Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 1

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

Лабораторная работа №1

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов задания. Классы должны удовлетворять следующим правилам:

- Должны иметь общий родительский класс Figure.
- Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
- Должный имеет общий виртуальный метод расчета площади фигуры – Square.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на экран и их площадь.

ОПИСАНИЕ

Классом называется составной тип данных, элементами которого являются функции и переменные (поля). В основу понятия класс положен тот факт, что «над объектами можно совершать различные операции». Свойства объектов описываются с помощью полей классов, а действия над объектами описываются с помощью функций, которые называются методами класса. Класс имеет имя, состоит из полей, называемых членами класса, и функций — методов класса.

Виртуальные функции — специальный вид функций-членов класса. Виртуальная функция отличается об обычной функции тем, что для обычной функции связывание вызова функции с ее определением осуществляется на этапе компиляции. Для виртуальных функций это происходит во время выполнения программы.

Для объявления виртуальной функции используется ключевое слово virtual. Функция-член класса может быть объявлена как виртуальная, если

- класс, содержащий виртуальную функцию, базовый в иерархии порождения;
- реализация функции зависит от класса и будет различной в каждом порожденном классе.

ЛИСТИНГ ПРОГРАММЫ

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
class Figure
{
  public:
    virtual double Square() = 0;
    virtual void Print() = 0;
    virtual ~Figure(){};
};
#endif /* FIGURE_H */
```

triangle.h

```
#ifndef
TRIANGLE_H
            #defineTRIANGLE_H
            #include <cstdlib>
            #include <iostream>
```

```
#include "figure.h"

class Triangle : public Figure{
public:
    Triangle();
    Triangle(std::istream &is);
    Triangle(size_t i,size_t j,size_t k);
    Triangle(const Triangle& orig);

    double Square() override;
    void   Print() override;

    virtual ~Triangle();
private:
    size_t side_a;
    size_t side_b;
    size_t side_c;
};

#endif /* TRIANGLE_H */
```

triangle.cpp

```
#include
"triangle.h"
  #include <iostream>
  #include <cmath>

  Triangle::Triangle() : Triangle(0, 0, 0) {
  }

  Triangle::Triangle(size_t i, size_t j, size_t k) : side_a(i), side_b(j),
  side_c(k) {
      std::cout << "Triangle created: " << side_a << ", " << side_b << ", " <<
  side_c << std::endl;
  }

  Triangle::Triangle(std::istream &is) {
      is >> side_a;
      is >> side_b;
      is >> side_c;
  }

  Triangle::Triangle(const Triangle& orig) {
      std::cout << "Triangle copy created" << std::endl;
      side_a = orig.side_a;
      side_b = orig.side_b;
      side_c = orig.side_c;
  }

  double Triangle::Square() {
      double p = double(side_a + side_b + side_c) / 2.0;
      return sqrt(p * (p - double(side_a))*(p - double(side_b))*(p -
  double(side_c)));

  }

  void Triangle::Print() {
      std::cout << "a=" << side_a << ", b=" << side_b << ", c=" << side_c <<
  std::endl;

  }

  Triangle::~Triangle() {
      std::cout << "Triangle deleted" << std::endl;
  }
```

square.h

```
#ifndef
FSQUARE_H
            #defineFSQUARE_H
```

```
#include <cstdlib>
#include <iostream>
#include "figure.h"

class FSquare : public Figure{
public:
    FSquare();
    FSquare(std::istream &is);
    FSquare(size_t a);
    FSquare(const FSquare& orig);

    double Square() override;
    void   Print() override;

    virtual ~FSquare();
private:
    size_t side_a;
};

#endif /* FSQUARE_H */
```

square.cpp

```
#include
"square.h"
 #include <iostream>
 #include <cmath>

 FSquare::FSquare() : FSquare(0) {
 }

 FSquare::FSquare(size_t a) : side_a(a) {
     std::cout << "FSquare created: " << side_a << std::endl;
 }

 FSquare::FSquare(std::istream &is) {
     is >> side_a;
 }

 FSquare::FSquare(const FSquare& orig) {
     std::cout << "FSquare copy created" << std::endl;
     side_a = orig.side_a;
 }

 double FSquare::Square() {
     double S = pow(side_a , 2);
     return S;

 }

 void FSquare::Print() {
     std::cout << "a=" << side_a << std::endl;

 }

 FSquare::~FSquare() {
     std::cout << "Square deleted" << std::endl;
 }
```

octagon.h

```
#ifndef
OCTAGON_H
#defineOCTAGON_H
#include <cstdlib>
#include <iostream>
#include "figure.h"

class Octagon : public Figure{
public:
    Octagon();
    Octagon(std::istream &is);
```

```
    Octagon(size_t a);
    Octagon(const Octagon& orig);

    double Square() override;
    void   Print() override;

    virtual ~Octagon();
private:
    size_t a;
};

#endif /* OCTAGON_H */
```

octagon.cpp

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

  Octagon::Octagon() : Octagon(0) {
  }

  Octagon::Octagon(size_t a) :
          a(a){
      std::cout << "Octagon with side is " << a << " created." << std::endl;
  }

  Octagon::Octagon(std::istream &is) {
      is >> a;
  }

  Octagon::Octagon(const Octagon& orig) {
      std::cout << "Octagon copy created" << std::endl;
      a = orig.a;
  }

  double Octagon::Square() {
      double S = 2*(1 + sqrt(2))*a;
      return S;

  }

  void Octagon::Print() {
      std::cout << "a=" << a << std::endl;

  }

  Octagon::~Octagon() {
      std::cout << "Octagon deleted" << std::endl;
  }
```

main.cpp

```
#include
<cstdlib>
#include "triangle.h"
#include "square.h"
#include "octagon.h"

using namespace std;

void print_menu(){
              cout << "Выберие фигуру:" << endl;
              cout << 1 << '\t' << "Восьмиугольник" << endl;
              cout << 2 << '\t' << "Треугольник" << endl;
              cout << 3 << '\t' << "Квадрат" << endl;
       }

int main(int argc, char** argv) {
       int c;
       Figure *ptr;
```

```
                print_menu();

                while (cin >> c){


                        switch (c){
                                case 1:

                                        cout << "Введите сторону правильного
восьмиугольника" << endl;
                                        ptr = new Octagon(cin);

                                case 2:

                                        cout << "Введите стороны треугольника" << endl;
                                        ptr = new Triangle(cin);
                                case 3:

                                        cout << "Введите сторону квадрата" << endl;
                                        ptr = new FSquare(cin);

                                default:
                                        print_menu();
                                        continue();

                        }
                        ptr->Print();
                        cout << "S = " << ptr->Square() << endl;
                        delete ptr;
                        break;
                }


            return 0;
        }
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
    1. Trapeze
    2. Rhombus
    3. Pentagon
    0. Exit
    Enter figure number:1
    Trapeze created
    Enter first side: 2
    Enter second side: 4
    Enter the height: 3
    Correct value
    Trapeze:
    side 1:  2 side 2: 4 height: 3
    Square of trapeze:9
    Trapeze deleted

    1. Trapeze
    2. Rhombus
    3. Pentagon
    0. Exit
    Enter figure number:2
    Rhombus created:
    Enter the angle: 30
    Enter the side: 5
    Correct value
    Rhombus:
    angle: 30 side: 5
    Square of rhombus:12.5
    Rhombus deleted

    1. Trapeze
    2. Rhombus
    3. Pentagon
    0. Exit
    Enter figure number:3
    Pentagon created:
    Enter coordinates by point в„– 1 2 3
```

```
Enter coordinates by point в„- 2 -2 5
Enter coordinates by point в„- 3 3 -4
Enter coordinates by point в„- 4 7 0
Enter coordinates by point в„- 5 4 1
Pentagon:
Coordinates of pentagon:
X: 2 Y: 3
X: -2 Y: 5
X: 3 Y: -4
X: 7 Y: 0
X: 4 Y: 1
Square of pentagon:27
Pentagon deleted
```

ВЫВОД

В первой лабораторной работе в курсе изучения объектно-ориентированного программирования нам предлагалось ознакомиться с базовыми понятиями C++ такими, как классы, наследование, конструкторы и деструкторы, виртуальные функции и тд. В ходе написания программ, я лучше разобрался в этих понятиях и изучил их принципы работы.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ
Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 2

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантов задания и (реализованную в ЛР1). Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен содежержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнераю
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

В C++ разработана новая библиотека ввода-вывода iostream, использующая концепцию объектно-ориентированного программирования:

Библиотека iostream определяет три стандартных потока:

- cin стандартный входной поток (stdin в C)
- cout стандартный выходной поток (stdout в C)
- cerr стандартный поток вывода сообщений об ошибках (stderr в C)

Механизм перегрузки операций позволяет обеспечить более традиционную и удобную запись действий над объектами. Для перегрузки встроенных операторов используется ключевое слова operator.

Тип возвращаемого значения должен быть отличным от void, если необходимо использовать перегруженную операцию внутри другого выражения.

Имеется два способа описания функции, соответствующей переопределяемой операции:

- если функция задается как обычная функция-элемент класса, то первым операндом операции является объект класса, указатель на который передается неявным параметром this;

- если первый операнд переопределяемой операции не является объектом некоторого класса, либо требуется передавать в качестве операнда не указатель, а сам объект (значение), то соответствующая функция должна быть определена как дружественная классу с полным списком аргументов.

ЛИСТИНГ ПРОГРАММЫ

На протяжении всех лабораторных работ мы работаем с классом фигур, в частности с фигурами, соответствующими определенному варианту. Такие файлы, как Figure.h, Trapeze.h, Trapeze.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp не изменяются.

