

С++

Лекция 4

Контейнеры

Объявление/реализация

Triangle.h
Объявление

```
class Triangle {  
private:  
    double a, b, c;  
  
public:  
    Triangle();  
    Triangle(double a_in,  
              double b_in,  
              double c_in);  
    double area() const;  
    double perimeter() const;  
    void scale(double s);  
};
```

Triangle.cpp
Реализация

```
#include "Triangle.h"  
  
Triangle::Triangle(double a_in,  
                   double b_in, double c_in)  
    : a(a_in), b(b_in), c(c_in) { }  
  
void Triangle::scale(double s) {  
    this->a *= s;  
    this->b *= s;  
    this->c *= s;  
}
```

Создание контейнера

- **Контейнер** - абстрактный тип данных для хранения объектов.
- Примеры:
 - arrays
 - vector
- Создадим свой контейнер: ***IntSet***
 - Set (множество) - набор уникальных объектов, объекты никак не отсортированы. IntSet - множество состоящее из целых чисел.

Что можно делать с множеством?

- Добавлять объект
- Удалять объект
- Проверить содержит ли множество объект
- Узнать количество элементов в множестве
- Вывести множество на экран

ИСПОЛЬЗОВАНИЕ МНОЖЕСТВА

- Примеры использования IntSet.

```
int main() {  
    IntSet set;  
    set.insert(7);  
    set.insert(32);  
  
    cout << "Size: " << set.size() << endl;  
    set.print(cout);  
  
    set.insert(42);  
    set.remove(32);  
  
    cout << "Contains 32? " << set.contains(32) << endl;  
    cout << "Contains 42? " << set.contains(42) << endl;  
}
```

Объявление IntSet (IntSet.h)

```
class IntSet {  
public:  
    // Максимальный размер множества.  
    static const int ELTS_CAPACITY = 10;  
    // ТРЕБОВАНИЕ: size() < ELTS_CAPACITY  
    // РЕЗУЛЬТАТ: добавление v в мн-во  
    void insert(int v);  
  
    // РЕЗУЛЬТАТ: удаление v из мн-ва  
    void remove(int v);  
  
    // РЕЗУЛЬТАТ: возвращает true в случае v  
    содержится  
    // в множестве  
    bool contains(int v) const;  
  
    // РЕЗУЛЬТАТ: возвращает количество элементов  
    int size() const;  
  
    // РЕЗУЛЬТАТ: выводит мн-во на экран  
    void print(std::ostream &os) const;  
};
```

Здесь
представлены
ТОЛЬКО
объявления
методов класса,
реализация в
файле .cpp

consts
значит поля
класса
IntSet не
будут
изменены.

Статические поля класса

- Поле класса объявленное с ключевым словом **static** является “общим” для всех экземпляров класса
- Статические поля похожи на глобальные переменные.
 - Время жизни = времени жизни программы (как у глобальной переменной).
 - Но область видимости ограничена областью видимости класса.

```
class IntSet {  
public:  
    // Максимальный размер множества.  
    static const int ELTS_CAPACITY = 10;  
  
    ...  
};
```

Почему размер ограничен?

- Потому что в данной реализации необходимо заранее знать сколько выделять памяти.
 - Сейчас размер выделяемой памяти определяется во время компиляции. (размер массива для хранения элементов IntSet)

```
class IntSet {  
public:  
    // Максимальный размер множества  
    static const int ELTS_CAPACITY = 10;  
  
    ...  
};
```


IntSet представление данных

- Что необходимо хранить?
 - Хранить массив элементов множества.
 - Хранить размер множества.

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Все ещё
объявление.

```
class IntSet {  
    ...  
private:  
    int elts[ELTS_CAPACITY];  
    int elts_size;  
    ...  
};
```

Адекватность способа представления данных

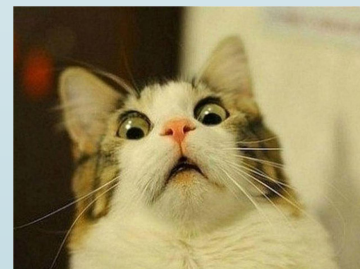
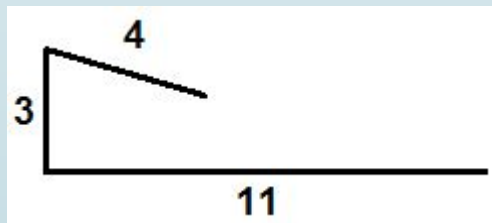
- Не всегда представление данных отвечает задумке (требованиям).
- Например:
Представление треугольника тремя числами с плавающей точкой.

```
class Triangle {  
    double a;  
    double b;  
    double c;  
};
```

- Пример:
Представление треугольника тремя числами с плавающей точкой.

```
class Triangle {  
    double a;  
    double b;  
    double c;  
};
```

- Проблема:
Далеко не любые три числа могут использоваться для представления треугольника.



Инварианты представления

- Проблема составных типов данных...
 - Некоторые комбинации значений полей недопустимы.
- Для валидации используются **инварианты представления**.
- Для треугольника:

Длина ребер
положительная

$$0 < a$$

$$0 < b$$

$$0 < c$$

Неравенства
треугольника

$$a + b > c$$

$$a + c > b$$

$$b + c > a$$

Условие, которому удовлетворяют все валидные объекты, называется **инвариантом представления**.



