Научно-исследовательский практикум

СТРУКТУРЫ. КЛАССЫ. КЛАСС-КОНТЕЙНЕР

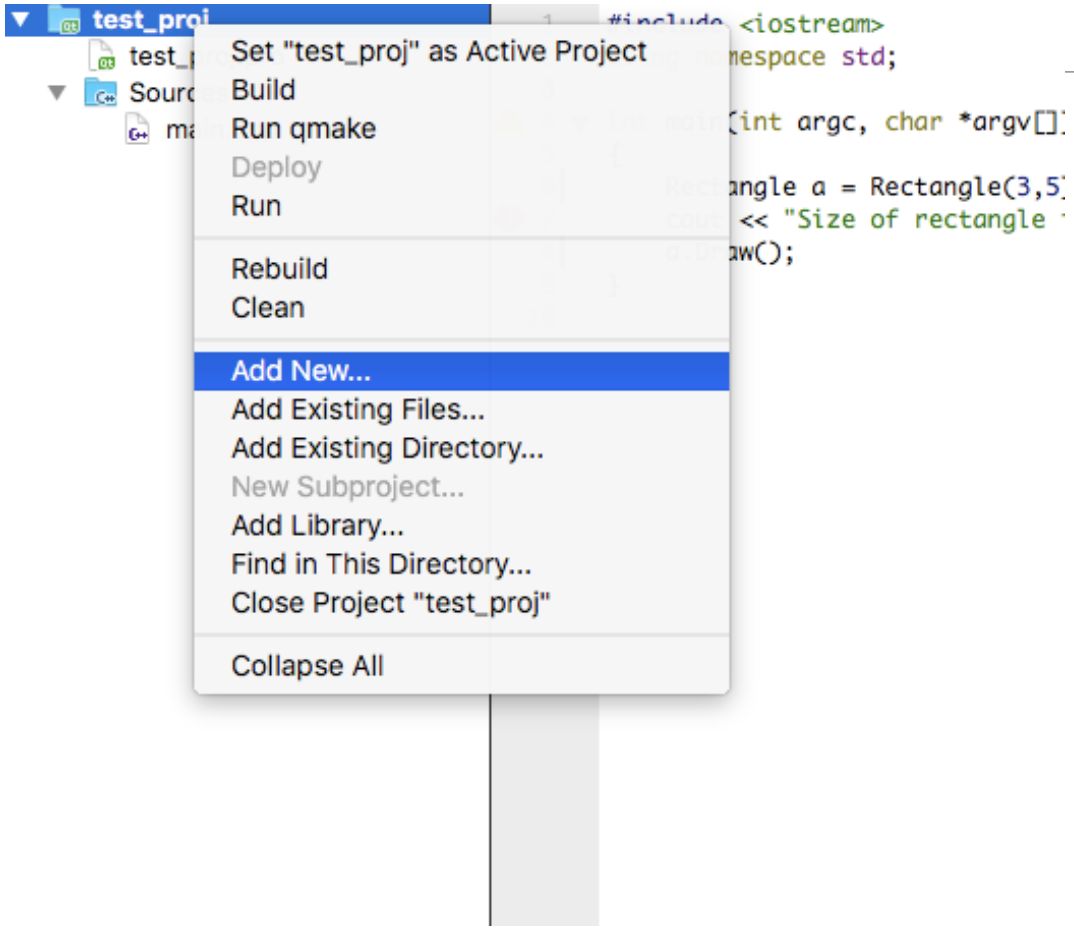
Чуть более сложный проект

По мере роста проекта писать программу, состоящую из одного файла становится неудобно

Выход – мы разбиваем наш проект на несколько файлов

Обычно реализацию классов делают в отдельном файле (точнее, 2х файлах)

