# Научно-исследовательский практикум

СТРУКТУРЫ И КЛАССЫ. КОНСТРУКТОР-КОПИРОВЩИК.

# Реализация контейнера из предыдущего занятия

Метод Add() разделился на 2 части

Первая отвечает за добавление элемента

Вторая – за изменение размера

```
class SimpleContainer
{
public:
        SimpleContainer();
        SimpleContainer(int size, double value=0);
        ~SimpleContainer();

        int size() const {return m_size;}
        int capacity() const {return m_capacity;}
        double get(int index) const;
        void set(int index, double value);
        void Add(double value);

private:
        void Reallocate();

        double* m_data;
        int m_size; // логический размер
        int m_capacity; // фактический размер
};
```

# Реализация контейнера из предыдущего занятия

m_data[m_size++] = value;

то же самое, что и

m_data[m_size] = value;

m_size++;

Здесь вместо того, чтобы копировать элементы по одному, мы воспользуемся функцией std::copy

```
void SimpleContainer::Add(double value)
{
        if (m_size==m_capacity)
                Reallocate();
        m_data[m_size++] = value;
}

void SimpleContainer::Reallocate()
{
        cout << "Now we need to reallocate some memory" << endl;
        int newCapacity = m_capacity*2;
        double* newData = new double [newCapacity];
        std::copy(m_data, m_data+m_capacity, newData);
        m_capacity = newCapacity;
        delete[] m_data;
        m_data = newData;
}
```

# Реализация контейнера из предыдущего занятия

Кроме того, я немного изменю конструктор по-умолчанию

Давайте посмотрим, что получилось…

```
SimpleContainer::SimpleContainer()
{
        m_size = 0;
        m_capacity = 2;
        m_data = new double[m_capacity];
}

int main(int argc, char *argv[])
{
        SimpleContainer testContainer;
        for (int i=0; i<5; i++)
        {
                testContainer.Add(i);
                cout << "Size is " << testContainer.size() \
                << " and capacity is " <<
testContainer.capacity() << endl;
        }
        for (int i=0; i<testContainer.size(); i++)
                cout << "Container[" << i <<"]=" <<
testContainer.get(i) << endl;
}
```

# Результат

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
Size is 1 and capacity is 2
Size is 2 and capacity is 2
Now we need to reallocate some memory
Size is 3 and capacity is 4
Size is 4 and capacity is 4
Now we need to reallocate some memory
Size is 5 and capacity is 8
Container[0]=0
Container[1]=1
Container[2]=2
Container[3]=3
Container[4]=4
/Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj exited with code 0
```

# Задание

Реализуйте в классе 3 метода:

1. erase() – удаляет все элементы, создаёт пустой контейнер
2. reset(int size, double value) – изменяет размер контейнера на size, все элементы равны value
3. resize(int size) – изменяет размер контейнера на size
   - Если элементов было больше – просто «обрезает» контейнер
   - Если элементов было меньше – инициализирует новые элементы каким-либо начальным значением

# Неожиданная проблема…

```
int main(int argc, char *argv[])
{
	SimpleContainer testContainer;
	for (int i=0; i<5; i++)
		testContainer.Add(i);
	SimpleContainer test2 = testContainer;
	test2.set(2, -1);
	cout << testContainer.get(2) << endl;
}
```

# Неожиданная проблема…

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
test_proj(4010,0x7fff7c640000) malloc: *** error for object 0x7fc3a8c001d0: pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
Now we need to reallocate some memory
Now we need to reallocate some memory
-1
The program has unexpectedly finished.
/Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj crashed.
```

# Правило трёх... или пяти?

- В C++ для любого класса (в *широком* смысле)есть некоторое количество функций-членов, которые в нём обязательно есть
- Даже если их не реализовать – компилятор сделает это за вас
- Эти три функции –деструктор, конструктор-копировщик и оператор присваивания
- А если уже вам пришлось реализовать хотя бы одну из этих функций – то надо реализовывать и две другие
- Это – так называемое «Правило трёх» (Закон трёх, The Rule of three, Law of three, Big three etc)
- С 2011 стандарта появились две дополнительные функции для move-механики – поэтому сейчас говорят о Правиле пяти

# Первая функция - деструктор

```
SimpleContainer::~SimpleContainer()
{
    delete[] m_data;
}
```

Раз мы выделяли память в конструкторе, то теперь её нужно где-то высвободить

# Нулевая функция - конструктор

- Конструктор по-умолчанию – это конструктор без параметров, который вызывает конструкторы по умолчанию все переменных класса (и базового класса, но об этом позже)
- Его можно просто не создавать – компилятор сделает это за вас
- Или создать вот такой:

```
SomeClass()
{
    // Ничего не пишем
}
```

- Или указать компилятору создать такой:

```
SomeClass() = default;
```

- На самом деле все эти варианты разные – но различие между ними уже на уровне Pro

# Вторая функция– конструктор-копировщик

- Это тоже конструктор, но специальный – он позволяет создать копию объекта того же класса
- Его тоже можно не создавать
- Или указать компилятору создать такой

```
SomeClass(SomeClass&) = default;
```

- Но!!! Если вы создадите его руками

```
SomeClass(SomeClass)
{
// То здесь нужно скопировать всё вручную
}
```

