Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 5

Тема: Основы работы с коллекциями: итераторы

Студент: Попов Илья Павлович
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2021

1. Постановка задачи

Создать шаблон динамической коллекцию, согласно варианту задания:
1. Коллекция должна быть реализована с помощью умных указателей (std::shared_ptr, std::weak_ptr). Опционально использование std::unique_ptr;
2. В качестве параметра шаблона коллекция должна принимать тип данных - шестиугольник;
3. Реализовать forward_iterator по коллекции;
4. Коллекция должны возвращать итераторы begin() и end();
5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator);
6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);
7. При выполнении недопустимых операций (например выход за границы коллекции или удаление несуществующего элемента) необходимо генерировать исключения;
8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count_if)
9. Список должен содержать метод доступа к элементу по оператору [].
10. Реализовать программу, которая:

позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;

позволяет удалять элемент из коллекции по номеру элемента;

выводит на экран введенные фигуры с помощью std::for_each;

выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count_if).

Варинат 15
Фигура: шестиугольник
Контейнер: список

2. Описание программы

Программа состоит из 8 файлов:

main.cpp - основное взаимодействие с пользователем
Hexagon.cpp - реализация 6-угольника
Hexagon.h - хедер-файл для 6-угольника
Tlist.cpp - реализация списка с использованием итераторов
Tlist.h - хедер-файл для списка
TIterator.h - реализация итератора
TListItem.cpp - реализация элемента списка
TListItem.h - хедер-файл для элемента списка

Моя динамическая структура реализована в файлах TListItem, TIterator, Tlist.
В них описана

1. Структура элемента(TListItem) - ссылка следующий и value из текущего
2. Методы для взаимодействия с множеством подобных элементов(Tlist)
3. Эти методы реализуются посредством итератора(TIterator) - ссылки на элемент.

3. Набор тестов

Проверка работы программы с некорректными входными данными.

Тест 1

1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print menu.
8. Print list with iterator.
9. Number of objects with an area less than the specified one.
0. Exit out program.

1
Side =1
Figure is added.
1
Side =2
Figure is added.
1

Side =3
Figure is added.
2
Side =4
Figure is added.
2
Side =5
Figure is added.
2
Side =6
Figure is added.
4
[1]a=3
area=31.1769

[2]a=2
area=13.8564

[3]a=1
area=3.4641

[4]a=4
area=55.4256

[5]a=5
area=86.6025

[6]a=6
area=124.708

8
a=3
area=31.1769

a=2
area=13.8564

a=1
area=3.4641

a=4
area=55.4256

a=5
area=86.6025

a=6
area=124.708

7
1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list

4. Print list.
5. Insert in list
6. Erase list.
7. Print menu.
8. Print list with iterator.
9. Number of objects with an area less than the specified one.
0. Exit out program.

3
Enter an index of figure you want to delete
1
Figure is deleted.
3
Enter an index of figure you want to delete
5
Figure is deleted.
4
[1]a=2
area=13.8564

[2]a=1
area=3.4641

[3]a=4
area=55.4256

[4]a=5
area=86.6025

5
Side =7
Please, enter an index.
1
Figure is added.
5
Side =8
Please, enter an index.
5
Figure is added.
4
[1]a=7
area=169.741

[2]a=2
area=13.8564

[3]a=1
area=3.4641

[4]a=4
area=55.4256

[5]a=8

area=221.703

[6]a=5
area=86.6025

9
Enter the number
70.1
[2]a=2
area=13.8564

[3]a=1
area=3.4641

[4]a=4
area=55.4256

0
Для продолжения нажмите любую клавишу . . .
List deleted!

Тест 2

Проверка работы программы с некорректными входными данными.

1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print menu.
8. Print list with iterator.
9. Number of objects with an area less than the specified one.
0. Exit out program.

dgag
Incorrect input!

5
Side =4
Enter an index.
dsfgsdg
Index error!

4. Результаты выполнения тестов

Представлены выше, с целью упростить прочтение.

5. Листинг программы

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <algorithm>
#include <functional>
#include "TListItem.h"
#include "TList.h"

void usage() {
	cout << "1. Add new item in begin of list." << endl;
	cout << "2. Add new item in end of list." << endl;
	cout << "3. Delete item from list" << endl;
	cout << "4. Print list." << endl;
	cout << "5. Insert in list" << endl;
	cout << "6. Erase list." << endl;
	cout << "7. Print menu." << endl;
	cout << "8. Print list with iterator." << endl;
	cout << "9. Number of objects with an area less than the specified one."
	<< endl;
	cout << "0. Exit out program." << endl << endl;
}

bool is_number(const string& s) {
	bool point = false;
	for (int i = 0; i < s.length(); ++i) {
		if (s[i] == '-' && i == 0) {
			continue;
		}
		else if (s[i] == '.') {
			if ((i == 0 || i == s.length() - 1) || point) {
				return false;
			}
			else {
				point = true;
			}
		}
		else if (s[i] < '0' || s[i] > '9') { return false; }
	}
	return true;
}