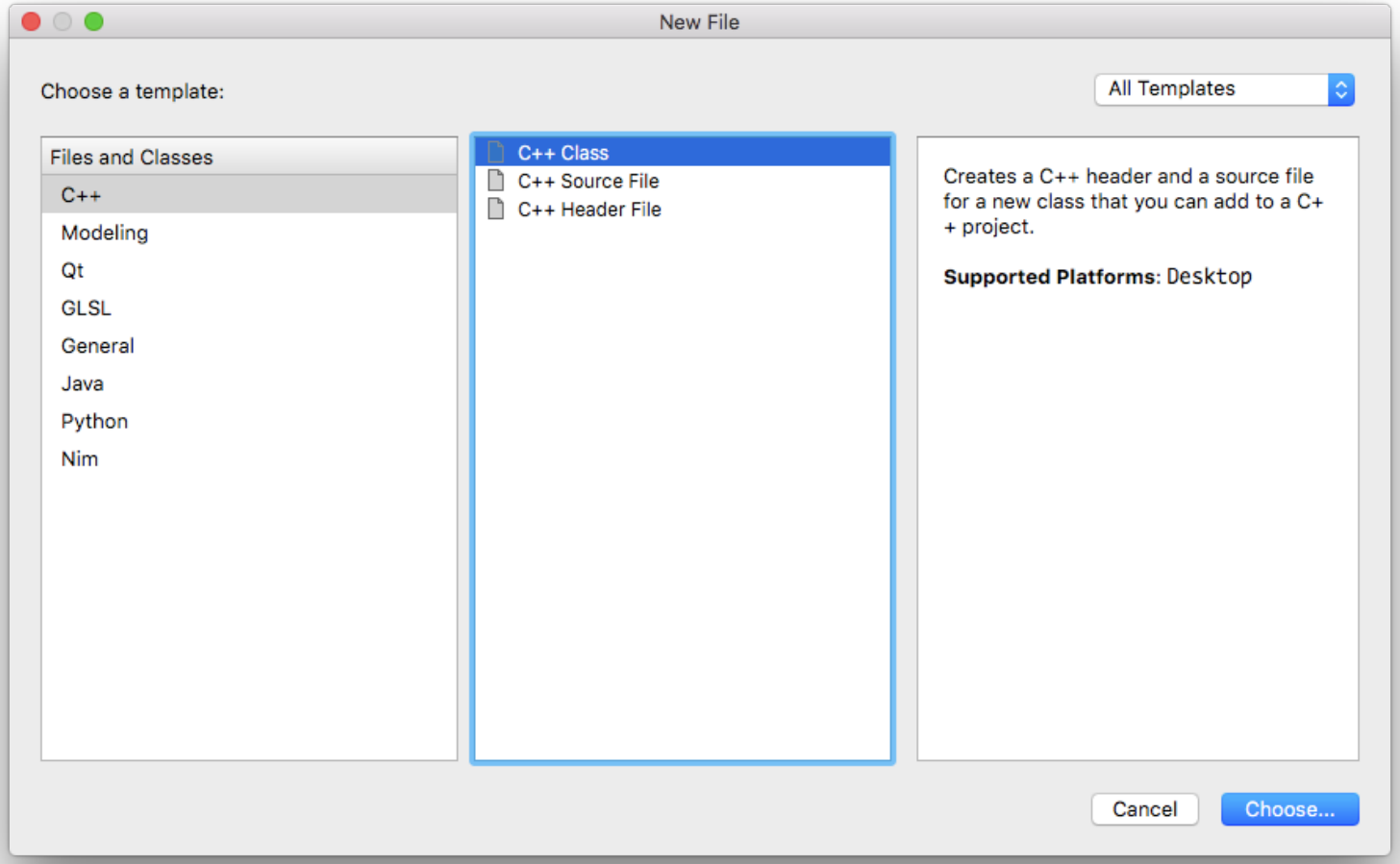
Как добавить файл или класс



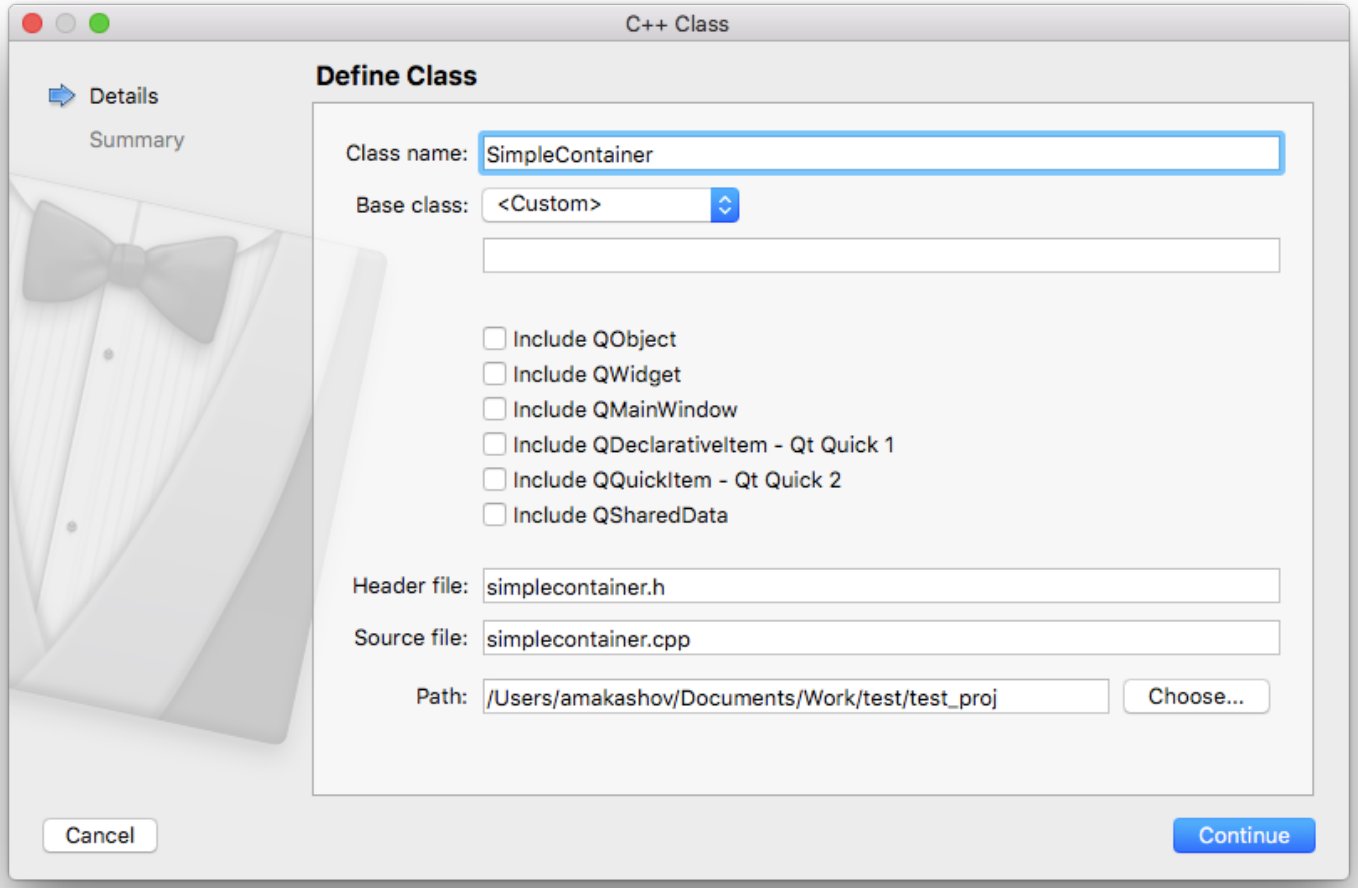
QtCreator многое сделает за вас

Однако всё это можно сделать и вручную

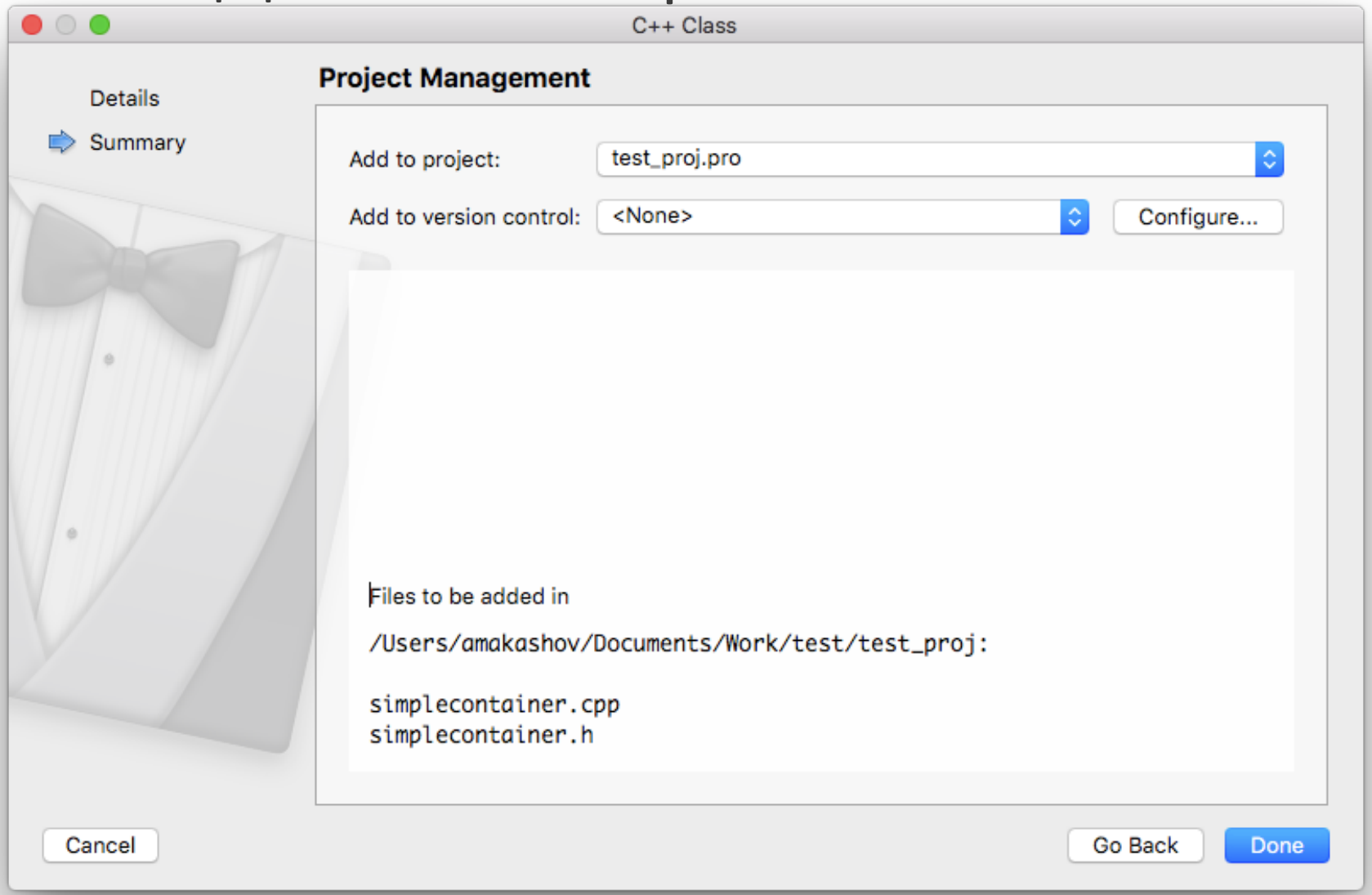
Как добавить файл или класс



Как добавить файл или класс

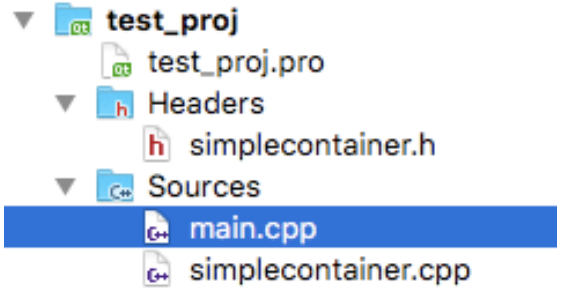


Как добавить файл или класс



Что получилось

СТРУКТУРА ПРОЕКТА



ПРОЕКТНЫЙ ФАЙЛ

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt

SOURCES += main.cpp \
    simplecontainer.cpp

HEADERS += \
    simplecontainer.h
```

Что получилось

SIMPLECONTAINER.H

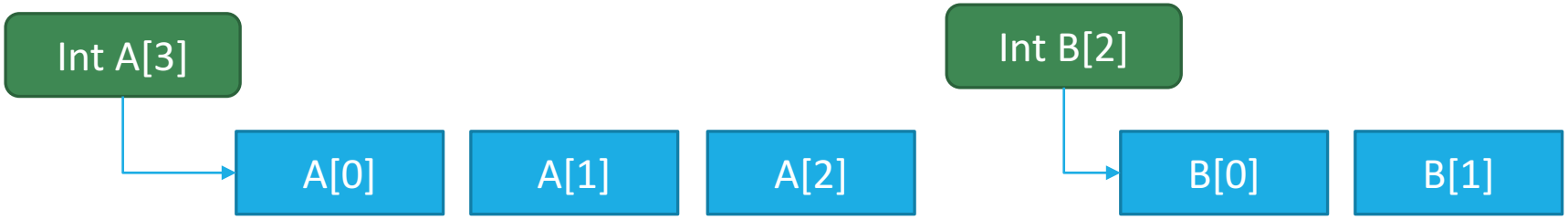
```
#ifndef SIMPLECONTAINER_H
#define SIMPLECONTAINER_H

class SimpleContainer
{
public:
     SimpleContainer();
};

#endif // SIMPLECONTAINER_H
```

SIMPLECONTAINER.CPP

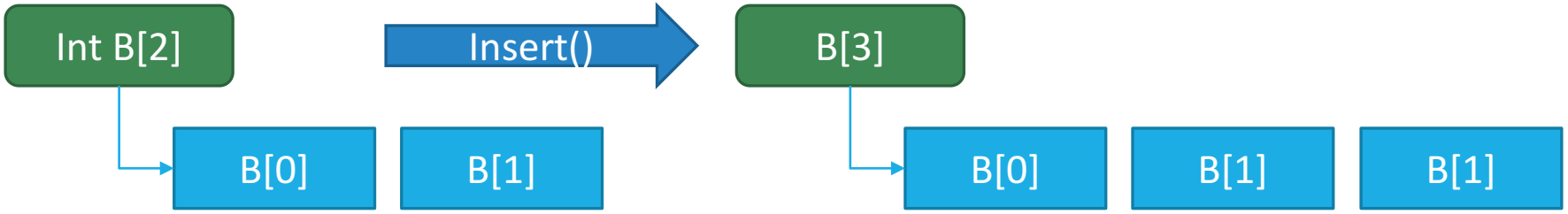
```
#include "simplecontainer.h"

SimpleContainer::SimpleContainer()
{

}
```

Что мы хотим сделать?

Реализовать массив переменного размера



Добавлять элементы в процессе выполнения программы



Начнём с создания контейнера

Будем хранить элементы типа double

Нам нужно их где-то хранить – m_data

Нам нужно знать, сколько их у нас – m_size

```
class SimpleContainer
{
public:
        SimpleContainer();

        double* m_data;
        int m_size;
};
```

Добавим конструкторы

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

        double* m_data;
        int m_size;
};
```

Эту часть мы пишем в заголовочном файле – хедере (header) simplecontainer.h

Реализацию вынесем в файл cpp

simplecontainer.cpp - реализация

```
SimpleContainer::SimpleContainer()
{
        m_data = nullptr; // nullptr – указатель на ноль
        m_size = 0;
}

SimpleContainer::SimpleContainer(int size, double value)
{
        m_data = new double[size];
        for (int i=0; i<size; i++)
                m_data[i]=value;
        m_size = size;
}
```

