

ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”
ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

Задача 2: Реализация на абстрактния тип данни *Vector*

ЛЮБОМИР ЧОРБАДЖИЕВ
lchorbadjiev@elsys-bg.org



12 май 2009 г.

1 Условие на задачата

1.1 Предварителни данни

Целта на задачата е да реализирате абстрактния тип данни `Vector`. Класът `Vector` трябва да поддържа динамично заделен буфер за съхранение на неговите елементи, като при нужда размера на буфера трябва да се променя. Примерна организация на работата е класа е:

```
template<class T> class Vector {
    int capacity_;
    int size_;
    T* buffer_;
public:
    Vector(int capacity=10);
    ~Vector();
    ...
};
```

Примерна реализация на конструктора и деструктора е дадена по-долу.

```
template<class T>
Vector<T>::Vector(int capacity)
: capacity_(capacity),
  size_(0),
  buffer_(new T[capacity])
{}

template<class T>
Vector::~~Vector() {
    delete [] buffer_;
}
```

1.2 Основни методи (20 точки)

Основните методи на класа `Vector`, които трябва да се реализират са следните:

- `int size() const;`
Връща размера на вектора (броя елементи във вектора)
- `bool empty() const;`
Връща `true` когато вектора е празен (т.е. когато `size()==0`).
- `T& operator[] (size_type n);`
Връща препратка към `n`-тия елемент на вектора (елемента с индекс `n`).
- `const T& operator[] (size_type n) const;`
Връща константна препратка към `n`-тия елемент на вектора (елемента с индекс `n`).
- `void clear();`
Изтрива всички елменти на вектора.

- `int capacity() const;`
Връща броя елементи за които е заделена памет. Методът `capacity()` винаги връща стойност, която е по-голяма или равна на `size()`.

1.3 Копиращ конструктор и оператор за присвояване (20 точки)

За класа `Vector` трябва да се дефинират копиращ конструктор и оператор за присвояване.

- `Vector(const Vector& other);`
Копиращ конструктор.
- `Vector& operator=(const Vector& other);`
Оператор за присвояване.

1.4 Клас `iterator` (20 точки)

В класа `Vector` трябва да се дефинира вътрешен клас `iterator`, който да позволява обхождане и манипулиране на елементите във вектора. В класа `Vector` трябва да се добавят методи `begin()` и `end()`, които връщат итератори насочени към първия елемент и с едно след последния елемент съответно.

Примерна декларация на класа `iterator` е дадена по-долу:

```
template<class T>
class Vector{
...
public:
    class iterator {
        ...
        public:
            iterator operator++();
            iterator operator++(int);
            bool operator==(const iterator& other) const;
            bool operator!=(const iterator& other) const;
            T& operator*();
            T* operator->();
    };
    ...
    iterator begin();
    iterator end();
};
```

1.5 Клас `const_iterator`, `reverse_iterator` и `const_reverse_iterator` (20 точки)

В класа `Vector` трябва да се дефинира вътрешни класове `iterator`, който да позволява обхождане и манипулиране на елементите във вектора. В класа `Vector` трябва да се добавят методи `begin()`, `end()`, `rbegin()` и `rend()`, които връщат итератори насочени към подходящите елементи на вектора.

Примерна декларация на тези класове и методи е дадена по-долу:

```
template<class T>
class Vector{
...
public:
    class const_iterator {
    ...
    public:
        iterator operator++();
        iterator operator++(int);
        bool operator==(const iterator& other) const;
        bool operator!=(const iterator& other) const;
        const T& operator*();
        const T* operator->();
    };
    ...
    const_iterator begin() const;
    const_iterator end() const;

    class reverse_iterator {...};
    class const_reverse_iterator {...};

    reverse_iterator rbegin();
    reverse_iterator rend();
    const_reverse_iterator rbegin() const;
    const_reverse_iterator rend() const;
};
```

1.6 Добавяне на елементи в края (20 точки)

Трябва да се реализират набор от методи, които добавят символи към края на вектора.

- **T& front();**
Връща препратка към първия елемент на вектора.
- **const T& front() const;**
Връща константна препратка към първия елемент на вектора.
- **T& back();**
Връща препратка към последния елемент на вектора.
- **const T& back() const;**
Връща константна препратка към последния елемент на вектора.
- **void push_back(const T&);**
Добавя нов елемент след последния елемент на вектора.
- **void pop_back();**
Изтрива последния елемент на вектора.

1.7 Вмъкване и изтриване на елементи (20 точки)

Трябва да се реализират набор от методи, които да вмъкват и изтриват елементи от вектора.

- `iterator insert(iterator pos, const T& x);`
Вмъква нов елемент във вектора преди елемента, към който е насочен итератора `pos`.
- `iterator erase(iterator pos);`
Изтрива елементът от вектора, към който е насочен итератора `pos`.
- `iterator erase(iterator first, iterator last);`
Изтрива елементите от вектора, които се намират в диапазона `[first, last)`.

1.8 Главна функция

Като се използва разработения клас `Vector` трябва да се дефинира главна функция, която да отговаря на следните условия.

1. Програмата трябва да получава входните си данни от командния ред. Командният ред, който обработва програмата трябва да изглежда по следния начин:

```
./hw02vector 1 6 15 21
```

Започвайки работа програмата трябва да създаде две вектора `v1` и `v2` от типа `Vector<int>`.