TNode.h

```
#ifndef TSTACKITEM_H
#define TSTACKITEM_H

#include "octagon.h"
class TNode {
public:
    TNode(const Octagon& octagon);
    TNode(const TNode& orig);
    friend std::ostream& operator<<(std::ostream& os, const TNode& obj);

    TNode* SetNext(TNode* next);
    TNode* GetNext();
    Octagon GetOctagon() const;

    virtual ~TNode();
private:
    Octagon octagon;
    TNode *next;
};

#endif /* TSTACKITEM_H */
```

TNode.cpp

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

TNode::TNode(const Octagon& octagon) {
    this->octagon = octagon;
    this->next = nullptr;
    std::cout << "List item: created" << std::endl;
}

TNode::TNode(const TNode& orig) {
    this->octagon = orig.octagon;
    this->next = orig.next;
    std::cout << "List item: copied" << std::endl;
}

TNode* TNode::SetNext(TNode* next) {
    TNode* old = this->next;
    this->next = next;
    return old;
}

Octagon TNode::GetOctagon() const {
    return this->octagon;
}

TNode* TNode::GetNext() {
    return this->next;
}

TNode::~TNode() {
    std::cout << "List item: deleted" << std::endl;
    delete next;

}

std::ostream& operator<<(std::ostream& os, const TNode& obj) {
    os << "[ " << obj.octagon << "]" << std::endl;
    return os;
```

```
}
```

Main.cpp

```
#include
<cstdlib>
#include <iostream>

#include "octagon.h"
#include "TNode.h"
#include "list.h"

// Simple list on pointers
int main(int argc, char** argv) {

    TList list;

    list.push(Octagon(1),0);
    list.push(Octagon(2),1);
    list.push(Octagon(1)+Octagon(2),1);

    std::cout << list;


    Octagon t;

    t = list.pop(2); std::cout << t;
    t = list.pop(1); std::cout << t;
    t = list.pop(0); std::cout << t;

    return 0;
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
3. Delete item
4. Print queue
5. Delete queue
6. Menu
0. Exit

1
Queue created
2
Enter first side: 1
Enter second side: 1
Enter the height: 1
Correct value
Trapeze created:
Trapeze copied
Queue item: created
Trapeze deleted
2
Enter first side: 2
Enter second side: 2
Enter the height: 2
Correct value
Trapeze created:
Trapeze copied
Queue item: created
Trapeze deleted
2
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
Trapeze created:
Trapeze copied
Queue item: created
Trapeze deleted
4
[a = 1, b = 1, h = 1]
[a = 2, b = 2, h = 2]
[a = 3, b = 3, h = 3]
```

```
3
Trapeze created:
Trapeze copy created
Trapeze copied
Trapeze deleted
Queue item: deleted
Trapeze deleted
a = 1, b = 1, h = 1
Trapeze deleted
4
[a = 2, b = 2, h = 2]
[a = 3, b = 3, h = 3]
3
Trapeze created:
Trapeze copy created
Trapeze copied
Trapeze deleted
Queue item: deleted
Trapeze deleted
a = 2, b = 2, h = 2
Trapeze deleted
4
[a = 3, b = 3, h = 3]
5
Queue item: deleted
Trapeze deleted
Queue deleted
0
```

ВЫВОД

В ходе второй лабораторной работы я создал динамическую структуру данных, являющейся очередью. Объекты передаются контейнеру первого уровня по вводимым с консоли значениям.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ
Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 3

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1). Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Класс-контейнер должен содержать объекты используя std:shared_ptr<…>.
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp)

ЛИСТИНГ ПРОГРАММЫ

TList.h

```
#ifndef TLIST_H
#define TLIST_H

#include <cstdint>
#include "octagon.h"
#include "square.h"
#include "triangle.h"
#include "TListItem.h"

class TList
{
public:
    TList();
    void Push(std::shared_ptr<Figure> &obj);
    const bool IsEmpty() const;
    uint32_t GetLength();
    std::shared_ptr<Figure> Pop();
    friend std::ostream& operator<<(std::ostream &os, const TList &list);
    void Del();
    virtual ~TList();

private:
    uint32_t length;
    std::shared_ptr<TListItem> head;

    std::shared_ptr<Figure> PopFirst();
    std::shared_ptr<Figure> PopLast();
    std::shared_ptr<Figure> PopAtIndex(int32_t ind);
};

#endif
```

TList.cpp

```
#include "TList.h"
#include <iostream>
#include <cstdint>
```

```
TList::TList()
{
    head = nullptr;
    length = 0;
}

void TList::Push(std::shared_ptr<Figure> &obj) {
    int32_t index = 0;
    std::cout << "Enter index = ";
    std::cin >> index;
    if (index > this->GetLength()|| index < 0) {
        std::cerr << "This index doesn't exist\n";
        return;
    }
    std::shared_ptr<TListItem> newItem = std::make_shared<TListItem>(obj);

    if (index == 0) {
        newItem->SetNext(head);
        head = newItem;
        ++length;
        return;
    } else {
        std::shared_ptr<TListItem> cur = this->head;
        std::shared_ptr<TListItem> prev = this->head;
        for (int i = 0; i < index; ++i){
            prev = cur;
            cur = cur->GetNext();
        }
        newItem->SetNext(cur);
        prev->SetNext(newItem);
    }
    ++length;
}

uint32_t TList::GetLength()
{
    return this->length;
}

const bool TList::IsEmpty() const
{
    return head == nullptr;
}

std::shared_ptr<Figure> TList::Pop()
{
    int32_t ind = 0;
    std::cout << "Enter index = ";
    std::cin >> ind;
    std::shared_ptr<Figure> res;
    if (ind > this->GetLength() - 1 || ind < 0 || this->IsEmpty()) {
        std::cout << "Change index" << std::endl;
        return res;
    }

    if (ind == 0) {
        res = this->PopFirst();
    } else if (ind == this->GetLength() - 1) {
        res = this->PopLast();
    } else {
        res = this->PopAtIndex(ind);
    }
    --length;
    return res;
}

std::shared_ptr<Figure> TList::PopAtIndex(int32_t ind)
{
    std::shared_ptr<TListItem> tmp = this->head;
    for(int32_t i = 0; i < ind - 1; ++i) {
        tmp = tmp->GetNext();
    }
    std::shared_ptr<TListItem> removed = tmp->GetNext();
```

```
    std::shared_ptr<Figure> res = removed->GetFigure();
    std::shared_ptr<TListItem> nextItem = removed->GetNext();
    tmp->SetNext(nextItem);
    nextItem->SetPrev(tmp);
    return res;
}

std::shared_ptr<Figure> TList::PopFirst()
{
    if (this->GetLength() == 1) {
        std::shared_ptr<Figure> res = this->head->GetFigure();
        this->head = nullptr;
        return res;
    }
    std::shared_ptr<TListItem> tmp = this->head;
    std::shared_ptr<Figure> res = tmp->GetFigure();
    this->head = this->head->GetNext();
    this->head->SetPrev(nullptr);
    return res;
}

std::shared_ptr<Figure> TList::PopLast()
{
    if (this->GetLength() == 1) {
        std::shared_ptr<Figure> res = this->head->GetFigure();
        this->head = nullptr;
        return res;
    }
    std::shared_ptr<TListItem> tmp = this->head;
    while(tmp->GetNext()->GetNext()) {
        tmp = tmp->GetNext();
    }
    std::shared_ptr<TListItem> removed = tmp->GetNext();
    std::shared_ptr<Figure>res = removed->GetFigure();
    tmp->SetNext(removed->GetNext());
    return res;
}


std::ostream& operator<<(std::ostream &os, const TList &list)
{
    if (list.IsEmpty()) {
        os << "The list is empty." << std::endl;
        return os;
    }

    std::shared_ptr<TListItem> tmp = list.head;
    for(int32_t i = 0; tmp; ++i) {
        os << "idx: " << i << "   ";
        tmp->GetFigure()->Print();
        os << std::endl;
        tmp = tmp->GetNext();
    }

    return os;
}

void TList::Del()
{
    while(!this->IsEmpty()) {
        this->PopFirst();
        --length;
    }
}

TList::~TList()
{
/*    while(!this->IsEmpty()) {
        this->PopFirst();
        --length;
    }*/
}
```

TListItem.h

```
#ifndef TLISTITEM_H
#define TLISTITEM_H

#include <memory>
#include "octagon.h"
#include "square.h"
#include "triangle.h"

class TListItem
{
public:
    TListItem(const std::shared_ptr<Figure> &obj);

    std::shared_ptr<Figure> GetFigure() const;
    std::shared_ptr<TListItem> GetNext();
    std::shared_ptr<TListItem> GetPrev();
    void SetNext(std::shared_ptr<TListItem> item);
    void SetPrev(std::shared_ptr<TListItem> item);
    friend std::ostream& operator<<(std::ostream &os, const TListItem &obj);

    virtual ~TListItem(){};

private:
    std::shared_ptr<Figure> item;
    std::shared_ptr<TListItem> next;
    std::shared_ptr<TListItem> prev;
};

#endif
```

TListItem.cpp

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

                TListItem::TListItem(const std::shared_ptr<Figure> &obj)
                {
                    this->item = obj;
                    this->next = nullptr;
                    this->prev = nullptr;
                }

                std::shared_ptr<Figure> TListItem::GetFigure() const
                {
                    return this->item;
                }

                std::shared_ptr<TListItem> TListItem::GetNext()
                {
                    return this->next;
                }

                std::shared_ptr<TListItem> TListItem::GetPrev()
                {
                    return this->prev;
                }

                void TListItem::SetNext(std::shared_ptr<TListItem> item)
                {
                    this->next = item;
                }

                void TListItem::SetPrev(std::shared_ptr<TListItem> item)
                {
                    this->prev = item;
                }

                std::ostream& operator<<(std::ostream &os, const TListItem &obj)
                {
                    os << obj.item << std::endl;
                    return os;
                }
```

Main.cpp

```
#include
<iostream>
#include <memory>
#include <cstdlib>
#include <cstring>
#include "octagon.h"
#include "square.h"
#include "triangle.h"
#include "TList.h"

void menu()
{
    std::cout << "Choose an operation:" << std::endl;
    std::cout << "1) Add triangle" << std::endl;
    std::cout << "2) Add square" << std::endl;
    std::cout << "3) Add octagon" << std::endl;
    std::cout << "4) Delete figure from list" << std::endl;
    std::cout << "5) Print list"  << std::endl;
    std::cout << "0) Exit" << std::endl;
}

int main(void)
{
    int32_t act = 0;
    TList list;
    std::shared_ptr<Figure> ptr;
    do {
        menu();
        std::cin >> act;
        switch(act) {
            case 1:
                ptr = std::make_shared<Triangle>(std::cin);
                list.Push(ptr);
                break;
            case 2:
                ptr = std::make_shared<FSquare>(std::cin);
                list.Push(ptr);
                break;
            case 3:
                ptr = std::make_shared<Octagon>(std::cin);
                list.Push(ptr);
                break;
            case 4:
                list.Pop();
                break;
            case 5:
                std::cout << list << std::endl;
                break;
            case 0:
                list.Del();
                break;
            default:
                std::cout << "Incorrect command" << std::endl;;
                break;
        }
    } while (act);
    return 0;
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
       3. Add trapeze
       4. Add rhombus
       5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
```

```
2
3
Enter first side: 1
Enter second side: 1
Enter the height: 1
Correct value
Queue item: created
2
4
Enter the angle: 30
Enter the side: 2
Correct value
Queue item: created
2
5
Enter the side: 3
Correct value
Queue item: created
7
Trapeze:
side 1:  1 side 2: 1 height: 1
Rhombus:
angle: 30 side: 2
Pentagon:
Side: 3
6
Queue item: deleted
Trapeze deleted
7
Rhombus:
angle: 30 side: 2
Pentagon:
Side: 3
6
Queue item: deleted
Rhombus deleted
7
Pentagon:
Side: 3
6
7
0
Queue item: deleted
Pentagon deleted
```

ВЫВОД

В лабораторной работе №3 мы добавляем оставшиеся две фигуры и оптимизируем программу с помощью умных указателей. Таким образом, умные указатели помогают нам с очищением памяти, когда это необходимо и избегать какие-либо утечки памяти.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 4

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

Лабораторная работа №4

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1). Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной работы 1.
- Шаблон класса-контейнера должен содержать объекты используя std:shared_ptr<…>.
- Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).
- Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
- Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

ОПИСАНИЕ

Шаблоны (templates) - очень мощное средство. Шаблонные функции и классы позволяют очень сильно упростить программисту жизнь и сберечь огромное количество времени, сил и нервов. Если вам покажется, что шаблоны не сильно-то и значимая тема для изучения, знайте - вы заблуждаетесь.