Деструктор и динамическая память

- Раз мы выделили память – её нужно где-то высвобождать
- Деструктор – вызывается при удалении нашего контейнера
- В файле simplecontainer.h

```
~SimpleContainer();
```

- В файле simplecontainer.cpp

```
SimpleContainer::~SimpleContainer()
{
        if (m_data)  //     mdata != 0
                delete[] m_data;
        m_data = nullptr;
}
```

Результат

```
int main(int argc, char *argv[])
{
        SimpleContainer testContainer;
        SimpleContainer testContainer2(3, 3.14);
        for (int i=0; i< testContainer2.m_size; i++)
                cout << testContainer2.m_data[i] << "\t";
        cout << endl;
        cout << "Something totaly unpredictable: " <<
testContainer.m_data[2] << endl;
}
```

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
3.14        3.14        3.14
The program has unexpectedly finished.
/Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj crashed.
```

Попробуем изменить возможность доступа

```
class SimpleContainer
{
public:
      SimpleContainer();
      SimpleContainer(int size, double value=0);
      ~SimpleContainer();
      int size() const {return m_size;}
      double get(int index) const;
      void set(int index, double value);
private:
      double* m_data;
      int m_size;
};
```

Реализация метода get()

```
double SimpleContainer::get(int index) const
{
	if (index>=0 && index<m_size)
		return m_data[index];
	else
	{
		std::cerr << "Wrong index " << index << endl;
		return 0;
	}
}
```

Реализация метода set()

```
void SimpleContainer::set(int index, double
value)
{
	if (index>=0 && index<m_size)
		m_data[index] = value;
	else
		std::cerr << "Wrong index " << index
<< endl;
}
```

Результат

```
int main(int argc, char *argv[])
{
        SimpleContainer testContainer;
        SimpleContainer testContainer2(3, 3.14);
        for (int i=0; i< testContainer2.size(); i++)
                cout << testContainer2.get(i) << "\t";
        cout << endl;
        cout << "Something totaly unpredictable: " <<
testContainer.get(2) << endl;
}
```

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
3.14        3.14        3.14
Something totaly unpredictable: 0
Wrong index 2
/Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj exited with code 0
```

А как динамически увеличить размер?

МОЖНО БЫЛО БЫ СРАЗУ ЗАРЕЗЕРВИРОВАТЬ ОЧЕНЬ МНОГО ПАМЯТИ…

ИЛИ ПРОСТО КАЖДЫЙ РАЗ СОЗДАВАТЬ НОВЫЙ ОБЪЕКТ НУЖНОГО РАЗМЕРА

А как динамически увеличить размер?

- А что, если мы будем изменять внутри класса размер m_data?
- В этом случае у нас остаётся один и тот же объект
- При этом происходящие изменения не видны за ределами класса

Как это сделать

```
void SimpleContainer::Add(double value)
{
        cout << "Container size is "<< m_size << " and we're going to
add value " << value << endl;
        int newSize = m_size+1;
        double* newData = new double [newSize];
        for (int i=0; i<m_size; i++)
                newData [i] = m_data[i];
        newData[m_size] = value;
        m_size = newSize;
        delete[] m_data;
        m_data = newData;
}
```