int main() {
	TList<Hexagon> list;
	string cur, ind_cur;
```

```
int menuNum = 7;
int ind;
int i = 1;
shared_ptr<Hexagon> sptr;

while (menuNum != 0) {
	try {
		switch (menuNum) {
		case 1:
			sptr = make_shared<Hexagon>(cin);
			list.addFirst(sptr);
			cout << "Figure is added." << endl;
			break;
		case 2:
			sptr = make_shared<Hexagon>(cin);
			list.addLast(sptr);
			cout << "Figure is added." << endl;
			break;
		case 3:
			cout << "Enter an index." << endl;
			cin >> ind_cur;
			if (!is_number(ind_cur)) {
				menuNum = 7;
				throw invalid_argument("Index error!\n\n");
				break;
			}
			ind = stoi(ind_cur);
			if (ind < 0) {
				menuNum = 7;
				throw invalid_argument("Index error!\n\n");
				break;
			}
			list.delElement(ind);
			cout << "Figure is deleted." << endl;
			break;
		case 4:
			cout << list << endl;
			break;
		case 5:
			sptr = make_shared<Hexagon>(cin);
			cout << "Enter an index." << endl;
			cin >> ind_cur;
			if (!is_number(ind_cur)) {
				menuNum = 7;
				throw invalid_argument("Index error!\n\n");
				break;
			}
			ind = stoi(ind_cur);
			if (ind < 0) {
				menuNum = 7;
				throw invalid_argument("Index error!\n\n");
				break;
			}
			list.insert(ind, sptr);
```

```
                cout << "Figure is added." << endl;
                break;
            case 6:
                list.eraseList();
                cout << "The list is cleared." << endl;
                break;
            case 7:
                usage();
                break;
            case 8:
                for (auto i : list) {
                    i->Print();
                }
                break;
            case 9:
                long double n;
                cout << "Enter the number " << endl;
                cin >> n;
                for (auto figure : list) {
                    if (figure->area() < n) {
                        cout << "[" << i << "]";
                        figure->Print();
                    }
                    i++;
                }
                break;
            }
            cin >> cur;
            if (!is_number(cur)) {
                menuNum = 7;
                throw invalid_argument("Incorrect input!\n\n");
            }
            menuNum = stoi(cur);
        }
        catch (invalid_argument& arg) {
            cout << arg.what() << endl;
        }
    }

    system("pause");
    return 0;
}
```

Hexagon.cpp

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

Hexagon::Hexagon() : Hexagon(0) {
}
```

```
Hexagon::Hexagon(istream& is) {
	cout << "Side =";
	is >> side;
}

Hexagon::Hexagon(long int i) : side(i) {}

Hexagon::Hexagon(const Hexagon& orig) { side = orig.side; }

Hexagon::~Hexagon() {}

long double Hexagon::area() {
	long double res = 2 * sqrt(3) * pow(side, 2);
	return res;
}

Hexagon& Hexagon::operator=(const Hexagon& right) {
	if (this == &right) return *this;
	side = right.side;

	return *this;
}

bool operator<(Hexagon& left, Hexagon& right) {
	return left.area() < right.area();
}

bool operator==(const Hexagon& left, const Hexagon& right) {
	return left.side == right.side;
}

void Hexagon::Print() {
	cout << "a=" << side << endl;
	cout << "   area=" << this->area() << endl << endl;
}

ostream& operator<<(ostream& os, const Hexagon& obj) {
	os << "a=" << obj.side;
	return os;
}

istream& operator>>(istream& is, Hexagon& obj) {
	is >> obj.side;
	return is;
}
```

Hexagon.h

```
#ifndef Hexagon_H
#define	Hexagon_H
```

```
#include <cstdlib>
#include <iostream>
//#include "Figure.h"
using namespace std;

class Hexagon{// : public Figure {
public:
        Hexagon();
        Hexagon(istream& is);
        Hexagon(const Hexagon& orig);
        Hexagon(long int i);

        friend bool operator==(const Hexagon& left, const Hexagon& right);
        friend bool operator<(Hexagon& left, Hexagon& right);

        friend ostream& operator<<(ostream& os, const Hexagon& obj);
        friend istream& operator>>(istream& is, Hexagon& obj);

        Hexagon& operator=(const Hexagon& right);

        long double area();
        void Print();
        ~Hexagon();
private:
        long int side;
};


#endif
```