У шаблонных функций должен быть аргумент, чтобы компилятор мог определить какой именно тип использовать. В шаблонах можно использовать несколько параметрических типов, и конечно же можно смешивать параметрические типы со стандартными (только нужно позаботиться о правильном приведении типов).

При создании объекта, после имени класса нужно поставить угловые скобки, в которых указать нужный тип. После этого объекты используются так, как мы привыкли. У шаблонных классов есть одна потрясающая особенность - кроме стандартных типов, они могут работать и с пользовательскими.

ЛИСТИНГ ПРОГРАММЫ

TList.h

```
#ifndef
TLIST_H
          #define TLIST_H

          #include <cstdint>
          #include "octagon.h"
          #include "square.h"
          #include "triangle.h"
          #include "TListItem.h"

          template <class T>
          class TList
          {
          public:
              TList();
              void Push(std::shared_ptr<T> &obj);
              const bool IsEmpty() const;
              uint32_t GetLength();
              std::shared_ptr<T> Pop();
```

```
    template <class A> friend std::ostream& operator<<(std::ostream &os, const
TList<A> &list);
    void Del();
    virtual ~TList();

private:
    uint32_t length;
    std::shared_ptr<TListItem<T>> head;

    std::shared_ptr<T> PopFirst();
    std::shared_ptr<T> PopLast();
    std::shared_ptr<T> PopAtIndex(int32_t ind);
};

#endif
```

TList.cpp

```
#include "TList.h"

#include <iostream>
#include <cstdint>

template <class T>
TList<T>::TList()
{
    head = nullptr;
    length = 0;
}

template <class T>
void TList<T>::Push(std::shared_ptr<T> &obj)
{
    int32_t index = 0;
    std::cout << "Enter index = ";
    std::cin >> index;
    if (index > this->GetLength()|| index < 0) {
        std::cerr << "This index doesn't exist\n";
        return;
    }
    std::shared_ptr<TListItem<T>> newItem =
std::make_shared<TListItem<T>>(obj);

    if (index == 0) {
        newItem->SetNext(head);
        head = newItem;
        ++length;
        return;
    } else {
        std::shared_ptr<TListItem<T>> cur = this->head;
        std::shared_ptr<TListItem<T>> prev = this->head;
        for (int i = 0; i < index; ++i){
            prev = cur;
            cur = cur->GetNext();
        }
        newItem->SetNext(cur);
        prev->SetNext(newItem);
    }
    ++length;
}

template <class T>
uint32_t TList<T>::GetLength()
{
    return this->length;
}

template <class T>
const bool TList<T>::IsEmpty() const
{
    return head == nullptr;
}

template <class T>
```

```
std::shared_ptr<T> TList<T>::Pop()
{
    int32_t ind = 0;
    std::cout << "Enter index = ";
    std::cin >> ind;
    std::shared_ptr<T> res;
    if (ind > this->GetLength() - 1 || ind < 0 || this->IsEmpty()) {
        std::cout << "Change index" << std::endl;
        return res;
    }

    if (ind == 0) {
        res = this->PopFirst();
    } else if (ind == this->GetLength() - 1) {
        res = this->PopLast();
    } else {
        res = this->PopAtIndex(ind);
    }
    --length;
    return res;
}

template <class T>
std::shared_ptr<T> TList<T>::PopAtIndex(int32_t ind)
{
    std::shared_ptr<TListItem<T>> tmp = this->head;
    for(int32_t i = 0; i < ind - 1; ++i) {
        tmp = tmp->GetNext();
    }
    std::shared_ptr<TListItem<T>> removed = tmp->GetNext();
    std::shared_ptr<T> res = removed->GetFigure();
    std::shared_ptr<TListItem<T>> nextItem = removed->GetNext();
    tmp->SetNext(nextItem);
    nextItem->SetPrev(tmp);
    return res;
}

template <class T>
std::shared_ptr<T> TList<T>::PopFirst()
{
    if (this->GetLength() == 1) {
        std::shared_ptr<T> res = this->head->GetFigure();
        this->head = nullptr;
        return res;
    }
    std::shared_ptr<TListItem<T>> tmp = this->head;
    std::shared_ptr<T> res = tmp->GetFigure();
    this->head = this->head->GetNext();
    this->head->SetPrev(nullptr);
    return res;
}

template <class T>
std::shared_ptr<T> TList<T>::PopLast()
{
    if (this->GetLength() == 1) {
        std::shared_ptr<T> res = this->head->GetFigure();
        this->head = nullptr;
        return res;
    }
    std::shared_ptr<TListItem<T>> tmp = this->head;
    while(tmp->GetNext()->GetNext()) {
        tmp = tmp->GetNext();
    }
    std::shared_ptr<TListItem<T>> removed = tmp->GetNext();
    std::shared_ptr<T> res = removed->GetFigure();
    tmp->SetNext(removed->GetNext());
    return res;
}


template <class T>
std::ostream& operator<<(std::ostream &os, const TList<T> &list)
```

```
{
    if (list.IsEmpty()) {
        os << "The list is empty." << std::endl;
        return os;
    }

    std::shared_ptr<TListItem<T>> tmp = list.head;
    for(int32_t i = 0; tmp; ++i) {
        os << "idx: " << i << "   ";
        tmp->GetFigure()->Print();
        os << std::endl;
        tmp = tmp->GetNext();
    }

    return os;
}

template <class T>
void TList<T>::Del()
{
    while(!this->IsEmpty()) {
        this->PopFirst();
        --length;
    }
}

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

#include "figure.h"
template class TList<Figure>;
template std::ostream& operator<<(std::ostream &out, const TList<Figure> &obj);
```

TListItem.h

```
#ifndef TLISTITEM_H
#define TLISTITEM_H

#include <memory>
#include "octagon.h"
#include "square.h"
#include "triangle.h"

template <class T>
class TListItem
{
public:
    TListItem(const std::shared_ptr<T> &obj);

    std::shared_ptr<T> GetFigure() const;
    std::shared_ptr<TListItem<T>> GetNext();
    std::shared_ptr<TListItem<T>> GetPrev();
    void SetNext(std::shared_ptr<TListItem<T>> item);
    void SetPrev(std::shared_ptr<TListItem<T>> item);
    template <class A> friend std::ostream& operator<<(std::ostream &os,
const TListItem<A> &obj);

    virtual ~TListItem(){};

private:
    std::shared_ptr<T> item;
    std::shared_ptr<TListItem<T>> next;
    std::shared_ptr<TListItem<T>> prev;
};

#endif
```

TListItem.cpp

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

```
template <class T>
TListItem<T>::TListItem(const std::shared_ptr<T> &obj)
{
    this->item = obj;
    this->next = nullptr;
    this->prev = nullptr;
}

template <class T>
std::shared_ptr<T> TListItem<T>::GetFigure() const
{
    return this->item;
}

template <class T>
std::shared_ptr<TListItem<T>> TListItem<T>::GetNext()
{
    return this->next;
}

template <class T>
std::shared_ptr<TListItem<T>> TListItem<T>::GetPrev()
{
    return this->prev;
}

template <class T>
void TListItem<T>::SetNext(std::shared_ptr<TListItem<T>> item)
{
    this->next = item;
}

template <class T>
void TListItem<T>::SetPrev(std::shared_ptr<TListItem<T>> item)
{
    this->prev = item;
}

template <class T>
std::ostream& operator<<(std::ostream &os, const TListItem<T> &obj)
{
    os << obj.item << std::endl;
    return os;
}

#include "figure.h"
template class TListItem<Figure>;
template std::ostream& operator<<(std::ostream &out, const
TListItem<Figure> &obj);
```

Main.cpp

```
#include
<iostream>
#include <memory>
#include <cstdlib>
#include <cstring>
#include "octagon.h"
#include "square.h"
#include "triangle.h"
#include "TList.h"

void menu()
{
    std::cout << "Choose an operation:" << std::endl;
    std::cout << "1) Add triangle" << std::endl;
    std::cout << "2) Add square" << std::endl;
    std::cout << "3) Add octagon" << std::endl;
    std::cout << "4) Delete figure from list" << std::endl;
    std::cout << "5) Print list"  << std::endl;
    std::cout << "0) Exit" << std::endl;
}
```

```
int main(void)
{
    int32_t act = 0;
    TList<Figure> list;
    std::shared_ptr<Figure> ptr;
    do {
        menu();
        std::cin >> act;
        switch(act) {
            case 1:
                ptr = std::make_shared<Triangle>(std::cin);
                list.Push(ptr);
                break;
            case 2:
                ptr = std::make_shared<FSquare>(std::cin);
                list.Push(ptr);
                break;
            case 3:
                ptr = std::make_shared<Octagon>(std::cin);
                list.Push(ptr);
                break;
            case 4:
                list.Pop();
                break;
            case 5:
                std::cout << list << std::endl;
                break;
            case 0:
                list.Del();
                break;
            default:
                std::cout << "Incorrect command" << std::endl;;
                break;
        }
    } while (act);
    return 0;
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
       3. Add trapeze
       4. Add rhombus
       5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
2
5
Enter the side: 1
Correct value
Queue item: created
2
4
Enter the angle: 30
Enter the side: 2
Correct value
Queue item: created
2
3
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
Queue item: created
7
Pentagon:
Side: 1
```

```
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
Pentagon deleted
7
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
Rhombus deleted
7
Trapeze:
side 1:  3 side 2: 3 height: 3
6
7
0
Queue item: deleted
Trapeze deleted
```

ВЫВОД

В ходе 4 лабораторной работы я познакомился с шаблонами. Они упрощают и сокращают код программы, так как создаются для всех типов данных и делают программу(часть программы) универсальной.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ
Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 5

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№4) спроектировать и разработать Итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for. Например: for(auto i : stack) std::cout << *i << std::endl.

