pragma once
#include <memory>

using namespace std;

template <class T>
class TLinkedListItem
{
private:
   shared_ptr<T> pValue;
   shared_ptr<TLinkedListItem<T>> pNextItem;

public:
```

```
   TLinkedListItem() {}
   TLinkedListItem(shared_ptr<T> pValue, shared_ptr<TLinkedListItem<T>>
pNextItem);
   virtual ~TLinkedListItem();
   shared_ptr<T> getValuePtr();
   shared_ptr<TLinkedListItem<T>> getNextItemPtr();
   void setNextItemPtr(shared_ptr<TLinkedListItem<T>> pNextItem);
};
```

TLikedListItem.cpp:

```
#include <memory>
#include "triangle.h"
#include "tlinkedlistitem.h"

using namespace std;

template<class T>
inline TLinkedListItem<T>::TLinkedListItem(shared_ptr<T> pValue,
shared_ptr<TLinkedListItem<T>> pNextItem)
{
   this->pValue = pValue;
   this->pNextItem = pNextItem;
}

template<class T>
TLinkedListItem<T>::~TLinkedListItem()
{
}

template<class T>
shared_ptr<T> TLinkedListItem<T>::getValuePtr()
{
   return pValue;
}

template<class T>
shared_ptr<TLinkedListItem<T>> TLinkedListItem<T>::getNextItemPtr()
{
   return pNextItem;
}

template<class T>
void TLinkedListItem<T>::setNextItemPtr(shared_ptr<TLinkedListItem<T>>
pNextItem)
{
   this->pNextItem = pNextItem;
}

template class TLinkedListItem<Triangle>;
```

TItteraror.h:

```
#pragma once
#include <memory>
```

```
using namespace std;

template<typename node, typename T>
class TIterator {
private:
    shared_ptr<node> ptr;

public:
    TIterator(shared_ptr<node> ptr)
    {
        this->ptr = ptr;
    }

    shared_ptr<T> operator*()
    {
        return ptr->getValuePtr();
    }

    shared_ptr<T> operator->()
    {
        return ptr->getValuePtr();
    }

    TIterator<node, T> operator++()
    {
        return ptr = ptr->getNextItemPtr();
    }

    TIterator<node, T> operator++(int)
    {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator==(TIterator<node, T> const& other)
    {
        return ptr == other.ptr;
    }

    bool operator!=(TIterator<node, T> const& other)
    {
        return !(*this == other);
    }
};
```

TVector.h:

```
#pragma once
template <class T>
class TVector
{
private:
   const size_t minCount = 3;
   const size_t maxCount = 10;

   T * data;
```

```
    size_t count;
    size_t length;

public:
    TVector();
    TVector(const TVector& other);
    void InsertLast(T item);
    void RemoveLast();
    T Last();
    T& operator[] (const size_t position);
    bool Empty();
    const size_t Length();
    void Clear();
    virtual ~TVector();

    const size_t Find(T item);
    void RemoveAt(const size_t position);

private:
    void increaseAllocation();
};
```

TVector.cpp:

```
#include <memory>
#include <stdexcept>
#include "tvector.h"

using namespace std;

template<class T>
TVector<T>::TVector()
{
    data =  new T[minCount];
    count = minCount;
    length = 0;
}

template<class T>
TVector<T>::TVector(const TVector& other)
{
    // TO DO
    data = new T[minCount];
    count = minCount;
    length = 0;
}

template<class T>
void TVector<T>::InsertLast(T item)
{
    if (length >= maxCount)
        throw runtime_error("cannot add new item to the list as the maximum size
reached");

    if (length >= count)
        increaseAllocation();
```

```
   data[length] = item;
   length++;
}

template<class T>
void TVector<T>::RemoveLast()
{
   if (length == 0)
      throw runtime_error("cannot remove last item in empty list");

   length--;
}

template<class T>
T TVector<T>::Last()
{
   if (length == 0)
      throw runtime_error("cannot get last item from empty list");

   return data[length-1];
}
template<class T>
T& TVector<T>::operator[](const size_t position)
{
   if (position >= length)
      throw runtime_error("invalid position");
   return data[position];
}

template<class T>
bool TVector<T>::Empty()
{
   return length == 0;
}

template<class T>
const size_t TVector<T>::Length()
{
   return length;
}

template<class T>
void TVector<T>::Clear()
{
   length = 0;
}

template<class T>
TVector<T>::~TVector()
{
   Clear();
   if (data != nullptr)
      delete data;
}

template<class T>
```

```
const size_t TVector<T>::Find(T item)
{
   for (size_t i = 0; i <= length; i++)
   {
      if (data[i] == item)
         return i;
   }
   return (size_t)-1;
}

template<class T>
void TVector<T>::RemoveAt(const size_t position)
{
   if (position >= length)
      throw runtime_error("invalid position");
}

template<class T>
void TVector<T>::increaseAllocation()
{
   size_t newCount = min(2 * count, maxCount);
   T* newData = new T[newCount];
   for (int i = 0; i < count; i++)
      newData[i] = data[i];
   delete data;
   data = newData;
   count = newCount;
}

template class TVector<int>;
template class TVector<byte*>;
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.21)
project(oop_exercise_6)

set(CMAKE_CXX_STANDARD 14)

include_directories(.)

add_executable(oop_exercise_6
        figure.cpp
        figure.h
        main.cpp
        point.cpp
        point.h
        tallocator.cpp
        tallocator.h
        tbinarytree.cpp
        tbinarytree.h
        tbinarytreeitem.cpp
        tbinarytreeitem.h
        titterator.cpp
        titterator.h
        tlinkedlist.cpp
        tlinkedlist.h
```

```
tlinkedlistitem.cpp
tlinkedlistitem.h
triangle.cpp
triangle.h
tvector.cpp
tvector.h)
```