Основная программа

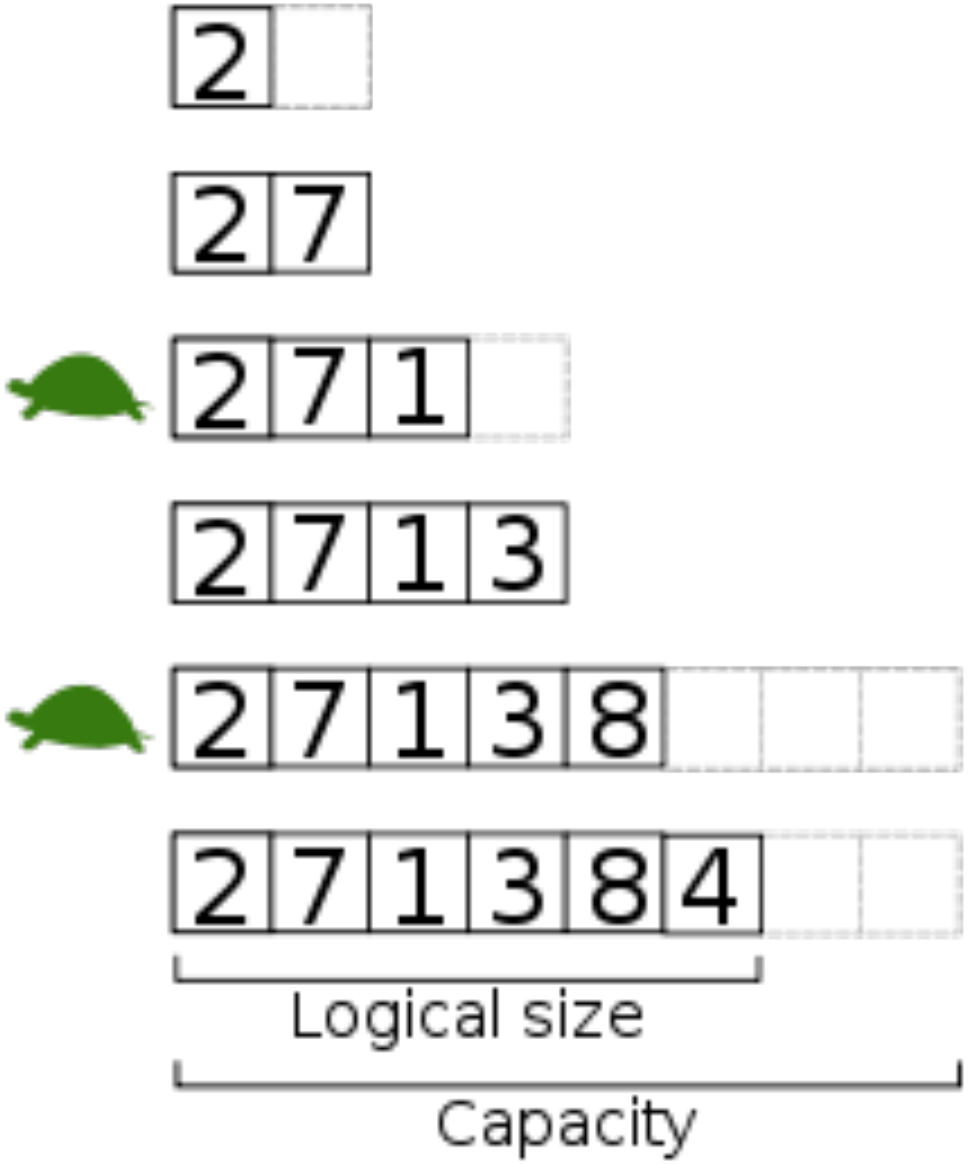
```
int main(int argc, char *argv[])
{
	SimpleContainer testContainer;
	for (int i=0; i<5; i++)
		testContainer.Add(i);
	for (int i=0; i<testContainer.size(); i++)
		cout << "Container[" << i <<"]=" <<
testContainer.get(i) << endl;
}
```

Результат

```
Starting /Users/amakashov/Documents/Work/test/build-test_proj-Desktop-Debug/test_proj...
Container size is 0 and we're going to add value 0
Container size is 1 and we're going to add value 1
Container size is 2 and we're going to add value 2
Container size is 3 and we're going to add value 3
Container size is 4 and we're going to add value 4
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
```

Нужно ли выделять память каждый раз?

- Выделение памяти и копирование – довольно затратный процесс
- Проще выделять памяти несколько больше, чем нужно
- Тогда «медленная» операция происходит реже



Реализация такого класса

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 такого класса

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 такого класса

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.
```

Причина

- Оператор присваивания по-умолчанию:
  - Копирует всё содержимое
  - В том числе и указатель на массив
  - А значит, у на получаются контейнера с общим массивом m_data
  - Что происходит при удалении?

Решение

- C++ позволяет «перегрузить» стандартные операторы
- Конструкторы – можно создать копию объекта

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 []