ОПИСАНИЕ

Для доступа к элементам некоторого множества элементов алгоритмы stl используют специальные объекты, называемые итераторами. В контейнерных типах stl они доступны через методы класса (например, begin() в шаблоне класса vector). Функциональные возможности указателей и итераторов близки, так что обычный указатель тоже может использоваться как итератор.
категории итераторов:

- итератор ввода (input iterator) - используется потоками ввода;
- итератор вывода (output iterator) - используется потоками вывода;
- однонаправленный итератор (forward iterator) - для прохода по элементам в одном направлении;
- двунаправленный итератор (bidirectional iterator) - способен пройтись по элементам в любом направлении. Такие итераторы реализованы в некоторых контейнерных типах stl (list, set, multiset, map, multimap);
- итераторы произвольного доступа (random access) - через них можно иметь доступ к любому элемента. Такие итераторы реализованы в некоторых контейнерных типах stl (vector, deque, string, array).

ЛИСТИНГ ПРОГРАММЫ

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
       3. Add trapeze
       4. Add rhombus
       5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
2
4
Enter the angle: 30
Enter the side: 1
Correct value
Queue item: created
2
3
Enter first side: 2
Enter second side: 2
Enter the height: 2
Correct value
Queue item: created
2
5
Enter the side: 3
```

```
Correct value
Queue item: created
7
Rhombus:
angle: 30 side: 1
Trapeze:
side 1:  2 side 2: 2 height: 2
Pentagon:
Side: 3
6
Queue item: deleted
Rhombus deleted
7
Trapeze:
side 1:  2 side 2: 2 height: 2
Pentagon:
Side: 3
6
Queue item: deleted
Trapeze deleted
7
Pentagon:
Side: 3
6
7
0
Queue item: deleted
Pentagon deleted
```

ВЫВОД

В 5 лабораторной работе мы добавляем итератор, что облегчает обращение к объектам и позволяет обращаться к ним по значениям.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 6

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛРN№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 варианта задания). Для вызова аллокатора должны быть переопределены оператор new и delete у классов-фигур.

ОПИСАНИЕ

Класс шаблона описывает объект, который управляет выделением и освобождением памяти для массивов объектов типа типа. Объект класса распределителя — объект распределителя по умолчанию, заданный в конструкторы несколько классов шаблонов контейнера из стандартной библиотеки C++.

Все контейнеры библиотеки стандартных шаблонов имеют параметр шаблона, который по умолчанию распределителя. Создание контейнера с пользовательским распределителем дает возможность управлять выделением и освобождением памяти для элементов контейнера.

Например, объект распределителя может выделить память в закрытой куче или в общей памяти. Он также может выполнить оптимизацию для крупных или мелких объектов. Он может также указывать, посредством определения типов, которые он предоставляет, что доступ к элементам возможен только через специальные объекты доступа, управляющие общей памятью или выполняющие автоматическую сборку мусора. Таким образом, класс, который выделяет память с использованием объекта распределителя, должен использовать эти типы для объявления указателя и объектов ссылок, как это делают контейнеры в стандартной библиотеке C++.

ЛИСТИНГ ПРОГРАММЫ

TList.h

```
#ifndef
TLIST_H
          #define TLIST_H

          #include <cstdint>
          #include "Triangle.h"
          #include "Foursquare.h"
          #include "Octagon.h"
          #include "TListItem.h"

          #include "TIterator.h"

          template <class T>
          class TList
          {
          public:
              TList();
              void Push(std::shared_ptr<T> &obj);
              const bool IsEmpty() const;
              uint32_t GetLength();
              std::shared_ptr<T> Pop();
              TIterator<TListItem<T>,T> begin();
              TIterator<TListItem<T>,T> end();

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

```
    virtual ~TList();

private:
    uint32_t length;
    std::shared_ptr<TListItem<T>> head;

    void PushFirst(std::shared_ptr<T> &obj);
    void PushLast(std::shared_ptr<T> &obj);
    void PushAtIndex(std::shared_ptr<T> &obj, int32_t ind);
    std::shared_ptr<T> PopFirst();
    std::shared_ptr<T> PopLast();
    std::shared_ptr<T> PopAtIndex(int32_t ind);
};

#endif
```

TList.cpp

```
#include "TList.h"

#include <iostream>
#include <cstdint>

template <class T>
TList<T>::TList()
{
    head = nullptr;
    length = 0;
}

template <class T>
void TList<T>::Push(std::shared_ptr<T> &obj) {
    int32_t index = 0;
    std::cout << "Enter index = ";
    std::cin >> index;
    if (index > this->GetLength()|| index < 0) {
        std::cerr << "This index doesn't exist\n";
        return;
    }
    std::shared_ptr<TListItem<T>> newItem =
std::make_shared<TListItem<T>>(obj);

    if (index == 0) {
        newItem->SetNext(head);
        head = newItem;
        ++length;
        return;
    } else {
        std::shared_ptr<TListItem<T>> cur = this->head;
        std::shared_ptr<TListItem<T>> prev = this->head;
        for (int i = 0; i < index; ++i){
            prev = cur;
            cur = cur->GetNext();
        }
        newItem->SetNext(cur);
        prev->SetNext(newItem);
    }
    ++length;
}

template <class T>
uint32_t TList<T>::GetLength()
{
    return this->length;
}

template <class T>
const bool TList<T>::IsEmpty() const
{
    return head == nullptr;
}


template <class T>
```

```
std::shared_ptr<T> TList<T>::Pop() {
    int32_t ind = 0;
    std::cout << "Enter index = ";
    std::cin >> ind;
    std::shared_ptr<T> res;
    if (ind > this->GetLength() - 1 || ind < 0 || this->IsEmpty()) {
        std::cout << "Change index" << std::endl;
        return res;
    }

    std::shared_ptr<TListItem<T>> tmp = this->head;
    for(int32_t i = 0; i < ind - 1; ++i) {
        tmp = tmp->GetNext();
    }
    std::shared_ptr<TListItem<T>> removed = tmp->GetNext();
    res = removed->GetFigure();
    std::shared_ptr<TListItem<T>> nextItem = removed->GetNext();
    tmp->SetNext(nextItem);
    //nextItem->SetPrev(tmp);
    //return res;
    --length;
    return res;
}



template <class T>
std::shared_ptr<T> TList<T>::PopFirst()
{
    if (this->GetLength() == 1) {
        std::shared_ptr<T> res = this->head->GetFigure();
        this->head = nullptr;
        return res;
    }
    std::shared_ptr<TListItem<T>> tmp = this->head;
    std::shared_ptr<T> res = tmp->GetFigure();
    this->head = this->head->GetNext();
    return res;
}


template <class T>
TIterator<TListItem<T>,T> TList<T>::begin()
{
    return TIterator<TListItem<T>,T>(head);
}

template <class T>
TIterator<TListItem<T>,T> TList<T>::end()
{
    return TIterator<TListItem<T>,T>(nullptr);
}



template <class T>
std::ostream& operator<<(std::ostream &os, const TList<T> &list)
{
    if (list.IsEmpty()) {
        os << "The list is empty." << std::endl;
        return os;
    }

    std::shared_ptr<TListItem<T>> tmp = list.head;
    for(int32_t i = 0; tmp; ++i) {
        os << "idx: " << i << "    ";
        tmp->GetFigure()->Print();
        os << std::endl;
        tmp = tmp->GetNext();
    }

    return os;
}
```

```
template <class T>
TList<T>::~TList()
{
    while(!this->IsEmpty()) {
        this->PopFirst();
        --length;
    }
}

#include "figure.h"
template class TList<Figure>;
template std::ostream& operator <<(std::ostream &out, const TList<Figure>
&obj);
```

TListItem.h

```
#ifndef
TLISTITEM_H
  #define TLISTITEM_H

  #include <memory>

  #include "Triangle.h"
  #include "Foursquare.h"
  #include "Octagon.h"

  #include "TAllocationBlock.h"

  template <class T>
  class TListItem
  {
  public:
      TListItem(const std::shared_ptr<T> &obj);

      std::shared_ptr<T> GetFigure() const;
      std::shared_ptr<TListItem<T>> GetNext();
      std::shared_ptr<T> remove();
      void SetNext(std::shared_ptr<TListItem<T>> item);
      template <class A> friend std::ostream& operator<<(std::ostream &os,
  const TListItem<A> &obj);

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

      virtual ~TListItem(){};

  private:
      std::shared_ptr<T> item;
      std::shared_ptr<TListItem<T>> next;

      static TAllocationBlock listitem_allocator;
  };

  #endif
```

TListItem.cpp

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

  template <class T>
  TListItem<T>::TListItem(const std::shared_ptr<T> &obj)
  {
      this->item = obj;
      this->next = nullptr;
  }


  template <class T> TAllocationBlock
  TListItem<T>::listitem_allocator(sizeof(TListItem<T>), 100);

  template <class T>
  std::shared_ptr<T> TListItem<T>::GetFigure() const
```

```
{
    return this->item;
}

template <class T>
std::shared_ptr<TListItem<T>> TListItem<T>::GetNext()
{
    return this->next;
}

template <class T>
void TListItem<T>::SetNext(std::shared_ptr<TListItem<T>> item)
{
    this->next = item;
}

template <class T>
std::shared_ptr<T> TListItem<T>::remove() {
  std::shared_ptr<TListItem<T>> removed = this->next;
  std::shared_ptr<T> item = removed->GetFigure();
  this->next = removed->GetNext();
  return item;
}

template <class T>
std::ostream& operator<<(std::ostream &os, const TListItem<T> &obj)
{
    os << obj.item << std::endl;
    return os;
}

template <class T>
void *TListItem<T>::operator new(size_t size)
{
    return listitem_allocator.Allocate();
}

template <class T>
void TListItem<T>::operator delete(void *ptr)
{
    listitem_allocator.Deallocate(ptr);
}

