Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Объектно-ориентированное программирование»

Студент: Севастьянов В. С.
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 19
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №2

1 Цель работы

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

2 Задача

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровная, содержащий одну фигуру, согласно варианту задания

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.
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream(»). Оператор должен вводить параметры фигуры.
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен содержать 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.
- Класс-контейнер должен иметь метод по удалению фигуры из 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).

Фигура: Прямоугольник.
Контейнер: N-дерево.

3 Описание

Динамические структуры данных используются в тех случаях, когда мы заранее не знаем, сколько памяти необходимо выделить для нашей программы – это выясняется только в процессе работы. В общем случае эта структура представляет собо отдельные элементы, связанные между собой с помощью ссылок. Каждый элемент состоит из двух областей памяти: поля данных и ссылок. Ссылки – это адреса других узлов того же типа, с которыми данный элемент логически связан. При добавлении нового элемента в такую структуру выделяется новый блок памяти и устанавливаются связи этого элемента с уже существующими.

Структура данных список является простейшим типом данных динамической структуры, состоящей из узлов. Каждый узел включает в себя в классическом варианте два поля: данные и указатель на следующий узел в списке. Элементы связного списка можно вставлять и удалять произвольным образом. Доступ к списку осуществляется через указатель, который содержит ядрес первого элемента списка, называемого головой списка.

Параметры в функцию могут передаваться одним из следующих способов: по значению и по ссылке. При передаче аргументов по значению компилятор создает временную копию объекта, который должен быть передан, и размещает его в области стековой памяти, предназначенной для хранения локальных объектов. Вызываемая функция оперирует именно с этой копией, не оказывая влияния на оригинал объекта. Прототипы функций, принимающих аргументы по значению, предусматривают в качестве параметров указание типа объекта,а не его адреса. Если же необходимо, чтобы функция модифицировала оригинал объекта, используется передача параметров по ссылке. При этом в функцию передается не сам объект, а только его адрес. Таким образом, все модификации в теле функции переданных ей по ссылке аргументов воздействуют на объект. Использование передачи адреса объекта весьма эффективный способ работы с большим числом данных. Кроме того, так как передается адрес, а не сам объект, существенно экономится стековая память.

4 Исходный код

```
1
2 class Tree
3 {
4 public:
5   Tree();
```

```
6     TreeItem* insert(TreeItem *node, size_t key);
7     void remove(size_t key);
8     void remove(TreeItem* tree, size_t key);
9     bool empty();
10    TreeItem* find(TreeItem* tree, size_t key);
11    void print();
12    void print(TreeItem *tree, size_t depth);
13    Rectangle data(size_t key);
14    friend std::ostream& operator<<(std::ostream& os, const Tree& obj);
15    ~Tree();
16  private:
17    TreeItem *root;
18  };
19
20  class TreeItem
21  {
22  public:
23    TreeItem(const Rectangle& data);
24    TreeItem(const TreeItem& orig);
25    TreeItem* getSon();
26    TreeItem* getSibling();
27    TreeItem* setSon(TreeItem* son);
28    TreeItem* setSibling(TreeItem* sibling);
29    size_t getData();
30    Rectangle TreeItem::getRectangle() const;
31    friend std::ostream& operator<<(std::ostream& os, const TreeItem& obj);
32    ~TreeItem();
33  private:
34    TreeItem *sibling;
35    TreeItem *son;
36    Rectangle data;
37  };
```

5 Выводы

В этой работе было необходимо реализовать собственную структуру данных. В моем случае ей являлось н-дерево. Мною было сделано дерево, вершины которого имеют указатели на ближайшего сына и братьев. Такая структура позволяет добавлять какое-угодно количество сыновей любой вершине без использования стандартных библиотек. Несмотря на то, что все основные структуры уже созданы до нас, каждый уважающий себя программист должен мочь реализовать их. Значения в дереве хранятся по ключу.