Разработка и реализация класса “Целочисленный массив с динамически изменяемыми границами”

1 Общие положения

В процессе выполнения работы должен быть разработан, реализован и протестирован класс **IntArray**, обеспечивающий работу с массивами целых чисел с динамически изменяемыми границами. В качестве инструмента реализации должна быть использована система программирования C++ (см. [3]).

В рамках данной работы массивы представляют собой одномерные изменяемые объекты, которые могут динамически сжиматься и расширяться (см. [2], стр. 366, приложение A.9.8). Каждый массив имеет нижнюю границу и последовательность элементов, пронумерованных, начиная с нижней границы. Все элементы массива относятся к одному и тому же типу. Индексы элементов массива являются целочисленными.

При добавлении элемента со стороны нижней границы значение индекса, соответствующего нижней границе, уменьшается на 1.

При добавлении элемента со стороны верхней границы значение индекса, соответствующего верхней границе, увеличивается на 1.

При удалении элемента со стороны нижней границы значение индекса, соответствующего нижней границе, увеличивается на 1.

При удалении элемента со стороны верхней границы значение индекса, соответствующего верхней границе, уменьшается на 1.

Для реализации рекомендуется использовать динамическую структуру вида “дек” (см. [1], стр. 295, раздел 2.2).

Разработка и реализация методов должна выполняться **строго** в соответствии с приведёнными в задании спецификациями этих методов.

Перед передачей выполненной реализации на проверку данная реализация должна быть протестирована разработчиком.

Для успешной сдачи работы разработчик должен понимать и уметь объяснить все приведённые в работе объявления классов, методов и их реализации.

2 Спецификация интерфейса класса

Интерфейс класса **IntArray** представляет собой набор следующих методов:

- **IntArray()** — конструктор создания массива с нижней границей, равной `1`;
- **IntArray(int lb)** — конструктор создания массива со значением нижней границы, равной `lb`;
- **IntArray(IntArray& parray)** — конструктор создания копии массива, соответствущей массиву `parray` (конструктор копирования);
- **IntArray(int lb, int cnt, int _element)** — конструктор создания массива, каждый элемент которого принимает значение `_element`, значение нижней границы массива равно `lb`, количество элементов массива равно `cnt`;
- **int Low()** — получение значения нижней границы массива;

- **int High()** — получение значения верхней границы массива;
- **int Size()** — получение количества элементов массива;
- **int Fetch(int index)** — получение значения элемента массива с индексом `i`;
- **void Store(int index, int _element)** — изменение значения элемента массива с индексом `i` на новое значение `_element`;
- **int& operator[](int index)** — переопределение операции обращения к элементу массива.
  Данное переопределение должно обеспечить возможность выполнения, в частности, операций, подобных следующему примеру:

```
a + b[i];
b[i] = 125;
```

  где `a`, `b` — объекты класса **IntArray**.
  В результате выполнения этой операции в массив `a` должен быть добавлен сверху элемент массива `b`, индекс которого равен `i`.
- **void AddH(int _element)** — добавление нового элемента со значением `_element` со стороны верхней границы массива.
  После выполнения операции значение верхней границы массива увеличивается на 1.
- **void AddL(int _element)** — добавление нового элемента со значением `_element` со стороны нижней границы массива.
  После выполнения операции значение нижней границы массива уменьшается на 1.
- **int RemH()** — удаление элемента со стороны верхней границы массива.
  В качестве результата операция возвращает значение удалённого элемента.
  После выполнения операции значение верхней границы массива уменьшается на 1.
- **int RemL()** — удаление элемента со стороны нижней границы массива.
  В качестве результата операция возвращает значение удалённого элемента.
  После выполнения операции значение нижней границы массива увеличивается на 1.
- **int operator--(int)** — переопределение постфиксной операции `--` .
  Данное переопределение должно обеспечить возможность выполнения операций, подобных следующему примеру:

```
a--;
```

  В результате выполнения данной операции в массиве `a` происходит удаление элемента со стороны верхней границы массива.
  В качестве результата операция возвращает значение удалённого элемента.
  После выполнения операции значение верхней границы массива уменьшается на 1.
- **int operator--()** — переопределение префиксной операции `--` .
  Данное переопределение должно обеспечить возможность выполнения операций, подобных следующему примеру:

```
--a;
```

  В результате выполнения данной операции в массиве a происходит удаление элемента со стороны нижней границы массива.
  В качестве результата операция возвращает значение удалённого элемента.
  После выполнения операции значение нижней границы массива увеличивается на 1.

- **~IntArray()** — деструктор, обеспечивающий удаление массива.

Помимо приведённых выше методов класса **IntArray**, должны быть реализованы следующие методы, не относящиеся к какому-либо классу:

- **void operator+(int _element, IntArray& intarray)** — переопределение бинарной операции `+` .
  Данное переопределение должно обеспечить возможность выполнения операций, подобных следующему примеру:

  ```
  123 + a;
  ```

  В результате выполнения данной операции в массиве a происходит добавление элемента со значением 123 со стороны нижней границы массива.
  После выполнения операции значение нижней границы уменьшается на `1`.

- **void operator+(IntArray& intarray, int _element)** — переопределение бинарной операции `+`.
  Данное переопределение должно обеспечить возможность выполнения операций, подобных следующему примеру:

  ```
  a + 123;
  ```

  В результате выполнения данной операции в массиве a происходит добавление элемента со значением 123 со стороны верхней границы массива.
  После выполнения операции значение верхней границы увеличивается на `1`.

При разработке и реализации класса необходимо решить, реализация каких операций может быть дана в рамках описания класса, то есть какие из операций будут “in-line” операциями. Исходные тексты реализации таких операций должны быть даны в рамках описания класса в файле `intarray.hpp`. Исходные тексты реализации остальных операций должны быть даны в файле `intarray.cpp`.

Рекомендуется сначала разработать, реализовать и протестировать класс без переопределения операций. После выполнения тестирования и исправления ошибок дополнить класс переопределением операций.

3 Требования

При выполнении курсовой работы ЗАПРЕЩАЕТСЯ использовать классы стандартной библиотеки классов системы программирования C++ !

В процессе выполнения курсовой работы должны быть созданы и предоставлены на проверку следующие файлы:

- `intarray.hpp` — файл с описанием класса **IntArray**;
- `intarray.cpp` — файл с исходным текстом реализации методов;
- `test.cpp` — файл, содержащий исходный текст на языке программирования `C++` с реализацией функции `main()`, обеспечивающей тестирование разработанного класса.

4 Требования к оформлению писем электронной почты

Подготовленные задания отправляются на адрес электронной почты:

- `andrei.stankevich@gmail.com`

Тема письма должна быть задана в соответствии со следующим форматом:

- `<Фамилия И.О.> <Группа> C++ ЭТАП <Номер этапа>`

Пример:

- `Иванов И.И. 60-208Б C++`

Все письма должны отправляться как ответы на соответствующие письма с сохранением истории переписки. Во вложении, в этом случае, должны быть только обновлённые файлы. Это значит, что предыдущие версии файлов в очередной ответ не вкладываются.

Внимание! Письма, оформленные с отступлением от приведённых правил, рассматриваться не будут.

Список литературы

[1] Дональд Кнут, Искусство программирования для ЭВМ, том 1 “Основные алгоритмы”, М., “Мир”, 1976.

[2] Барбара Лисков, Джон Гатэг, Использование абстракций и спецификаций при разработке программ, М., “Мир”, 1989.

[3] Т.А. Павловская, C/C++. Программирование на языке высокого уровня, СПб., “Питер”, 2013.