#include "figure.h"
template class TListItem<Figure>;
template std::ostream& operator <<(std::ostream &out, const
TListItem<Figure> &obj);
```

TAllocationBlock.h

```
#ifndef
TALLOCATIONBLOCK_H
     #defineTALLOCATIONBLOCK_H

     #include <cstdlib>

     class TAllocationBlock {
     public:
         TAllocationBlock(size_t size,size_t count);
         void *Allocate();
         void Deallocate(void *pointer);
         bool Empty();

         virtual ~TAllocationBlock();
     private:
         size_t _size;
         size_t _count;

         char  *_used_blocks;
         void  **_free_blocks;

         size_t _free_count;
```

```
};
#endif
```

TAllocationBlock.cpp

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

                        TAllocationBlock::TAllocationBlock(size_t size,size_t count):
                        _size(size),_count(count)  {
                            _used_blocks = (char*)malloc(_size*_count);
                            _free_blocks = (void**)malloc(sizeof(void*)*_count);

                            for(size_t i=0;i<_count;i++) _free_blocks[i] =
                        _used_blocks+i*_size;
                            _free_count = _count;
                              std::cout << "TAllocationBlock: Memory init" << std::endl;
                        }

                        void *TAllocationBlock::Allocate() {
                            void *result = nullptr;

                            if(_free_count>0)
                            {
                                result = _free_blocks[_free_count-1];
                                _free_count--;
                                std::cout << "TAllocationBlock: Allocate " << (_count-
                        _free_count) << " 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;

                          _free_blocks[_free_count] = pointer;
                          _free_count ++;

                        }

                        bool TAllocationBlock::Empty() {
                            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;
                           free(_free_blocks);
                           free(_used_blocks);
                        }
```

Main.cpp

```
#include
<iostream>
              #include <memory>
              #include <cstdlib>
              #include <cstring>
              #include "Triangle.h"
              #include "Foursquare.h"
              #include "Octagon.h"
              #include "TList.h"

              void menu()
              {
```

```
    std::cout << "Choose an operation:" << std::endl;
    std::cout << "1) Add triangle" << std::endl;
    std::cout << "2) Add foursquare" << std::endl;
    std::cout << "3) Add octagon" << std::endl;
    std::cout << "4) Delete figure from list" << std::endl;
    std::cout << "5) Print list"  << std::endl;
    std::cout << "6) Print list using iterator" << std::endl;
    std::cout << "0) Exit" << std::endl;
}

int main(void)
{
    int32_t act = 0;
    TList<Figure> list;
    std::shared_ptr<Figure> ptr;
    do {
        menu();
        std::cin >> act;
        switch(act) {
            case 1:
                ptr = std::make_shared<Triangle>(std::cin);
                list.Push(ptr);
                break;
            case 2:
                ptr = std::make_shared<Foursquare>(std::cin);
                list.Push(ptr);
                break;
            case 3:
                ptr = std::make_shared<Octagon>(std::cin);
                list.Push(ptr);
                break;
            case 4:
                ptr = list.Pop();
                break;
            case 5:
                std::cout << list << std::endl;
                break;
            case 6:
                if(!list.IsEmpty()) {
                     std::cout << "List is printed using iterator" <<
std::endl;
                    for(auto i : list) {
                        i->Print();
                    }
                } else {
                    std::cout << "List is empty." << std::endl;
                }
                break;

            case 0:
                break;
            default:
                std::cout << "Incorrect command" << std::endl;;
                break;
        }
    } while (act);
    return 0;
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
TAllocationBlock: Memory init
1. Create queue
2. Add item
       3. Add trapeze
       4. Add rhombus
       5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
```

```
2
5
Enter the side: 1
Correct value
TAllocationBlock: Allocate 1 of 100
Queue item: created
2
4
Enter the angle: 30
Enter the side: 2
Correct value
TAllocationBlock: Allocate 2 of 100
Queue item: created
2
3
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
TAllocationBlock: Allocate 3 of 100
Queue item: created
7
Pentagon:
Side: 1
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
TAllocationBlock: Deallocate block
0xbb3410
Pentagon deleted
7
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
TAllocationBlock: Deallocate block
0xbb34c0
Rhombus deleted
7
Trapeze:
side 1:  3 side 2: 3 height: 3
0
Queue item: deleted
Trapeze deleted
TAllocationBlock: Deallocate block
TAllocationBlock: Memory freed
```

ВЫВОД

В ходе 6 работы был я ознакомился и создал аллокатор памяти, который помогает оптимизировать выделение и освобождение памяти. Свободные блоки хранятся в аллокаторе в контейнере второго уровня, который представляет собой список.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 7

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер (Контейнер 1-го уровня) – очередь.

Каждым элементом контейнера, в свою очередь, является динамической структурой данных одного из следующих видов (Контейнер 2-го уровня) – N-дерево.

ОПИСАНИЕ

Принцип открытости/закрытости (Open/Closed Principle) можно сформулировать так:

Сущности программы должны быть открыты для расширения, но закрыты для изменения.

Суть этого принципа состоит в том, что система должна быть построена таким образом, что все ее последующие изменения должны быть реализованы с помощью добавления нового кода, а не изменения уже существующего.

Противоречие является лишь кажущимся, поскольку термины соответствуют разным целевым установкам:

- Модуль называют открытым, если он еще доступен для расширения. Например, имеется возможность расширить множество операций в нем или добавить поля к его структурам данных.
- Модуль называют закрытым, если он доступен для использования другими модулями. Это означает, что модуль (его интерфейс — с точки зрения скрытия информации) уже имеет строго определенное окончательное описание. На уровне реализации закрытое состояние модуля означает, что модуль можно компилировать, сохранять в библиотеке и делать его доступным для использования другими модулями (его клиентами). На этапе проектирования или спецификации закрытие модуля означает, что он одобрен руководством, внесен в официальный репозиторий утвержденных программных элементов проекта — базу проекта (project baseline), и его интерфейс опубликован в интересах авторов других модулей.

ЛИСТИНГ ПРОГРАММЫ

TList.h

```
#ifndef TLIST_H
#define TLIST_H
#include <memory>
#include <iostream>
#include "TAllocator.h"
#include <future>
#include <thread>
#include <functional>

template <typename T> class TList
{
private:
    class TNode {
    public:
        TNode();
        TNode(const std::shared_ptr<T>&);
        auto GetNext() const;
        auto GetItem() const;
        std::shared_ptr<T> item;
        std::shared_ptr<TNode> next;
```

```
        void* operator new(size_t);
        void operator delete(void*);
        static TAllocator nodeAllocator;
    };

    template <typename N, typename M>
    class TIterator {
    private:
        N nodePtr;
    public:
        TIterator(const N&);
        std::shared_ptr<M> operator* ();
        std::shared_ptr<M> operator-> ();
        void operator ++ ();
        bool operator == (const TIterator&);
        bool operator != (const TIterator&);
    };

    int length;

    std::shared_ptr<TNode> head;
    auto psort(std::shared_ptr<TNode>&);
    auto pparsort(std::shared_ptr<TNode>& head);
    auto partition(std::shared_ptr<TNode>&);

public:
    TList();
    bool PushFront(const std::shared_ptr<T>&);
    bool Push(const std::shared_ptr<T>&, const int);
    bool PopFront();
    bool Pop(const int);
    bool IsEmpty() const;
    int GetLength() const;
    auto& getHead();
    auto&& getTail();
    void sort();
    void parSort();

    TIterator<std::shared_ptr<TNode>, T> begin() {return
TIterator<std::shared_ptr<TNode>, T>(head->next);};
    TIterator<std::shared_ptr<TNode>, T> end() {return
TIterator<std::shared_ptr<TNode>, T>(nullptr);};

    template <typename A> friend std::ostream& operator<< (std::ostream&,
TList<A>&);
};

#include "TList.hpp"
#include "TIterator.hpp"
#endif
```