# Первый пример конструктора-копировщика

Здесь использован автоматически создаваемый

```
Starting /Users/amakashov/
1685807688 10 0
1685807688 10 123
/Users/amakashov/projects/
```

```
class SomeClass
{
public:
        SomeClass() = default;
        SomeClass(SomeClass&) = default;

        int a;
        double b = 10; // Так можно только в C++11
        double c;
};


int main(int argc, char *argv[])
{
        SomeClass first;
        cout << first.a << " " << first.b << " " \
<< first.c << endl;
        first.c=123;
        SomeClass second(first);
        cout << second.a << " " << second.b \
<< " " << second.c << endl;
}
```

# Второй пример конструктора-копировщика

А здесь – созданный нами вручную. Он копирует только одну переменную

```
Starting /Users/amakashov/
1771713096 25 123
1 10 123
/Users/amakashov/projects/
```

```
class SomeClass
{
public:
        SomeClass() = default;
        SomeClass(SomeClass& other)
        {
                c = other.c;
        }
        int a;
        double b = 10; // Так можно только в C++11
        double c;
};


int main(int argc, char *argv[])
{
        SomeClass first;
        first.b = 25;
        first.c=123;
        cout << first.a << " " << first.b << " " << \
first.c << endl;
        SomeClass second(first);
        cout << second.a << " " << second.b << " " << \
second.c << endl;
}
```

# Третья функция – оператор присваивания

- Это то, что происходит при выполнении строчки

```
SomeClass second = first;
```

- Он тоже может быть дефолтным (по умолчанию)
- Тогда просто копируются значения всех полей
- При этом сначала важно, кому мы делаем присваивание

# Присваиваем уже созданному экземпляру

При этом сначала вызывается конструктор, а потом оператор присваивания

```
Starting /Users/amakashov/p
1768649288 25 123
1768649288 25 123
/Users/amakashov/projects/b
```

```
class SomeClass
{
public:
        SomeClass() = default;
        SomeClass(SomeClass& other)
        {
                c = other.c;
        }

        int a;
        double b = 10; // Так можно только в C++11
        double c;
};

int main(int argc, char *argv[])
{
        SomeClass first;
        first.b = 25;
        first.c=123;
        cout << first.a << " " << first.b << " " <<
first.c << endl;
        SomeClass second;
        second = first;
        cout << second.a << " " << second.b << " " <<
second.c << endl;
}
```

# Здесь мы присваиваем создаваемому классу

При этом вызывается сначала оператор присваивания, а затем конструктор-копировщик

```
Starting /Users/amakashov/
1637241416 25 123
1 10 123
/Users/amakashov/projects/
```

```
class SomeClass
{
public:
        SomeClass() = default;
        SomeClass(SomeClass& other)
        {
                c = other.c;
        }

        int a;
        double b = 10; // Так можно только в C++11
        double c;
};

int main(int argc, char *argv[])
{
        SomeClass first;
        first.b = 25;
        first.c=123;
        cout << first.a << " " << first.b << " " <<
first.c << endl;
        SomeClass second = first;
        cout << second.a << " " << second.b << " " <<
second.c << endl;
}
```

# Причина наших проблемм с контейнером?

- Вернёмся к слайду 7. Неожиданная проблема…
- Оператор присваивания по-умолчанию:
  - Копирует всё содержимое
  - В том числе и указатель на массив
  - А значит, у нас получаются 2 контейнера с общим массивом m_data
  - Что происходит при удалении?

# Решение

- Принцип «Большой тройки» - если вам нужен хотя бы один из его элементов – вам, скорее всего, нужны все три
- В нашем конструкторе мы выделяли память
- Поэтому нам необходим деструктор – для её высвобождения
- Но нам мало скопировать указатель на память – нам нужна копия содержимого
- Поэтому придётся создать конструктор-копировщик и оператор присваивания

# Специальный указатель `this`

- Указатель на объект, функция-член которого вызвана
- Позволяет обратиться к любому элементу объекта
- В нём содержится указатель на текущий объект
- При обращении по указателю к элементам объекта используется специальный синтаксис, вместо

```
(*this).c = 10;
```

пишется

```
this->c = 10;
```

# Конструктор-копировщик

```
SimpleContainer::SimpleContainer(const SimpleContainer &container)
{
        m_size = m_capacity = container.m_size;
        m_data = new double[m_capacity];
        std::copy(container.m_data, container.m_data+m_capacity,
m_data);
}
```

# Оператор присваивания

```
SimpleContainer& SimpleContainer::operator= (const SimpleContainer&
rhs)
{
        if (this != &rhs)
        {
                if (m_data) // Тут можно было бы проверять размер
                        delete[] m_data;
                m_size = m_capacity = rhs.m_size;
                m_data = new double[m_capacity];
                std::copy(rhs.m_data, rhs.m_data+rhs.m_size, m_data);
        }
        return *this;
}
```

# Другие операторы: сложение

```
SimpleContainer SimpleContainer::operator+ (const SimpleContainer& rhs) const
{
	SimpleContainer tmp;
	if (m_size == rhs.m_size)
	{
		for (int i = 0; i< m_size; i++)
		tmp.Add(m_data[i] + rhs.m_data[i]);
	}
	return tmp;
}
```

# Задача

Попробуйте перегрузить оператор []

```
double& SimpleContainer::operator[] (const int index)
{
}
```

- Здесь index – это то, что будет стоять в квадратных скобках
- Обратите внимание, что мы возвращаем ссылку!!!
- Второй вариант (перегруженный)

```
double SimpleContainer::operator[] (const int index) const
{
}
```