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

ЛАБОРАТОРНАЯ РАБОТА №8

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студентка: *Варламова Анна Борисовна*

Группа: *М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Москва, 2021

Задание:

Используя структуру данных, разработанную для лабораторной работы №7, спроектировать и разработать аллокатор памяти для динамической структуры данных. Целью построения аллокатора является 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 операторы `new` и `delete` у классов-фигур.

Вариант №8:

- Фигура: 8-угольник (Octagon)
- Контейнер первого уровня: Связный список (TLinkedList)
- Контейнер второго уровня: Связный список (TLinkedList)

Описание программы:

Исходный код разделён на много файлов:

- `point.h` – описание класса точки
- `point.cpp` – реализация класса точки
- `octagon.h` – описание класса восьмиугольника
- `octagon.cpp` – реализация класса восьмиугольника
- `item.h` – описание элемента списка
- `item.cpp` – реализация элемента списка
- `item2.h` – описание элемента списка2
- `item2.cpp` – реализация элемента списка2
- `titerator.h` – описание и реализация итератора
- `tlinkedlist.h` – описание списка
- `tlinkedlist.cpp` – реализация списка
- `tlinkedlist2.h` – описание списка для хранения блоков памяти
- `tlinkedlist2.cpp` – реализация списка для хранения блоков памяти
- `TAllocationBlock.h` – описание аллокационного блока
- `tallocationblock.cpp` – реализация аллокационного блока
- `main.cpp` – основная программа (для тестирования)

Дневник отладки:

При выполнении работы возникла проблема в подключении библиотек: одна библиотека через промежуточные вызывала другую, а та, в свою очередь, вызывала первую библиотеку. Проблема была решена разделением кода связного списка из прошлой лабораторной и кода списка для хранения блоков памяти.

Тестирование:

```
Octagon created
Octagon created
Octagon created
The list is empty
1
47
47 -> 47
47 -> 47 -> 47
47 -> 47 -> 47 -> 27.5
47 -> 47 -> 27.5 -> 47 -> 27.5
47 -> 47 -> 27.5 -> 47 -> 27.5 -> 24
24 -> 47 -> 47 -> 27.5 -> 47 -> 27.5 -> 24
0

Octagon: (3, 0) (1, 2) (1, 4) (3, 5) (5, 5) (7, 4) (7, 2) (5, 0)
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)
Octagon: (3, 0) (1, 2) (1, 4) (3, 5) (5, 5) (7, 4) (7, 2) (5, 0)

24 -> 47 -> 47 -> 27.5 -> 27.5 -> 24
6
24 -> 47 -> 47 -> 27.5 -> 27.5
47 -> 47 -> 27.5 -> 27.5
47 -> 47 -> 27.5
27.5 -> 47 -> 47 -> 27.5

Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)
```

```
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)

Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (3, 1) (2, 4) (4, 8) (7, 8) (9, 6) (10, 3) (9, 1) (6, 0)
Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)
Octagon: (2, 0) (1, 2) (1, 5) (5, 6) (6, 5) (7, 3) (6, 1) (4, 0)
Octagon created
Octagon deleted

Octagon deleted
Octagon deleted
Octagon deleted
destructor action
```

Вывод:

При выполнении задания я на практике освоила основы реализации и работы с аллокационными блоками. Они позволяют избежать частый запрос системных вызовов для выделения небольших кусков памяти. Навык работы с памятью как таковой крайне полезен, так как при выполнении некоторых задач без работы с ней обойтись тяжело, но я не уверена, что мне ещё где-то это пригодится, потому что аллокаторы – довольно низкоуровневая часть программирования.