TList.hpp

```
#ifdef
TLIST_H

template <typename T> TList<T>::TNode::TNode()
{
    item = std::shared_ptr<T>();
    next = nullptr;
}

template <typename T> TList<T>::TNode::TNode(const std::shared_ptr<T>& obj)
{
    item = obj;
    next = nullptr;
}

template <typename T> TAllocator
TList<T>::TNode::nodeAllocator(sizeof(TList<T>::TNode), 100);

template <typename T> void* TList<T>::TNode::operator new(size_t size)
{
    return nodeAllocator.allocate();
}
```

```
template <typename T> void TList<T>::TNode::operator delete(void* ptr)
{
    nodeAllocator.deallocate(ptr);
}

template <typename T> TList<T>::TList()
{
    head = std::make_shared<TNode>();
    length = 0;
}

template <typename T> bool TList<T>::IsEmpty() const
{
    return this->length == 0;
}

template <typename T> auto& TList<T>::getHead()
{
    return this->head->next;
}

template <typename T> auto&& TList<T>::getTail()
{
    auto tail = head->next;
    while (tail->next != nullptr) {
        tail = tail->next;
    }

    return tail;
}

template <typename T> int TList<T>::GetLength() const
{
    return this->length;
}

template <typename T> bool TList<T>::PushFront(const std::shared_ptr<T>& obj)
{
    auto Nitem = std::make_shared<TNode>(obj);
    std::swap(Nitem->next, head->next);
    std::swap(head->next, Nitem);
    length++;

    return true;
}

template <typename T> bool TList<T>::Push(const std::shared_ptr<T>& obj, int pos)
{
    if (pos == 1 || length == 0)
        return PushFront(obj);
    if (pos < 0 || pos > length + 1)
        return false;

    auto iter = head->next;
    int i = 0;

    while (i < pos - 2) {
        iter = iter->next;
        i++;
    }

    auto Nitem = std::make_shared<TNode>(obj);
    std::swap(Nitem->next, iter->next);
    std::swap(iter->next, Nitem);
    length++;

    return true;
}

template <typename T> bool TList<T>::PopFront()
{
    if (IsEmpty())
```

```
        return false;

    head->next = std::move(head->next->next);

    length--;

    return true;
}

template <typename T> bool TList<T>::Pop(int pos)
{
    if (pos < 1 || pos > length || IsEmpty())
        return false;
    if (pos == 1)
        return PopFront();

    auto iter = head->next;
    int i = 0;

    while (i < pos - 2) {
        iter = iter->next;
        i++;
    }

    iter->next = std::move(iter->next->next);
    length--;

    return true;
}

template <typename T> auto TList<T>::TNode::GetNext() const
{
    return this->next;
}

template <typename T> auto TList<T>::TNode::GetItem() const
{
    return this->item;
}

template <typename A> std::ostream& operator<< (std::ostream& os, const TList<A>&
list)
{
    if (list.IsEmpty()) {
        os << "The list is empty!" << std::endl;
        return os;
    }

    auto tmp = list.head->GetNext();
    while(tmp != nullptr) {
        tmp->GetItem()->Print();
        tmp = tmp->GetNext();
    }

    return os;
}

template <typename T> auto TList<T>::psort(std::shared_ptr<TNode>& head)
{
    if (head == nullptr || head->next == nullptr) {
        return head;
    }

    auto partitionedEl = partition(head);
    auto leftPartition = partitionedEl->next;
    auto rightPartition = head;

    partitionedEl->next = nullptr;

    if (leftPartition == nullptr) {
        leftPartition = head;
        rightPartition = head->next;
        head->next = nullptr;
```

```
    }

    rightPartition = psort(rightPartition);
    leftPartition = psort(leftPartition);
    auto iter = leftPartition;
    while (iter->next != nullptr) {
        iter = iter->next;
    }

    iter->next = rightPartition;

    return leftPartition;
}

template <typename T> auto TList<T>::partition(std::shared_ptr<TNode>& head)
{
    if (head->next->next == nullptr) {
        if (head->next->GetItem()->Square() > head->GetItem()->Square()) {
            return head->next;
        } else {
            return head;
        }
    } else {
        auto i = head->next;
        auto pivot = head;
        auto lastElSwapped = (pivot->next->GetItem()->Square()
                              >= pivot->GetItem()->Square()) ? pivot->next : pivot;

        while ((i != nullptr) && (i->next != nullptr)) {
            if (i->next->GetItem()->Square() >= pivot->GetItem()->Square()) {
                if (i->next == lastElSwapped->next) {
                    lastElSwapped = lastElSwapped->next;
                } else {
                    auto tmp = lastElSwapped->next;
                    lastElSwapped->next = i->next;
                    i->next = i->next->next;
                    lastElSwapped = lastElSwapped->next;
                    lastElSwapped->next = tmp;
                }
            }

            i = i->next;
        }

        return lastElSwapped;
    }

}

template <typename T> void TList<T>::sort()
{
    head->next = psort(head->next);
}

template <typename T> void TList<T>::parSort()
{
    head->next = pparsort(head->next);
}

template <typename T> auto TList<T>::pparsort(std::shared_ptr<TNode>& head)
{
    if (head == nullptr || head->next == nullptr) {
        return head;
    }

    auto partitionedEl = partition(head);
    auto leftPartition = partitionedEl->next;
    auto rightPartition = head;

    partitionedEl->next = nullptr;

    if (leftPartition == nullptr) {
        leftPartition = head;
```

```
            rightPartition = head->next;
            head->next = nullptr;
        }

        std::packaged_task<std::shared_ptr<TNode>(std::shared_ptr<TNode>&)>
            task1(std::bind(&TList<T>::pparsort, this, std::placeholders::_1));
        std::packaged_task<std::shared_ptr<TNode>(std::shared_ptr<TNode>&)>
            task2(std::bind(&TList<T>::pparsort, this, std::placeholders::_1));
        auto rightPartitionHandle = task1.get_future();
        auto leftPartitionHandle = task2.get_future();


        std::thread(std::move(task1), std::ref(rightPartition)).join();
        rightPartition = rightPartitionHandle.get();
        std::thread(std::move(task2), std::ref(leftPartition)).join();
        leftPartition = leftPartitionHandle.get();
        auto iter = leftPartition;
        while (iter->next != nullptr) {
            iter = iter->next;
        }

        iter->next = rightPartition;

        return leftPartition;
    }


    #endif
```

TNList.h

```
#ifndef
TNLIST_H
 #define TNLIST_H

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

 template <typename Q, typename O> class TNList
 {
 private:
     class Node {
     public:
         Q data;
         std::shared_ptr<Node> next;
         Node();
         Node(const O&);
         int itemsInNode;
     };

     std::shared_ptr<Node> head;
     int count;
 public:
     TNList();

     void push(const O&);
     void print();
     void removeByType(const int&);
     void removeLesser(const double&);
 };

 #include "TNList.hpp"
 #endif
```

TNList.hpp

```
#ifdef
TLIST_H

template <typename Q, typename O> TNList<Q, O>::TNList()
{
    head = std::make_shared<Node>(Node());
    count = 0;
}
```

```
template <typename Q, typename O> TNList<Q, O>::Node::Node()
{
    next = nullptr;
    itemsInNode = 0;
}

template <typename Q, typename O> TNList<Q, O>::Node::Node(const O& item)
{
    data.PushFront(item);
    itemsInNode = 1;
}

template <typename Q, typename O> void TNList<Q, O>::removeByType(const int&
type)
{
    auto tmp = head;
    while(tmp) {
        if (tmp->itemsInNode) {
            for (int i = 0; i < 5; ++i) {
                auto iter = tmp->data.begin();

                for (int k = 0; k < tmp->data.GetLength(); ++k) {
                    if (iter->Type() == type) {
                        tmp->data.Pop(k + 1);
                        tmp->itemsInNode--;
                        break;
                    }
                    ++iter;
                }
            }
        }
        tmp = tmp->next;
    }
}

template <typename Q, typename O> void TNList<Q, O>::push(const O& item)
{
    auto tmp = this->head;
    while(tmp->next) {
       tmp = tmp->next;
    }
    if (tmp->itemsInNode < 5) {
        tmp->data.PushFront(item);
        tmp->itemsInNode++;
    } else {
        auto newNode = std::make_shared<Node>(Node(item));
        tmp->next = newNode;
        ++count;
    }
}


template <typename Q, typename O> void TNList<Q, O>::print()
{
    auto tmp = head;
    while (tmp) {
        if (tmp->itemsInNode) {
            tmp->data.sort();
            for (const auto &i: tmp->data) {
                i->Print();
            }
            std::cout << std::endl;
        }
        tmp = tmp->next;
    }
}


template <typename Q, typename O> void TNList<Q, O>::removeLesser(const double&
sqr)
{
    auto tmp = head;
    while(tmp) {
```

```
            if (tmp->itemsInNode) {
                for (int i = 0; i < 5; ++i) {
                    auto iter = tmp->data.begin();

                    for (int k = 0; k < tmp->data.GetLength(); ++k) {
                        if (iter->Square() < sqr) {
                            tmp->data.Pop(k + 1);
                            tmp->itemsInNode--;
                            break;
                        }
                        ++iter;
                    }
                }
            }
            tmp = tmp->next;
        }
    }

    #endif
```

Main.cpp

```
#include "TList.h"
    #include <iostream>
    #include "Foursquare.h"
    #include "Octagon.h"
    #include "Triangle.h"
    #include "TNList.h"

    void menu(void)
    {
        std::cout << "Choose an operation:" << std::endl;
        std::cout << "1) Add Triangle" << std::endl;
        std::cout << "2) Add Foursquare" << std::endl;
        std::cout << "3) Add Octagon" << std::endl;
        std::cout << "4) Delete Figure" << std::endl;
        std::cout << "5) Print"  << std::endl;
        std::cout << "0) Exit" << std::endl;
    }

    int main(void)
    {
        TNList<TList<Figure>, std::shared_ptr<Figure> > list;
        int act, index;
        Triangle tmp1;
        Foursquare tmp2;
        Octagon tmp3;

        do {
            menu();
            std::cin >> act;
            switch(act) {
                case 1:
                    list.push(std::make_shared<Triangle>(std::cin));
                    std::cout << "Item was added" << std::endl;
                    break;
                case 2:
                    list.push(std::make_shared<Foursquare>(std::cin));
                    std::cout << "Item was added" << std::endl;
                    break;
                case 3:
                    list.push(std::make_shared<Octagon>(std::cin));
                    std::cout << "Item was added" << std::endl;
                    break;
                case 4:{
                    std::cout << "Enter criteria" << std::endl;
                    std::cout << "1) by type\n2) lesser than square\n";
                    std::cin >> index;
                    if (index == 1) {
                        std::cout << "1) Foursquare\n2) Octagon\n3) Triangle\n";
                        std::cout << "Enter type" << std::endl;
```

```
                            std::cin >> index;
                            list.removeByType(index);
                        } else if (index == 2) {
                            double sqr = 0.0;
                            std::cout << "Enter square" << std::endl;
                            std::cin >> sqr;
                            list.removeLesser(sqr);
                        } else {
                            break;
                        }
                        break;
                    }
                    case 5:
                        list.print();
                        break;
                    case 0:
                        break;
                    default:
                        std::cout << "Incorrect command" << std::endl;;
                        break;
                }

            } while(act);

            return 0;
        }
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
TAllocationBlock: Memory init
TAllocationBlock: Memory init
1. Create queue
2. Add item
  3. Add trapeze
  4. Add rhombus
  5. Add pentagon
6. Delete item
7. Print queue
8. Menu
9. Create Tree
10. Add Tree
0. Exit

9
  1 Trapeze
  2 Rhombus
  3 Pentagon
3
Enter the side: 1
Correct value
Tree created
2
3
Enter path:
L
Enter first side: 2
Enter second side: 2
Enter the height: 2
Correct value
2
3
Enter path:
L
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
2
4
Enter path:
S
Enter the angle: 30
Enter the side: 4
Correct value
2
```

```
5
Enter path:
SB
Enter the side: 5
Correct value
1
Queue created
10
TAllocationBlock: Allocate 1 of 100
Queue item: created
9
  1 Trapeze
  2 Rhombus
  3 Pentagon
1
Enter first side: 1
Enter second side: 1
Enter the height: 1
Correct value
Tree created
2
4
Enter path:
L
Enter the angle: 30
Enter the side: 2
Correct value
2
5
Enter path:
L
Enter the side: 3
Correct value
2
4
Enter path:
SB
Enter the angle: 30
Enter the side: 4
Correct value
2
5
Enter path:
SB
Enter the side: 5
Correct value
10
TAllocationBlock: Allocate 2 of 100
Queue item: created
7

Pentagon:
Side: 1

Trapeze:
side 1:  2 side 2: 2 height: 2

 Rhombus:
angle: 30 side: 4

Trapeze:
side 1:  3 side 2: 3 height: 3

Pentagon:
Side: 5

Trapeze:
side 1:  1 side 2: 1 height: 1

Rhombus:
angle: 30 side: 2

Pentagon:
Side: 3

 Rhombus:
```

```
angle: 30 side: 4

Pentagon:
Side: 5
6
Queue item: deleted
TAllocationBlock: Deallocate block

Pentagon:
Side: 1

Trapeze:
side 1:  2 side 2: 2 height: 2

 Rhombus:
angle: 30 side: 4

Trapeze:
side 1:  3 side 2: 3 height: 3

Pentagon:
Side: 5
Pentagon deleted
Trapeze deleted
Rhombus deleted
Trapeze deleted
Pentagon deleted
7
Trapeze:
side 1:  1 side 2: 1 height: 1

Rhombus:
angle: 30 side: 2

Pentagon:
Side: 3

 Rhombus:
angle: 30 side: 4

Pentagon:
Side: 5

0
Queue item: deleted
Pentagon deleted
Rhombus deleted
Pentagon deleted
Rhombus deleted
Trapeze deleted
TAllocationBlock: Deallocate block
TAllocationBlock: Memory freed
TAllocationBlock: Memory freed
```

ВЫВОД

В седьмой лабораторной работе создается программа, которая позволяет хранить контейнер в контейнере, в котором хранятся фигуры. При этом фигуры должны быть отсортированы по площади. Если контейнер второго уровня освобождается, то он удаляется. Эта лабораторная работа показалась мне наиболее сложной из тех, которые мне приходилось выполнять по ООП, но во время её выполнения я узнал очень много нового.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ
Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 8

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера .

Необходимо разработать два вида алгоритма:

- Обычный, без параллельных вызовов.
- С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков использовать механизмы:

- Future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

ОПИСАНИЕ

Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров. Параллельное программирование может быть сложным, которое включает в себя черты более традиционного или последовательного программирования, но в параллельном имеются три дополнительных этапа:

- определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.
- Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять задачи.
- Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

ЛИСТИНГ ПРОГРАММЫ

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H

#include <memory>
#include <iostream>

template <class N, class T>
class TIterator
{
public:
    TIterator(std::shared_ptr<N> n) {
        cur = n;
    }

    std::shared_ptr<T> operator* () {
        return cur->GetFigure();
    }

    std::shared_ptr<T> operator-> () {
        return cur->GetFigure();
    }

    void operator++() {
        cur = cur->GetNext();
    }
```

```
        TIterator operator++ (int) {
            TIterator cur(*this);
            ++(*this);
            return cur;
        }

        bool operator== (const TIterator &i) {
            return (cur == i.cur);
        }

        bool operator!= (const TIterator &i) {
            return (cur != i.cur);
        }

    private:
        std::shared_ptr<N> cur;
    };

    #endif
```

TList.h

```
#ifndef
TLIST_H
#define TLIST_H

#include <cstdint>
#include <future>
#include <mutex>

#include "Triangle.h"
#include "Foursquare.h"
#include "Octagon.h"

#include "TListItem.h"

#include "TIterator.h"

template <class T>
class TList
{
public:
    TList();
    void Push(std::shared_ptr<T> &obj);
    const bool IsEmpty() const;
    uint32_t GetLength();
    std::shared_ptr<T> Pop();
    //void Del();

    TIterator<TListItem<T>,T> begin();
    TIterator<TListItem<T>,T> end();

    void Sort();
    void ParSort();

    template <class A> friend std::ostream& operator<<(std::ostream &os, const
TList<A> &list);
    virtual ~TList();

private:
    uint32_t length;
    std::shared_ptr<TListItem<T>> head;
    std::shared_ptr<TListItem<T>> PSort(std::shared_ptr <TListItem<T>>&);
    std::shared_ptr<TListItem<T>> PParSort(std::shared_ptr<TListItem<T>> &head);
    std::shared_ptr<TListItem<T>> Partition(std::shared_ptr<TListItem<T>>&);
    std::mutex mutex;

    void PushFirst(std::shared_ptr<T> &obj);
    void PushLast(std::shared_ptr<T> &obj);
    void PushAtIndex(std::shared_ptr<T> &obj, int32_t ind);
    std::shared_ptr<T> PopFirst();
    std::shared_ptr<T> PopLast();
    std::shared_ptr<T> PopAtIndex(int32_t ind);
};
```

```
#endif
```

TList.cpp

```
#include "TList.h"

#include <iostream>
#include <cstdint>

template <class T>
TList<T>::TList()
{
    head = std::make_shared<TListItem<T>>();
    length = 0;
}

template <class T>
void TList<T>::Push(std::shared_ptr<T> &obj) {
    if (this->IsEmpty()) {
        std::shared_ptr<TListItem<T>> newItem =
std::make_shared<TListItem<T>>(obj);
        this->head->SetNext(newItem);
        length+=1;
        return;
    }
    std::shared_ptr<TListItem<T>> newItem =
std::make_shared<TListItem<T>>(obj);
    std::shared_ptr<TListItem<T>> tmp = this->head->GetNext();

    while (tmp->GetNext() != nullptr) {
        tmp = tmp->GetNext();
    }
    tmp->SetNext(newItem);

    length+=1;
}

template <class T>
uint32_t TList<T>::GetLength()
{
    return this->length;
}

template <class T>
const bool TList<T>::IsEmpty() const
{
    return this->length == 0;
}

template <class T>
std::shared_ptr<T> TList<T>::Pop() {
    if (this->IsEmpty()) {
        std::cout << "The list is empty" << std::endl;
        return nullptr;
    }
    if (this->GetLength() == 1) {
        std::shared_ptr<T> res = this->head->GetNext()->GetFigure();
        this->head->SetNext(nullptr);
        this->length--;
        return res;
    }

    std::shared_ptr<TListItem<T>> tmp = this->head->GetNext();
    while (tmp->GetNext()->GetNext()) {
        tmp = tmp->GetNext();
    }

    std::shared_ptr<TListItem<T>> removed = tmp->GetNext();
    std::shared_ptr<T> res = removed->GetFigure();
    tmp->SetNext(removed->GetNext());
    this->length--;
    return res;
}
```

```
template <class T>
std::ostream& operator<<(std::ostream &os, const TList<T> &list)
{
    if (list.IsEmpty()) {
        os << "The list is empty." << std::endl;
        return os;
    }

    std::shared_ptr<TListItem<T>> tmp = list.head->GetNext();
    for(int32_t i = 0; tmp; ++i) {
        tmp->GetFigure()->Print();
        os << std::endl;
        tmp = tmp->GetNext();
    }

    return os;
}


template <class T>
TIterator<TListItem<T>,T> TList<T>::begin()
{
    return TIterator<TListItem<T>,T>(head->GetNext());
}

template <class T>
TIterator<TListItem<T>,T> TList<T>::end()
{
    return TIterator<TListItem<T>,T>(nullptr);
}

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

template <class T>
std::shared_ptr<TListItem<T>> TList<T>::PSort(std::shared_ptr<TListItem<T>>
&head)
{
    if (head == nullptr || head->GetNext() == nullptr) {
        return head;
    }

    std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
    std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
    std::shared_ptr<TListItem<T>> rightPartition = head;

    partitionedEl->SetNext(nullptr);

    if (leftPartition == nullptr) {
        leftPartition = head;
        rightPartition = head->GetNext();
        head->SetNext(nullptr);
    }

    rightPartition = PSort(rightPartition);
    leftPartition = PSort(leftPartition);
    std::shared_ptr<TListItem<T>> iter = leftPartition;
    while (iter->GetNext() != nullptr) {
        iter = iter->GetNext();
    }

    iter->SetNext(rightPartition);

    return leftPartition;
}

template <class T>
std::shared_ptr<TListItem<T>> TList<T>::Partition(std::shared_ptr<TListItem<T>>
&head)
```

```
{
    std::lock_guard<std::mutex> lock(mutex);
    if (head->GetNext()->GetNext() = nullptr) {
        if (head->GetNext()->GetFigure()->Square() > head->GetFigure()-
>Square()) {
            return head->GetNext();
        } else {
            return head;
        }
    } else {
        std::shared_ptr<TListItem<T>> i = head->GetNext();
        std::shared_ptr<TListItem<T>> pivot = head;
        std::shared_ptr<TListItem<T>> lastElSwapped = (pivot->GetNext()-
>GetFigure()->Square() >= pivot->GetFigure()->Square()) ? pivot->GetNext() :
pivot;

        while ((i != nullptr) && (i->GetNext() != nullptr)) {
            if (i->GetNext()->GetFigure()->Square() >= pivot->GetFigure()-
>Square()) {
                if (i->GetNext() == lastElSwapped->GetNext()) {
                    lastElSwapped = lastElSwapped->GetNext();
                } else {
                    std::shared_ptr<TListItem<T>> tmp = lastElSwapped-
>GetNext();
                    lastElSwapped->SetNext(i->GetNext());
                    i->SetNext(i->GetNext()->GetNext());
                    lastElSwapped = lastElSwapped->GetNext();
                    lastElSwapped->SetNext(tmp);
                }
            }
            i = i->GetNext();
        }
        return lastElSwapped;
    }
}

template <class T>
void TList<T>::Sort()
{
    if (head == nullptr)
        return;
    std::shared_ptr<TListItem<T>> tmp = head->GetNext();
    head->SetNext(PSort(tmp));
}

template <class T>
void TList<T>::ParSort()
{
    if (head == nullptr)
        return;
    std::shared_ptr<TListItem<T>> tmp = head->GetNext();
    head->SetNext(PParSort(tmp));
}

template <class T>
std::shared_ptr<TListItem<T>> TList<T>::PParSort(std::shared_ptr<TListItem<T>>
&head)
{
    if (head == nullptr || head->GetNext() == nullptr) {
        return head;
    }

    std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
    std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
    std::shared_ptr<TListItem<T>> rightPartition = head;

    partitionedEl->SetNext(nullptr);

    if (leftPartition == nullptr) {
        leftPartition = head;
        rightPartition = head->GetNext();
        head->SetNext(nullptr);
    }
```

```
std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&
)>
        task1(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));

std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&
)>
        task2(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
    auto rightPartitionHandle = task1.get_future();
    auto leftPartitionHandle = task2.get_future();

    std::thread(std::move(task1), std::ref(rightPartition)).join();
    rightPartition = rightPartitionHandle.get();
    std::thread(std::move(task2), std::ref(leftPartition)).join();
    leftPartition = leftPartitionHandle.get();
    std::shared_ptr<TListItem<T>> iter = leftPartition;
    while (iter->GetNext() != nullptr) {
        iter = iter->GetNext();
    }

    iter->SetNext(rightPartition);
    return leftPartition;
}

#include "figure.h"
template class TList<Figure>;
template std::ostream& operator<<(std::ostream &out, const TList<Figure> &obj);
```

TListItem.h