Tlist.cpp

```
#include "TList.h"
#include "TIterator.h"


template <class T>
TList<T>::TList() {        first = nullptr;}

template <class T>
TList<T>::~TList() { cout << "List deleted!" << endl; }

template <class T>
int TList<T>::length() {
        int i = 0;
        shared_ptr<TListItem<T>> item = this->first;
        while (item != nullptr){
                item = item->GetNext();
                i++;
        }
        return i;
```

```
}

template <class T>
void TList<T>::addFirst(shared_ptr<T> &figure) {
	shared_ptr<TListItem<T>>                    other                    =
	make_shared<TListItem<T>>(figure);

	other->SetNext(first);
	first = other;
}

template <class T>
void TList<T>::insert(int index, shared_ptr<T> &figure) {
	shared_ptr<TListItem<T>> iter = this->first;
	shared_ptr<TListItem<T>>                    other                    =
	make_shared<TListItem<T>>(figure);

	if (index == 1) {
		other->SetNext(iter);
		this->first = other;
	}
	else {
		if (index <= this->length()) {
			for (int i = 1; i < index - 1; ++i) {
				iter = iter->GetNext();
			}
			other->SetNext(iter->GetNext());
			iter->SetNext(other);
		}
		else {
			throw invalid_argument("Index error!\n\n");
		}
	}
}

template <class T>
void TList<T>::addLast(shared_ptr<T> &figure) {
	shared_ptr<TListItem<T>>                    other                    =
	make_shared<TListItem<T>>(figure);
	shared_ptr<TListItem<T>> iter = this->first;
	if (first != nullptr) {
		while (iter->GetNext() != nullptr) {
			iter = iter->SetNext(iter->GetNext());
		}
		iter->SetNext(other);
		other->SetNext(nullptr);
	}
	else {
		first = other;
	}
}

template <class T>
bool TList<T>::empty() {
```

```
	return first == nullptr;
}

template <class T>
void TList<T>::delElement(int &index) {
	shared_ptr<TListItem<T>>iter = this->first;
	if (index <= this->length()) {
		if (index == 1) {
			this->first = iter->GetNext();
		}
		else {
			for (int i = 1; i < index - 1; ++i) {
				iter = iter->GetNext();
			}
			iter->SetNext(iter->GetNext()->GetNext());
		}
	}
	else {
		throw invalid_argument("Index error!\n\n");
	}

}

template <class T>
void TList<T>::eraseList() {
	first = nullptr;
}

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

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

template <class T>
ostream& operator<<(ostream& os, const TList<T>& list) {
	shared_ptr<TListItem<T>> item = list.first;
	int i = 1;
	while (item != nullptr) {
		cout << "[" << i << "]";
		item->GetFigure()->Print();
		item = item->GetNext();
		i++;
	}
	return os;
}

template class TList<Hexagon>;
template ostream& operator<<(ostream &out, const TList<Hexagon> &figure);
```

Tlist.h

```
#ifndef TLIST_H
#define	TLIST_H

#include <memory>
#include "Hexagon.h"
#include "TListItem.h"
#include "TIterator.h"

template <class T>
class TList {
public:
	TList();
	virtual ~TList();

	int length();

	void addFirst(shared_ptr<T> &figure);
	void insert(int index, shared_ptr<T> &figure);
	void addLast(shared_ptr<T> &figure);

	bool empty();
	void delElement(int &index);
	void eraseList();

	TIterator<TListItem<T>, T> begin();
	TIterator<TListItem<T>, T> end();
	template <class A> friend ostream& operator<<(ostream& os, const
	TList<A>& list);

private:
	shared_ptr<TListItem<T>> first;
};

#endif/* TLIST_H */
```

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H

#include <memory>
#include <iostream>
using namespace std;
```

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

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

	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:
	shared_ptr<N> cur;
};

#endif
```

TListItem.cpp

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

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

```
template <class T>
TListItem<T>::~TListItem() {}

template <class T>
shared_ptr<TListItem<T>>
	TListItem<T>::SetNext(shared_ptr<TListItem<T>> next) {
	shared_ptr<TListItem<T>> old = this->next;
	this->next = next;
	return old;
}

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

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

template <class T>
ostream& operator<<(ostream& os, const TListItem<T>& obj) {
	os << "[" << obj.figure << "]";
	return os;
}

template class TListItem<Hexagon>;
template ostream& operator<<(ostream& out, const TListItem<Hexagon>&
	obj);
```

TListItem.h

```
#ifndef TLISTITEM_H
#define	TLISTITEM_H

#include "Hexagon.h"
#include <memory>


template <class T>
class TListItem {
public:
	TListItem(const shared_ptr<T> &figure);
	template <class A> friend ostream& operator<<(ostream& os, const
	TListItem<A>& obj);

	shared_ptr<TListItem<T>> SetNext(shared_ptr<TListItem<T>> next);
	shared_ptr<TListItem<T>> GetNext();
```

```
        shared_ptr<T> GetFigure();

        virtual ~TListItem();
private:
        shared_ptr<T> figure;
        shared_ptr<TListItem<T>> next;
};

#endif
```

6. Выводы

В ходе этой лабораторной работы я познакомился с концепцией шаблонов в языке C++, а также с таким понятием, как умный указатель. Он зачастую бывает очень удобен, так как при выходе объекта из зоны видимости указатель не просто очищает занимаемую им память, но и проверяет, имеется ли обращение к этому объекту из других участков программы, и если таковые имеются, не станет удалять объект.

7. ЛИТЕРАТУРА

1. Справочник по языку C++ [Электронный ресурс].
URL: https://ravesli.com (дата обращения:17.10.2021).
2. Умный указатель std::shared_ptr в C++ [Электронный ресурс].
URL: https://ravesli.com/urok-194-std-shared_ptr/ (дата обращения:15.11.2021).