Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа №5.

Группа: М8О – 104Б-16
Студент: Чекушкин Денис Игоревич
Преподаватель: Поповкин Александр Викторович
Вариант: №18

Москва
2017

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Закрепление навыков работы с шаблонами классов.
- Построение итераторов для динамических структур данных.

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛРNo4) 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;

Листинг

btree.h

```
#ifndef BTREE_H
#define BTREE_H
#include <iostream>
#include "btree_item.h"
#include "iterator.h"
template <class T>
class Btree
{
public:
 Btree();

 void bstInsert(const std::shared_ptr<T>& figure);
 void bstRemove(const std::shared_ptr<T>& figure);
 void Insert(const std::shared_ptr<T>& figure);

 Iterator<BTreeItem<T>, T> begin() const;
 Iterator<BTreeItem<T>, T> end() const;



 template <class B>
 friend std::ostream& operator << (std::ostream& os, const Btree<B>& Btree);
 template <class B>
 void print1(const Btree<B>& btree);
```

```
private:
 std::shared_ptr<BTreeItem<T>> m_root;
};

#include "btree.cpp"
#endif
```

iterator.h

```
#ifndef ITERATOR_H
#define ITERATOR_H

template <class N, class T>
class Iterator
{
public:
 Iterator(const std::shared_ptr<N>& item);
 std::shared_ptr<T> operator * ();
 std::shared_ptr<T> operator -> ();
 Iterator left ();
 Iterator right ();
 Iterator operator ++ ();
 //Iterator operator ++ (int index);
 bool operator == (const Iterator& other) const;
 bool operator != (const Iterator& other) const;

private:
 std::shared_ptr<N> m_item;
  void nextInorder()
              {
              if (m_item->element == NULL)return;
              else {
              nextInorder(m_item->left);
              nextInorder(m_item->element);
              nextInorder(m_item->right);
        }
    }

};
```

```
#include "iterator.cpp"
#endif
```

iterator.cpp

```
template <class N, class T>
Iterator<N, T>::Iterator(const std::shared_ptr<N>& item)
{
 m_item = m_item->GoFarLeft(item);//
}
template <class N, class T>
std::shared_ptr<T> Iterator<N, T>::operator * ()
{
 return m_item->getFigure();
}
template <class N, class T>
std::shared_ptr<T> Iterator<N, T>::operator -> ()
{
 return m_item->getFigure();
}
template <class N, class T>
Iterator<N, T> Iterator<N, T>::left ()
{
 m_item = m_item->getLeft();

 return *this;
}
template <class N, class T>
Iterator<N, T> Iterator<N, T>::right ()
{
 m_item = m_item->getRight();

 return *this;
}
template <class N, class T>
Iterator<N, T> Iterator<N, T>::operator ++ ()
{
 m_item = m_item->GetNext();
}
```

```
template <class N, class T>
bool Iterator<N, T>::operator == (const Iterator& other) const
{
 return m_item == other.m_item;
}
template <class N, class T>
bool Iterator<N, T>::operator != (const Iterator& other) const
{
 return !(*this == other);
}
```

Tstack.h

```
#ifndef TSTACK_H
#define TSTACK_H

#include <iostream>
#include <memory>
using std::shared_ptr;
const int MAX_TSTACK_SIZE = 20;

template <typename T>
class TStack {
private:
    T items[MAX_TSTACK_SIZE];
    int top;
public:
    TStack() : top(0) {};
    bool IsEmpty() const { return  top == 0; };
    bool IsFull() const { return top == MAX_TSTACK_SIZE; };
    void Push(const T & itm);
    T Pop();
    void ClearStack() { top = 0; };
};
#endif /* TSTACK_H */
```

Tstack.cpp

```
#include "TStack.h"
template <typename T>
```

```
void TStack<T>::Push(const T & itm) {
    if (top == MAX_TSTACK_SIZE) {
        std::cerr << "TStack is full!\n";
        exit(1);
    }
    else {
        items[top++] = itm;
    }
}
template <typename T>
T TStack<T>::Pop() {
    T temp(0);
    if (top > 0) {
        temp = items[--top];
        return temp;
    }
    else {
        std::cerr << "TStack is empty!\n";
        exit(1);
    }
}

#include "figure.h"
#include "btree_item.h"
template class Tstack<shared_ptr<TNode<Figure>>>;
```

Выводы: в данной лабораторной работе я получил навыки пограммирования итераторов на языке C++, закрепил навык работы с шаблонами классов. Добавил итератор.

https://github.com/israelcode/oop/tree/master/sem2/lab5