```
#ifndef
TLISTITEM_H
  #define TLISTITEM_H

  #include <memory>

  #include "Triangle.h"
  #include "Foursquare.h"
  #include "Octagon.h"

  #include "TAllocationBlock.h"

  template <class T>
  class TListItem
  {
  public:
      TListItem(const std::shared_ptr<T> &obj);
      TListItem();

      std::shared_ptr<T> GetFigure() const;
      std::shared_ptr<TListItem<T>> GetNext();
      std::shared_ptr<T> remove();
      void SetNext(std::shared_ptr<TListItem<T>> item);
      template <class A> friend std::ostream& operator<<(std::ostream &os,
  const TListItem<A> &obj);

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

      virtual ~TListItem(){};

  private:
      std::shared_ptr<T> item;
      std::shared_ptr<TListItem<T>> next;

      static TAllocationBlock listitem_allocator;
  };

  #endif
```

TListItem.cpp

```
#include
"TListItem.h"
```

```
#include <iostream>

template <class T>
TListItem<T>::TListItem(const std::shared_ptr<T> &obj)
{
    this->item = obj;
    this->next = nullptr;
}

template <class T>
TListItem<T>::TListItem::TListItem()
{
    item = std::shared_ptr<T>();
    next = nullptr;
}


template <class T> TAllocationBlock
TListItem<T>::listitem_allocator(sizeof(TListItem<T>), 100);

template <class T>
std::shared_ptr<T> TListItem<T>::GetFigure() const
{
    return this->item;
}

template <class T>
std::shared_ptr<TListItem<T>> TListItem<T>::GetNext()
{
    return this->next;
}

template <class T>
void TListItem<T>::SetNext(std::shared_ptr<TListItem<T>> item)
{
    this->next = item;
}

template <class T>
std::shared_ptr<T> TListItem<T>::remove() {
  std::shared_ptr<TListItem<T>> removed = this->next;
  std::shared_ptr<T> item = removed->GetFigure();
  this->next = removed->GetNext();
  return item;
}

template <class T>
std::ostream& operator<<(std::ostream &os, const TListItem<T> &obj)
{
    os << obj.item << std::endl;
    return os;
}

template <class T>
void *TListItem<T>::operator new(size_t size)
{
    return listitem_allocator.Allocate();
}

template <class T>
void TListItem<T>::operator delete(void *ptr)
{
    listitem_allocator.Deallocate(ptr);
}

#include "figure.h"
template class TListItem<Figure>;
template std::ostream& operator <<(std::ostream &out, const
TListItem<Figure> &obj);
```

Main.cpp

```
#include
<iostream>
            #include <memory>
```

```cpp
#include <cstdlib>
#include <cstring>
#include "Triangle.h"
#include "Foursquare.h"
#include "Octagon.h"
#include "TList.h"

void menu()
{
    std::cout << "Choose an operation:" << std::endl;
    std::cout << "1) Add triangle" << std::endl;
    std::cout << "2) Add foursquare" << std::endl;
    std::cout << "3) Add octagon" << std::endl;
    std::cout << "4) Delete figure from list" << std::endl;
    std::cout << "5) Print list"  << std::endl;
    std::cout << "6) Sort list by qsort" << std::endl;
    std::cout << "0) Exit" << std::endl;
}

int main(void)
{
    int32_t act = 0;
    TList<Figure> list;
    std::shared_ptr<Figure> ptr;
    do {
        menu();
        std::cin >> act;
        switch(act) {
            case 1:
                ptr = std::make_shared<Triangle>(std::cin);
                list.Push(ptr);
                break;
            case 2:
                ptr = std::make_shared<Foursquare>(std::cin);
                list.Push(ptr);
                break;
            case 3:
                ptr = std::make_shared<Octagon>(std::cin);
                list.Push(ptr);
                break;
            case 4:
                ptr = list.Pop();
                break;
            case 5:
                std::cout << list << std::endl;
                break;
            case 9:
                for(auto i : list) {
                    i->Print();
                }
                break;
            case 6:
                std::cout << "1) using regular sort" << std::endl;
                std::cout << "2) to parallel" << std::endl;
                std::cin >> act;
                if (act == 1) {
                    list.Sort();
                } else if (act == 2) {
                    list.ParSort();
                } else {
                    std::cout << "Unknown command" << std::endl;
                    break;
                }
                break;
            case 0:
                break;
            default:
                std::cout << "Incorrect command" << std::endl;;
                break;
        }
    } while (act);
    return 0;
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
3
Queue item: created
Queue item: created
Queue item: created
Queue item: created
Queue item: created
Queue item: created
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Rhombus:
angle: 80 side: 9
79.7694
Rhombus:
angle: 58 side: 58
2852.83
Rhombus:
angle: 31 side: 80
3296.24
Sort:0
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Rhombus:
angle: 80 side: 9
79.7694
Rhombus:
angle: 58 side: 58
2852.83
Rhombus:
angle: 31 side: 80
3296.24
ParallelSort:0
Queue item: deleted
Queue item: deleted
Queue item: deleted
Rhombus deleted
Rhombus deleted
Rhombus deleted
Queue item: deleted
Queue item: deleted
Queue item: deleted
Rhombus deleted
Rhombus deleted
Rhombus deleted
1. Push rhombus in queue
```

```
2. Push pentagon in queue
3. Push trapeze in queue
4. Pop element
5. Print queue
6. Menu
7. Exit
2
Enter the side: 1
Correct value
Queue item: created
1
Enter the angle: 20
Enter the side: 2
Correct value
Queue item: created
5
Pentagon:
Side: 1
Rhombus:
angle: 20 side: 2
3
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
Queue item: created
5
Pentagon:
Side: 1
Rhombus:
angle: 20 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
4
Queue item: deleted
Pentagon:
Side: 1
Pentagon deleted
5
Rhombus:
angle: 20 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
4
Queue item: deleted
Rhombus:
angle: 20 side: 2
Rhombus deleted
4
Queue item: deleted
Trapeze:
side 1:  3 side 2: 3 height: 3
Trapeze deleted
7
```

ВЫВОД

В ходе лабораторной работы я изучил параллельное программирование, которое позволяет осуществлять сортировку иным способом, который работает быстрее. То есть, все рекурсивные вызовы осуществяются в разных потоках. Для этой лабораторной работы очень пригодились знания полученные в курсе ОС.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики и программирования

Отчет по лабораторной работе № 9

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Захаров И. С.
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

Подпись студента

Число
21.10.2018

Москва-2018

Задание

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер первого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованием шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контенйром 1-го уровня:
  - Генерация фигур со случайным значением параметров;
  - Печать контейнера на экран;
  - Удаление элементов со значением площади меньше определенного числа;
- В контенер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контенере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

```
#include
<iostream>
                #include <memory>
                #include <cstdlib>
                #include <cstring>
                #include <random>
                #include "Triangle.h"
                #include "Foursquare.h"
                #include "Octagon.h"
                #include "TList.h"
                #include "TNList.h"


                int main(void)
                {
                   TList<Figure> list;
                   typedef std::function<void(void)> Command;
                   TNList<std::shared_ptr<Command>> nlist;
                   std::mutex mtx;

                   Command cmdInsert = [&]() {
                      std::lock_guard<std::mutex> guard(mtx);

                      uint32_t seed =
              std::chrono::system_clock::now().time_since_epoch().count();

                      std::default_random_engine generator(seed);
                      std::uniform_int_distribution<int> distFigureType(1, 3);
                      std::uniform_int_distribution<int> distFigureParam(1, 10);
                      for (int i = 0; i < 5; ++ i) {
                         std::cout << "Command: Insert" << std::endl;
```

```
            switch(distFigureType(generator)) {
                case 1: {
                    std::cout << "Inserted Triangle" << std::endl;

                    int side_a = distFigureParam(generator);
                    int side_b = distFigureParam(generator);
                    int side_c = distFigureParam(generator);

                    std::shared_ptr<Figure> ptr =
std::make_shared<Triangle>(Triangle(side_a, side_b, side_c));
                    list.PushFirst(ptr);
                    break;
                }

                case 2: {
                    std::cout << "Inserted Octagon" << std::endl;

                    int side = distFigureParam(generator);
                    std::shared_ptr<Figure> ptr =
std::make_shared<Octagon>(Octagon(side));
                    list.PushFirst(ptr);

                    break;
                }

                case 3: {
                    std::cout << "Inserted Foursquare" << std::endl;
                    int side = distFigureParam(generator);
                    std::shared_ptr<Figure> ptr =
std::make_shared<Foursquare>(Foursquare(side));

                    list.PushFirst(ptr);

                    break;
                }
            }
        }
    };


    Command cmdRemove = [&]() {
        std::lock_guard<std::mutex> guard(mtx);

        std::cout << "Command: Remove" << std::endl;

        if (list.IsEmpty()) {
            std::cout << "List is empty" << std::endl;
        } else {
            uint32_t seed =
std::chrono::system_clock::now().time_since_epoch().count();

            std::default_random_engine generator(seed);
            std::uniform_int_distribution<int> distSquare(1, 150);
            double sqr = distSquare(generator);
            std::cout << "Lesser than " << sqr << std::endl;
```

```
            for (int32_t i = 0; i < 5; ++i) {
                auto iter = list.begin();
                for (int32_t k = 0; k < list.GetLength(); ++k) {
                    if (iter->Square() < sqr) {
                        list.Pop(k);
                        break;
                    }
                    ++iter;
                }
            }
        }
    };

    Command cmdPrint = [&]() {
        std::lock_guard<std::mutex> guard(mtx);

        std::cout << "Command: Print" << std::endl;
        if(!list.IsEmpty()) {
            std::cout << list << std::endl;
        } else {
            std::cout << "List is empty." << std::endl;
        }
    };
    nlist.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
    nlist.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
    nlist.Push(std::shared_ptr<Command>(&cmdRemove, [](Command*){}));
    nlist.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));


    while (!nlist.IsEmpty()) {
        std::shared_ptr<Command> cmd = nlist.Top();
        std::future<void> ft = std::async(*cmd);
        ft.get();
        nlist.Pop();
    }

    return 0;
}
```

Вывод

Данная лабораторная работа была отличным завершением всех лабораторных работ. В ней я познакомился с новыми для себя лямбда функциями, которые, я думаю, пригодятся для дальнейшего изучения программирования.