Исходный код:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include "point.h"

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

#endif // FIGURE_H
```

point.h:

```cpp
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
  Point();
  Point(std::istream &is);
  Point(double x, double y);

  double dist(Point& other);

  bool operator==(const Point &other);
  friend std::istream& operator>>(std::istream& is, Point& p);
  friend std::ostream& operator<<(std::ostream& os, const Point& p);

private:
  double x_;
  double y_;
};

#endif // POINT_H
```

point.cpp:

```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::dist(Point& other)
{
   double dx = (other.x_ - x_);
   double dy = (other.y_ - y_);
   return std::sqrt(dx*dx + dy*dy);
}

bool Point::operator==(const Point &other)
{
   return ((x_ == other.x_) && (y_ == other.y_));
}

std::istream& operator>>(std::istream& is, Point& p)
{
   is >> p.x_ >> p.y_;
   return is;
}
```

```
std::ostream& operator<<(std::ostream& os, const Point& p)
{
    os << "(" << p.x_ << ", " << p.y_ << ")";
    return os;
}
```

octagon.h:

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include <iostream>
#include "figure.h"
#include "TAllocationBlock.h"


class Octagon : public Figure
{
public:
    Octagon();
    Octagon(Point t_1, Point t_2, Point t_3, Point t_4,
            Point t_5, Point t_6, Point t_7, Point t_8);
    Octagon(const Octagon& other);
    Octagon(std::istream &is);

    Octagon &operator=(const Octagon &other);
    bool operator==(const Octagon &other);
    friend std::istream& operator>>(std::istream& is, Octagon& o);
    friend std::ostream& operator<<(std::ostream& os, const Octagon& o);

    double Area();
    size_t VertexesNumber();

    void * operator new (size_t size);
    void operator delete(void *ptr);

    virtual ~Octagon();

private:
    Point t1;
    Point t2;
    Point t3;
    Point t4;
    Point t5;
    Point t6;
    Point t7;
    Point t8;
    static TAllocationBlock block;
};

#endif // OCTAGON_H
```

octagon.cpp:

```cpp
#include "octagon.h"

#include <iostream>
#include <cmath>

Octagon::Octagon()
   : t1(0.0, 0.0), t2(0.0, 0.0), t3(0.0, 0.0), t4(0.0, 0.0),
   t5(0.0, 0.0), t6(0.0, 0.0), t7(0.0, 0.0), t8(0.0, 0.0) {}
Octagon::Octagon(Point t_1, Point t_2, Point t_3, Point t_4,
                 Point t_5, Point t_6, Point t_7, Point t_8)
   : t1(t_1), t2(t_2), t3(t_3), t4(t_4),
   t5(t_5), t6(t_6), t7(t_7), t8(t_8) {}

Octagon::Octagon(const Octagon& other)
  : Octagon(other.t1, other.t2, other.t3, other.t4,
            other.t5, other.t6, other.t7, other.t8) {}

Octagon::Octagon(std::istream &is)
{
  is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8;
}

Octagon &Octagon::operator=(const Octagon &other)
{
   if (this == &other) {
      return *this;
   }
   t1 = other.t1;
   t2 = other.t2;
   t3 = other.t3;
   t4 = other.t4;
   t5 = other.t5;
   t6 = other.t6;
   t7 = other.t7;
   t8 = other.t8;
   return *this;
}

bool Octagon::operator==(const Octagon &o)
{
   if ((t1 == o.t1) && (t2 == o.t2) && (t3 == o.t3) && (t4 == o.t4) &&
         (t5 == o.t5) && (t6 == o.t6) && (t7 == o.t7) && (t8 == o.t8))
     return true;
     else
     return false;
}

std::istream& operator>>(std::istream& is, Octagon& o)
{
  is >> o.t1 >> o.t2 >> o.t3 >> o.t4 >> o.t5 >> o.t6 >> o.t7 >> o.t8;
  return is;
}

std::ostream& operator<<(std::ostream& os, const Octagon& o)
{
```

```
 os << "Octagon: " << o.t1 << " " << o.t2 << " " << o.t3 << " " << o.t4
         << " " << o.t5 << " " << o.t6 << " " << o.t7 << " " << o.t8;
 return os;
}

size_t Octagon::VertexesNumber()
{
 return (size_t)8;
}

double Heron(Point A, Point B, Point C)
{
 double AB = A.dist(B);
 double BC = B.dist(C);
 double AC = A.dist(C);
 double p = (AB + BC + AC) / 2;
 return sqrt(p * (p - AB) * (p - BC) * (p - AC));
}

double Octagon::Area()
{
 double area1 = Heron(t1, t2, t3);
 double area2 = Heron(t1, t4, t3);
 double area3 = Heron(t1, t4, t5);
 double area4 = Heron(t1, t5, t6);
 double area5 = Heron(t1, t6, t7);
 double area6 = Heron(t1, t7, t8);
 return area1 + area2 + area3 + area4 + area5 + area6;
}

TAllocationBlock Octagon::block(sizeof(Octagon), 1000);

void *Octagon::operator new(size_t size) {
   return block.Allocate();
}

void Octagon::operator delete(void *ptr) {
   block.Deallocate(ptr);
}

Octagon::~Octagon() {}
```