Първата двойка числа, които програмата получава като аргументи от командния ред трябва да се схващат като дефиниция на диапазон `[b1, e1)` от стойности на елементите на първия вектор, а втората двойка — като диапазон `[b2, e2)` от стойности на елементите на втория вектор.

В разгледания пример елементите на вектора `v1` трябва да са `{1, 2, 3, 4, 5}`, а елементите на вектора `v2` са `{15, 16, 17, 18, 19, 20}`.

2. Да изведе на стандартния изход стойностите на елементите на двата вектора:

```
v1: {1,2,3,4,5,}  
v2: {15,16,17,18,19,20,}
```

3. Да обходи елементите на двата вектора като използва класа `iterator` и да преброи елементите, които присъстват в двата вектора:

```
equal element in v1 and v2: 0
```

4. Като използва метода `push_back()` да добави елемента `-100` в края на двата вектора и да ги изведе:

```
v1: {1,2,3,4,5,-100,}  
v2: {15,16,17,18,19,20,-100,}
```

5. Като използва копиращ конструктор да създаде нов вектор `v`, който да е копие на вектора `v2`:

```
v: {15,16,17,18,19,20,-100,}
```

6. Като използва `reverse_iterator` да обходи всички елементи на вектора `v1` и да ги добави в началото на вектора `v`. Резултатният вектор трябва да има вида:

```
v: {1,2,3,4,5,-100,15,16,17,18,19,20,-100,}
```

7. Да създаде итератор `bit` и да го насочи към първия елемент със стойност `-100` на вектора `v`. Като се използва метода `erase()` да се изтрият всички елементи на вектора `v`, които са след елемента, към който е насочен итератора `bit`. Резултатният вектор трябва да има вида:

```
v: {1,2,3,4,5,}
```

2 Изисквания към решението и оценяване

1. Решението на задачата трябва да бъде написано на езика C++ съгласно ISO/IEC 14882:1998.
2. Правилата за оценяване са следните. Приемаме, че напълно коректна и написана спрямо изискванията програма получава максималния брой точки — 100% или 120 точки. Ако в решението има пропуски, максималният брой точки ще бъде намален съгласно правилата описани по-долу.
3. Задължително към файловете с решението трябва да е приложен и `Makefile`. Изпълнимият файл, който се създава по време на компилация на решението, трябва да се казва `hw02vector`.

4. При проверка на решението програмата ви ще бъде компилирана и тествана по следния начин:

```
make
./hw02vector 5 8 5 11
```

Предходната процедура ще бъде изпълнена няколко пъти с различни входни данни за да се провери дали вашата програма работи коректно.

5. Реализацията на програмата трябва да спазва точно изискванията. Всяко отклонение от изискванията ще доведе до получаване на 0 точки за съответната част от условието.
6. Работи, които са предадени по-късно от обявеното (или не са предадени), ще бъдат оценени с 0 точки.
7. Програмата ви трябва да съдържа достатъчно коментари. Оценката на решения без коментари или с недостатъчно и/или мъгляви коментари ще бъде намалена с 30%.

8. Всеки файл от решението трябва да започва със следният коментар:

```
//-----  
// NAME: Ivan Ivanov  
// CLASS: XIa  
// NUMBER: 13  
// PROBLEM: #1  
// FILE NAME: xxxxxx.yyy.zzz (unix file name)  
// FILE PURPOSE:  
// няколко реда, които описват накратко  
// предназначението на файла  
// ...  
//-----
```

Всяка функция във вашата програма трябва да включва кратко описание в следния формат:

```
//-----  
// FUNCTION: ххууzz (име на функцията)  
// предназначение на функцията  
// PARAMETERS:  
// списък с параметрите на функцията  
// и тяхното значение  
//-----
```

9. Лош стил на програмиране и липсващи заглавни коментари ще ви костват 30%.
10. Програми, които не се компилират получават 0 точки. Под “не се компилират” се има предвид произволна причина, която може да причини неуспешна компилация, включително липсващи файлове, неправилни имена на файлове, синтактични грешки, неправилен или липсващ `Makefile`, и т.н. Обърнете внимание, че в UNIX имената на файловете са “case sensitive”.
11. Програми, които се компилират, но не работят, не могат да получат повече от 50%. Под „компилира се, но не работи“ се има предвид, че вие сте се опитали да решите проблема до известна степен, но не сте успели да направите пълно решение. Често срещан проблем, който спада към този случай, е че вашият `Makefile` генерира изпълним файл, но той е именуван с име, различно от очакваното (т.е. `hw02vector` в разглеждания случай).
12. Безсмислени или мъгляви програми ще бъдат оценявани с 0 точки, независимо че се компилират.
13. Програми, които дават неправилни или непълни резултати, или програми, в които изходът и/или форматиранието се различава от изискванията ще получат не повече от 70%.
14. Всички наказателни точки се сумират. Например, ако вашата програма няма задължителните коментари в началото на файлове и функциите се отнемат 30%, ако няма достатъчно коментари се отнемат

още 30%, компилира се, но не работи правилно — още 30%, то тогава резултатът ще бъде: $120 * (100 - 30 - 30 - 30)\% = 120 * 10\% = 12$ точки

15. Работете самостоятелно. Групи от работи, които имат твърде много прилики една с друга, ще бъдат оценявани с 0 точки.