Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа №7.

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

Москва
2017

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Создание сложных динамических структур данных.
- Закрепление принципа ОСР.

ЗАДАНИЕ

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер, одного из следующих видов (Контейнер 1-го уровня):

1. Массив
2. Связанный список
3. Бинарное- Дерево.
4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).
5. Очередь
6. Стек

Каждым элементом контейнера, в свою, является динамической структурой данных одного из следующих видов (Контейнер 2-го уровня):

1. Массив
2. Связанный список
3. Бинарное- Дерево.
4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).
5. Очередь
6. Стек

Таким образом у нас получается контейнер в контейнере. Т.е. для варианта (1,2) это будет массив, каждый из элементов которого – связанный список.

При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5.

Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня. Например, для варианта (1,2) добавление объектов будет выглядеть следующим образом:

1. Вначале массив пустой.
2. Добавляем Объект1: В массиве по индексу 0 создается элемент с типом список, в список добавляется Объект 1.
3. Добавляем Объект2: Объект добавляется в список, находящийся в массиве по индекс 0.
4. Добавляем Объект3: Объект добавляется в список, находящийся в

массиве по индекс 0.

5. Добавляем Объект4:Объект добавляется в список, находящийся в массиве по индекс 0.

6. Добавляем Объект5: Объект добавляется в список, находящийся в массиве по индекс 0.

7. Добавляем Объект6:В массиве по индексу 1 создается элемент с типом список, в список добавляется Объект 6. Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта

Листинг

criteria.h

```
#ifndef CRITERIA_H
#define CRITERIA_H
template <class T>
class Criteria
{
public:
    virtual bool check(const std::shared_ptr<T>& item) const = 0;
};

#endif
```

ctiteria_area.h

```
#ifndef CRITERIA_AREA_H
#define CRITERIA_AREA_H

#include "criteria.h"

template <class T>
class CriteriaArea : public Criteria<T>
{
public:
    CriteriaArea(double area);

    bool check(const std::shared_ptr<T>& item) const override;

private:
    double m_area;
};
```

```
#include "criteria_area_impl.cpp"
#endif
criteria_area_impl.cpp
template <class T>
CriteriaArea<T>::CriteriaArea(double area)
{
    m_area = area;
}
template <class T>
bool CriteriaArea<T>::check(const std::shared_ptr<T>& item) const
{
    return ((item->area()) == m_area);
}
criteria_type.h
#ifndef CRITERIA_TYPE_H
#define CRITERIA_TYPE_H
#include <cstring>
#include "criteria.h"
template <class T>
class CriteriaType : public Criteria<T>
{
public:
    CriteriaType(const char* type);

    bool check(const std::shared_ptr<T>& item) const override;

private:
    char m_type[16];
};
#include "criteria_type_impl.cpp"
#endif
ctiteria_type_impl.cpp
template <class T>
CriteriaType<T>::CriteriaType(const char* type)
{
    strcpy(m_type, type);
}
```

```
template <class T>
bool CriteriaType<T>::check(const std::shared_ptr<T>& item) const
{
    return strcmp(m_type, item->getName()) == 0;
}
container.h
#ifndef CONTAINTER_H
#define CONTAINTER_H
#include <memory>
#include <cstring>
#include "btree.h"
#include "stack.h"
#include "criteria.h"
template <class T>
class Container
{
public:
    void add(const std::shared_ptr<T>& item);
    void erase(const Criteria<T>& criteria);
Stack<Figure> stack2;
    template <class B>
    friend std::ostream& operator << (std::ostream& os, const Container<B>& container);

private:
    Btree<Stack<T>> m_container;
};

#include "container.cpp"
#endif
container.cpp
template <class T>
void Container<T>::add(const std::shared_ptr<T>& item)
{
auto lastContIt = m_container.begin();
    if (lastContIt == m_container.end()){
    m_container.bstInsert(std::make_shared<Stack<T>>());
    }
```

```
lastContIt = m_container.begin();
while (lastContIt.getItem()->GetNext() != nullptr){
    ++lastContIt;
    }
if ((*lastContIt)->size() == 5)
    {
      m_container.bstInsert(std::make_shared<Stack<T>>());
      ++lastContIt;
    }
(*lastContIt)->Push(item);
}
template <class T>
void Container<T>::erase(const Criteria<T>& criteria)
{
int size=0;
for (auto subCont : m_container)
    {
      while (true)
      {
            bool isRemoved = false;

            for (unsigned int i = 0; i < subCont->size(); ++i)
            {
                  auto elemIt = subCont->get(i);
                  size=subCont->size();

                  if (criteria.check(*elemIt))
                  {
                        subCont->erase(subCont, i,size, stack2);
                        isRemoved = true;
                        break;
                  }
            }

            if (!isRemoved)
                  break;
      }
    }
```

```
}
template <class K>
std::ostream& operator << (std::ostream& os, const Container<K>& container)
{
    if (container.m_container.size() == 0)
    {
      os << "================" << std::endl;
      os << "Container is empty" << std::endl;
    }
    else
    {
      unsigned int containerCnt1 = 1;

      for (auto subCont : container.m_container)
      {
            unsigned int containerCnt2 = 1;

            os << "================" << std::endl;
            os << "Container #" << (containerCnt1++) << ":" << std::endl;

            for (auto subItem : *subCont)
            {
                  os << "================" << std::endl;
                  os << "Item #" << (containerCnt2++) << ":" << std::endl;
                  subItem->print();

                  os << "Area: " << subItem->area() << std::endl;
            }
      }
    }
    return os;
}
```

Выводы: в данной лабораторной работе я закрепил навыки работы с памятью на языке C++, получил навыки содания сложных динамических. Добавил хранилище данных - «Контейнер»(Бинарное дерево со стеками)

https://github.com/israelcode/oop/tree/master/sem2/lab7