item.h:

```
#ifndef ITEM_H
#define ITEM_H

#define tT template <typename T>
#define sIT std::shared_ptr<Item<T>>
#define sT std::shared_ptr<T>
#define IT Item<T>
#define sI std::shared_ptr<Item>

#include "octagon.h"
#include <memory>
```

```
#include <iostream>

tT
class Item
{
public:
    Item(const sT &s);
    Item(const Item &other);

    sI Left();
    sI Right();

    void ToLeft(sI node);
    void ToRight(sI node);

    sT GetOctagon() const;

    template <class O>
    friend std::ostream &operator<<(std::ostream &os, const Item<O> &node);

    virtual ~Item();

private:
    sT octagon;
    sIT prev;
    sIT next;
};

#endif // ITEM_H
```

item.cpp:

```
#include "item.h"

tT
IT::Item(const sT &o)
{
    this->octagon = o;
    this->next = nullptr;
    this->prev = nullptr;
}

tT
IT::Item(const IT &other)
{
    this->octagon = other.octagon;
    this->next = other.next;
    this->prev = other.prev;
}

tT
sIT IT::Left()
{
    return this->prev;
}
```

```
tT
sIT IT::Right()
{
    return this->next;
}

tT
void IT::ToLeft(sIT node)
{
    this->prev = node;
}

tT
void IT::ToRight(sIT node)
{
    this->next = node;
}


tT
sT IT::GetOctagon() const
{
    return this->octagon;
}

tT
std::ostream &operator<<(std::ostream &os, const IT& node)
{
    os << node.octagon << std::endl;
    return os;
}

tT
IT::~Item() {}

template class Item<Octagon>;
template std::ostream& operator<<(std::ostream& os, const Item<Octagon>& item);
```

tlinkedlist.h:

```
#ifndef TLINKEDLIST_H
#define TLINKEDLIST_H

#define LT TLinkedList<T>

#include "item.h"

tT
class TLinkedList
{
public:
    TLinkedList();
    TLinkedList(const LT &other);
```

```
    sT First();
    sT Last();
    sT GetItem(size_t idx);

    size_t Length();
    bool Empty();

    void InsertFirst(sT octagon);
    void InsertLast(sT octagon);
    void Insert(sT octagon, size_t position);

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

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

    void Clear();
    virtual ~TLinkedList();

private:
    sIT beginning;
    sIT end;
};

#endif // TLINKEDLIST_H
```