13

Exercise: Инварианты представления

- Какие инварианты представления нужны для класса IntSet?

```
class IntSet {  
private:  
    int elts[ELTS_CAPACITY];  
    int elts_size;  
    ...  
};
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: Инварианты представления

- Какие инварианты представления нужны для класса IntSet?

```
class IntSet {  
private:  
    int elts[ELTS_CAPACITY];  
    int elts_size;  
};
```

Валидация размера

$0 \leq \text{elts_size}$
 $\text{elts_size} \leq \text{ELTS_CAPACITY}$

Валидация элементов

Нет дублирующихся
элементов.

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |


elts_size

4

IntSet конструктор

- Необходимо удостовериться что инварианты представления заданы верно и обеспечивают валидность данных.

```
class IntSet {  
public:  
    IntSet();  
    ...  
};
```



Объявление (в .h
файле) реализация в
.cpp файле.



16

Exercise: IntSet::contains

- Напишите реализацию функции contains.

```
bool IntSet::contains(int v) const {  
    // CODE  
}
```

Реализация в
файле
IntSet.cpp

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: IntSet::contains

Реализация функции contains.

```
bool IntSet::contains(int v) const {  
    for (int i = 0; i < elts_size; ++i) {  
        if (elts[i] == v) {  
            return true;  
        }  
    }  
    return false;  
}
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Повторное использование кода

- Реализация функций **remove** и **contains** содержит код для поиска элемента.
- Этот код лучше вынести в отдельную функцию.

```
int IntSet::indexOf(int v) const {  
    for (int i = 0; i < elts_size; ++i) {  
        if (elts[i] == v) {  
            return i;  
        }  
    }  
    return -1;  
}
```

Если элемент найден,
вернется индекс элемента

В противном случае вернется -1.



19

Exercise: IntSet реализация

- Напишите реализацию оставшихся функций IntSet.

```
void IntSet::insert(int v) {  
    // CODE  
}  
void IntSet::remove(int v) {  
    // CODE  
}  
bool IntSet::contains(int v) const {  
    // CODE  
}
```

Используйте
indexOf для
реализации
remove и
contains.

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: IntSet реализация

```
void IntSet::insert(int v) {  
    assert(size() <ELTS_CAPACITY);  
    if (contains(v)) {  
        return;  
    }  
    elts[elts_size] = v;  
    ++elts_size;  
}
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: IntSet реализация

```
void IntSet::remove(int v) {  
    if (!contains(v)) {  
        return;  
    }  
    elts[indexOf(v)] = elts[elts_size - 1];  
    --elts_size;  
}
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: IntSet реализация

```
bool IntSet::contains(int v) const {  
    return indexOf(v) != -1;  
}
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 3 | 5 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

SortedIntSet

- Рассмотрим другое представление данных множества:
 - Хранить **отсортированный** массив элементов.
 - Хранить количество элементов.

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 6 | 7 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

```
class SortedIntSet {  
private:  
    int elts[ELTS_CAPACITY];  
    int elts_size;  
};
```

Инварианты представления

- Какие инварианты представления необходимы для класса SortedIntSet?

```
class SortedIntSet {  
private:  
    int elts[ELTS_CAPACITY];  
    int elts_size;  
};
```

Валидация размера

$0 \leq \text{elts_size}$
 $\text{elts_size} \leq \text{ELTS_CAPACITY}$

Валидация элементов

Нет дублирующихся элементов. Элементы отсортированы

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 6 | 7 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

10/12/2016



25

Exercise: SortedIntSet

- Напишите реализацию SortedIntSet.

```
void SortedIntSet::insert(int v) {  
    // CODE  
}  
void SortedIntSet::remove(int v) {  
    // CODE  
}  
bool SortedIntSet::contains(int v) const {  
    return indexOf(v) != -1;  
}  
int SortedIntSet::indexOf(int v) const {  
    // уже реализовано  
};
```

Используйте
indexOf для
реализации
remove и
contains.

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 6 | 7 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: SortedIntSet::insert

```
void SortedIntSet::insert(int v) {  
    assert(size() <ELTS_CAPACITY);  
    if (!contains(v)) {  
        int index = elts_size;  
        while (index > 0 && elts[index - 1] > v) {  
            elts[index] = elts[index - 1];  
            --index;  
        }  
        elts[index] = v;  
        ++elts_size;  
    }  
}
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 6 | 7 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Solution: SortedIntSet::remove

```
void SortedIntSet::remove(int v) {  
    if (!contains(v)) {  
        return;  
    }  
    for (int i = indexOf(v); i < elts_size - 1; ++i) {  
        elts[i] = elts[i + 1];  
    }  
    --elts_size;  
}
```

elts

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 6 | 7 | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

4

Преимущества сортировки

- Реализация `indexOf` может использовать преимущества отсортированного массива.
- Можно использовать бинарный поиск, который быстрее линейного.

```
int SortedIntSet::indexOf(int v) const {  
    //  
};
```

elts

| | | | | | | | | | |
|----|----|----|---|---|---|---|---|----|----|
| -7 | -5 | -1 | 0 | 2 | 4 | 5 | 8 | 11 | 17 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

elts_size

10

IntSet Efficiency

- Какова сложность каждой операции?

| | IntSet | SortedIntSet |
|--------------------|--------|--------------|
| insert | $O(n)$ | $O(n)$ |
| remove | $O(n)$ | $O(n)$ |
| contains | $O(n)$ | $O(\log n)$ |
| size | $O(1)$ | $O(1)$ |
| constructor | $O(1)$ | $O(1)$ |