Приложение C. Базовые классы

В данном приложении документируются базовые классы, которые применялись в примерах кода на C++ в описаниях различных паттернов проектирования. Мы специально стремились сделать эти классы простыми и минимальными. Будут описаны следующие классы:

- `List` – упорядоченный список объектов;
- `Iterator` – интерфейс для последовательного доступа к объектам в агрегате;
- `ListIterator` – итератор для обхода списка;
- `Point` – точка с двумя координатами;
- `Rect` – прямоугольник, стороны которого параллельны осям координат.

Некоторые появившиеся сравнительно недавно стандартные типы C++, возможно, реализованы еще не во всех компиляторах. В частности, если ваш компилятор не поддерживает тип `bool`, его можно определить самостоятельно:

```
typedef int bool;
const int true = 1;
const int false = 0;
```

C.1. List

Шаблон класса `List` представляет собой базовый контейнер для хранения упорядоченного списка объектов. В списке хранятся значения элементов, то есть он пригоден как для встроенных типов, так и для экземпляров классов. Например, запись `List<int>` объявляет список целых `int`. Но в большинстве паттернов в списке хранятся указатели на объекты, скажем, `List<Glyph*>`. Это позволяет использовать класс `List` для хранения разнородных объектов (точнее, указателей на них).

Для удобства в классе `List` есть синонимы для операций со стеком. Это позволяет явно использовать список в роли стека, не определяя дополнительного класса:

```
template <class Item>
class List {
public:
    List(long size = DEFAULT_LIST_CAPACITY);
    List(List&);
    ~List();
    List& operator=(const List&);
```

```
    long Count() const;
    Item& Get(long index) const;
    Item& First() const;
    Item& Last() const;
    bool Includes(const Item&) const;

    void Append(const Item&);
    void Prepend(const Item&);

    void Remove(const Item&);
    void RemoveLast();
    void RemoveFirst();
    void RemoveAll();

    Item& Top() const;
    void Push(const Item&);
    Item& Pop();
};
```

В следующих разделах операции описываются более подробно.

Конструктор, деструктор, инициализация и присваивание

`List(long size)` – инициализирует список. Параметр `size` определяет начальное число элементов в списке.

`List(List&)` – замещает определяемый по умолчанию копирующий конструктор для правильной инициализации данных-членов.

`~List()` – освобождает внутренние структуры данных списка, но *не* элементы списка. Не предполагается, что у этого класса будут производные, поэтому деструктор не объявлен виртуальным.

`List& operator=(const List&)` – реализует операцию присваивания.

Доступ

Следующие операции обеспечивают доступ к элементам списка.

`long Count() const` – возвращает число объектов в списке.

`Item& Get(long index) const` – возвращение объекта с заданным индексом.

`Item& First() const` – возвращает первый объект в списке.

`Item& Last() const` – возвращение последнего объекта в списке.

Добавление

`void Append(const Item&)` – добавляет свой аргумент в конец списка.

`void Prepend(const Item&)` – добавляет свой аргумент в начало списка.

Удаление

`void Remove(const Item&)` – удаляет заданный элемент из списка. Для применения этой операции требуется, чтобы тип элементов поддерживал оператор сравнения на равенство ==.

`void RemoveFirst()` – удаляет первый элемент из списка.

`void RemoveLast()` – удаление последнего элемента из списка.

`void RemoveAll()` – удаляет все элементы из списка.

Интерфейс стека

`Item& Top() const` – возвращает элемент, находящийся на вершине стека.
`void Push(const Item&)` – «заталкивает» элемент в стек.
`Item& Pop()` – «выталкивает» элемент с вершины стека.

C.2. Iterator

`Iterator` – это абстрактный класс, который определяет интерфейс обхода агрегата:

```
template <class Item>
class Iterator {
public:
    virtual void First() = 0;
    virtual void Next() = 0;
    virtual bool IsDone() const = 0;
    virtual Item CurrentItem() const = 0;
protected:
    Iterator();
};
```

Операции делают следующее:

`virtual void First()` – позиционирует итератор на первый объект в агрегате.

`virtual void Next()` – позиционирует итератор на следующий по порядку объект.

`virtual bool IsDone() const` – возвращает `true`, если больше не осталось объектов.

`virtual Item CurrentItem() const` – возвращает объект, находящийся в текущей позиции.

C.3. ListIterator

`ListIterator` реализует интерфейс класса `Iterator` для обхода списка `List`. Его конструктор принимает в качестве аргумента список, который нужно обойти:

```
template <class Item>
class ListIterator : public Iterator<Item> {
public:
    ListIterator(const List<Item>* aList);

    virtual void First();
    virtual void Next();
    virtual bool IsDone() const;
    virtual Item CurrentItem() const;
};
```