tlinkedlist.cpp:

```
#include "tlinkedlist.h"

tT
LT::TLinkedList() : beginning(nullptr), end(nullptr) {}

tT
LT::TLinkedList(const TLinkedList &other)
{
    beginning = other.beginning;
    end = other.end;
}

tT
sT LT::First()
{
    if (beginning == nullptr) {
        std::cout << "The list is empty" << std::endl;
        exit(1);
    }
    return beginning->GetOctagon();
}

tT
sT LT::Last()
```

```
{
    if (end == nullptr) {
        std::cout << "The list is empty" << std::endl;
        exit(1);
    }
    return end->GetOctagon();
}

tT
sT LT::GetItem(size_t position)
{
    size_t n = this->Length();
    if (beginning == nullptr) {
        std::cout << "The list is empty" << std::endl;
        exit(1);
    }
    if (position > n) {
        std::cout << "The is no such position" << std::endl;
        exit(1);
    }
    if (position == 1) {
        return beginning->GetOctagon();
    }
    if (position == n) {
        return end->GetOctagon();
    }
    sIT node = beginning;
    for (size_t i = 1; i < position; ++i) {
        node = node->Right();
    }
    return node->GetOctagon();
}

tT
bool LT::Empty()
{
    return (beginning == nullptr);
}

tT
size_t LT::Length()
{
    size_t size = 0;
    for (sIT i = beginning; i != nullptr; i = i->Right()) {
        ++size;
    }
    return size;
}

tT
void LT::InsertFirst(sT octagon)
{
    sIT node(new IT(octagon));
    if (beginning == nullptr) {
        beginning = (end = node);
```

```
        return;
    }
    node->ToLeft(nullptr);
    node->ToRight(beginning);
    beginning->ToLeft(node);
    beginning = node;
}

tT
void LT::InsertLast(sT octagon)
{
    sIT node(new IT(octagon));
    if (beginning == nullptr) {
        beginning = (end = node);
        return;
    }
    node->ToLeft(end);
    node->ToRight(nullptr);
    end->ToRight(node);
    end = node;
}

tT
void LT::Insert(sT octagon, size_t position)
{
    size_t n = this->Length();
    if (position > n + 1) {
        std::cout << "The is no such position" << std::endl;
        return;
    }
    if (position == 1) {
        InsertFirst(octagon);
        return;
    }
    if (position == n + 1) {
        InsertLast(octagon);
        return;
    }
    sIT node(new IT(octagon));
    sIT now = beginning;
    for (size_t i = 1; i < position; ++i) {
        now = now->Right();
    }
    sIT before = now->Left();
    before->ToRight(node);
    now->ToLeft(node);
    node->ToLeft(before);
    node->ToRight(now);
}

tT
void LT::RemoveFirst()
{
    if (beginning == nullptr) {
        std::cout << "The list is empty" << std::endl;
```

```
        return;
    }
    if (end == beginning) {
        beginning = (end = nullptr);
        return;
    }
    sIT node = beginning;
    beginning = beginning->Right();
    beginning->ToLeft(nullptr);
}

tT
void LT::RemoveLast()
{
    if (end == nullptr) {
        std::cout << "The list is empty" << std::endl;
        return;
    }
    if (end == beginning) {
        beginning = (end = nullptr);
        return;
    }
    sIT node = end;
    end = end->Left();
    end->ToRight(nullptr);
}

tT
void LT::Remove(size_t position)
{
    size_t n = this->Length();
    if (beginning == nullptr) {
        std::cout << "The list is empty" << std::endl;
        return;
    }
    if (position > n) {
        std::cout << "The is no such position" << std::endl;
        return;
    }
    if (position == 1) {
        RemoveFirst();
        return;
    }
    if (position == n) {
        RemoveLast();
        return;
    }
    sIT node = beginning;
    for (size_t i = 1; i < position; ++i) {
        node = node->Right();
    }
    sIT node_left = node->Left();
    sIT node_right = node->Right();
    node_left->ToRight(node_right);
    node_right->ToLeft(node_left);
```

```
}

tT
std::ostream &operator<<(std::ostream &os, const LT &list)
{
    if (list.beginning == nullptr) {
        os << "List is empty" << std::endl;
        return os;
    }
    for (sIT i = list.beginning; i != nullptr; i = i->Right()) {
        if (i->Right() != nullptr)
            os << i->GetOctagon()->Area() << " -> ";
        else
            os << i->GetOctagon()->Area();
    }
    return os;
}

tT
void LT::Clear()
{
    while (beginning != nullptr) {
        RemoveFirst();
    }
}

tT
LT::~TLinkedList()
{
    while (beginning != nullptr) {
        RemoveFirst();
    }
}

template class TLinkedList<Octagon>;
template std::ostream& operator<<(std::ostream& os, const TLinkedList<Octagon>& list);
```

titerator.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->GetOctagon();
    }

    std::shared_ptr<T> operator->()
    {
        return node_ptr->GetOctagon();
    }
    // --i ++i
    TIterator operator++(int)
    {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    TIterator operator--(int)
    {
        TIterator iter(*this);
        --(*this);
        return iter;
    }
    // i++ i--
    void operator++()
    {
        node_ptr = node_ptr->Right();
    }

    void operator--()
    {
        node_ptr = node_ptr->Left();
    }

    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
```

main.cpp:

```
#include "tlinkedlist.h"
```

```
int main(void)
{
   TLinkedList<Octagon> l;
   Point x1(3, 1);
   Point x2(2, 4);
   Point x3(4, 8);
   Point x4(7, 8);
   Point x5(9, 6);
   Point x6(10, 3);
   Point x7(9, 1);
   Point x8(6, 0);

   Point y1(3, 0);
   Point y2(1, 2);
   Point y3(1, 4);
   Point y4(3, 5);
   Point y5(5, 5);
   Point y6(7, 4);
   Point y7(7, 2);
   Point y8(5, 0);

   Point z1(2, 0);
   Point z2(1, 2);
   Point z3(1, 5);
   Point z4(5, 6);
   Point z5(6, 5);
   Point z6(7, 3);
   Point z7(6, 1);
   Point z8(4, 0);

   std::shared_ptr<Octagon> o1(new Octagon(x1, x2, x3, x4, x5, x6, x7, x8));
   std::shared_ptr<Octagon> o2(new Octagon(y1, y2, y3, y4, y5, y6, y7, y8));
   std::shared_ptr<Octagon> o3(new Octagon(z1, z2, z3, z4, z5, z6, z7, z8));

   /*std::shared_ptr<Octagon> o1(new Octagon);
   std::shared_ptr<Octagon> o2(new Octagon);
   std::shared_ptr<Octagon> o3(new Octagon);
   std::cin >> *o1 >> *o2 >> *o3;*/
   l.Remove(5);
   std::cout << l.Empty() << std::endl;
   l.Insert(o1, 1);
   std::cout << l << std::endl;
   l.Insert(o1, 2);
   std::cout << l << std::endl;
   l.Insert(o1, 3);
   std::cout << l << std::endl;
   l.Insert(o3, 4);
   std::cout << l << std::endl;
   l.Insert(o3, 3);
   std::cout << l << std::endl;
   l.Insert(o2, 6);
   std::cout << l << std::endl;
   l.Insert(o2, 1);
   std::cout << l << std::endl;
```

```
    std::cout << l.Empty() << std::endl;
    std::cout << std::endl;
    for (auto i : l) {
        std::cout << *i << std::endl;
    }
    std::cout << std::endl;
    l.Remove(5);
    std::cout << l << std::endl;
    std::cout << l.Length() << std::endl;
    l.Remove(l.Length());
    std::cout << l << std::endl;
    l.RemoveFirst();
    std::cout << l << std::endl;
    l.RemoveLast();
    std::cout << l << std::endl;
    l.InsertFirst(o3);
    std::cout << l << std::endl;
    std::cout << std::endl;
    for (auto i : l) {
        std::cout << *i << std::endl;
    }
    std::cout << std::endl;
    std::cout << *l.GetItem(1) << std::endl;
    std::cout << *l.GetItem(2) << std::endl;
    std::cout << *l.GetItem(3) << std::endl;
    std::cout << *l.GetItem(4) << std::endl;
    std::cout << *l.Last() << std::endl;
    return 0;
}
```

item2.h

```
#ifndef ITEM2_H
#define ITEM2_H

#include <memory>

class Item2 {
public:
    Item2(void *ptr);

    Item2* to_right(Item2* next);
    Item2* Next();
    void* GetItem();

    virtual ~Item2();
private:
    void* link;
    Item2* next;
};

#endif // ITEM2_H
```

item2.cpp

```
#include "item2.h"
#include <iostream>


Item2::Item2(void* link) {
    this->link = link;
    this->next = nullptr;
}

Item2* Item2::to_right(Item2* next) {
    Item2* set = this->next;
    this->next = next;
    return set;
}

Item2* Item2::Next() {
    return this->next;
}

void* Item2::GetItem() {
    return this->link;
}

Item2::~Item2() {}
```

tlinkedlist2.h

```
#ifndef TLINKEDLIST2_H
#define TLINKEDLIST2_H


#include "item2.h"
#include "titerator.h"
#include <memory>
#include <iostream>

class TLinkedList2{
public:
    TLinkedList2();
    void InsertFirst(void *link);
    void InsertLast(void *link);
    void Insert(size_t position, void *link);
    size_t Length();
    bool Empty();
    void Remove(size_t &position);
    void Clear();

    void* GetItem();

    virtual ~TLinkedList2();
private:
    Item2* first;
};
#endif // TLINKEDLIST2_H
```

tlinkedlist2.cpp

```
#include "octagon.h"
#include "tlinkedlist2.h"
#include <iostream>

TLinkedList2::TLinkedList2() {
    first = nullptr;
}

void TLinkedList2::InsertFirst(void* link) {
    auto *other = new Item2(link);
    other->to_right(first);
    first = other;
}

void TLinkedList2::Insert(size_t position, void *link) {
    Item2 *iter = this->first;
    auto *other = new Item2(link);
    if (position == 1) {
        other->to_right(iter);
        this->first = other;
    } else {
        if (position <= this->Length()) {
            for (size_t i = 1; i < position - 1; ++i)
                iter = iter->Next();
            other->to_right(iter->Next());
            iter->to_right(other);
        }
    }
}

void TLinkedList2::InsertLast(void *link) {
    auto *other = new Item2(link);
    Item2 *iter = this->first;
    if (first != nullptr) {
        while (iter->Next() != nullptr) {
            iter = iter->to_right(iter->Next());
        }
        iter->to_right(other);
        other->to_right(nullptr);
    }
    else {
        first = other;
    }
}

size_t TLinkedList2::Length() {
    size_t len = 0;
    Item2* item = this->first;
    while (item != nullptr) {
        item = item->Next();
        len++;
```

```
    }
    return len;
}

bool TLinkedList2::Empty() {
    return first == nullptr;
}

void TLinkedList2::Remove(size_t &position) {
    Item2 *iter = this->first;
    if (position <= this->Length()) {
        if (position == 1) {
            this->first = iter->Next();
        } else {
            size_t i = 1;
            for (i = 1; i < position - 1; ++i) {
                iter = iter->Next();
            }
            iter->to_right(iter->Next()->Next());
        }

    } else {
        std::cout << "error" << std::endl;
    }
}

void TLinkedList2::Clear() {
    first = nullptr;
}

void * TLinkedList2::GetItem() {
    return this->first->GetItem();
}

TLinkedList2::~TLinkedList2() {
    delete first;
}
```

TAllacationBlock.h

```
#ifndef TALLOCATIONBLOCK_H
#define TALLOCATIONBLOCK_H

#include <iostream>
#include <cstdlib>
#include "tlinkedlist2.h"

class TAllocationBlock {
public:
    TAllocationBlock(size_t size, size_t count);

    void *Allocate();
    void Deallocate(void *ptr);
    bool Empty();
```

```
    size_t Size();

    virtual ~TAllocationBlock();

private:
    char *used;
    TLinkedList2 unused;
};

#endif //TALLOCATIONBLOCK_H
```

tallocationblock.cpp

```
#include "TAllocationBlock.h"

TAllocationBlock::TAllocationBlock(size_t size, size_t count) {
    used = (char *)malloc(size * count);
    for (size_t i = 0; i < count; ++i) {
        void *ptr = (void *)malloc(sizeof(void *));
        ptr = used + i * size;
        unused.InsertLast(ptr);
    }
}

void *TAllocationBlock::Allocate() {
    if (!unused.Empty()) {
        void *res = unused.GetItem();
        size_t first = 1;
        unused.Remove(first);
        std::cout << "Octagon created" << std::endl;
        return res;
    } else {
        throw std::bad_alloc();
    }
}

void TAllocationBlock::Deallocate(void *ptr) {
    unused.InsertFirst(ptr);
    std::cout << "Octagon deleted" << std::endl;
}

bool TAllocationBlock::Empty() {
    return unused.Empty();
}

size_t TAllocationBlock::Size() {
    return unused.Length();
}

TAllocationBlock::~TAllocationBlock() {
    while (!unused.Empty()) {
        size_t first = 1;
        unused.Remove(first);
    }
```

```
    free(used);
    std::cout << "destructor action" << std::endl;
}
```