02393 Programming in C++
Module 7: Classes and Objects II

Sebastian Mödersheim

October 10, 2016

Lecture Plan

#	Date	Topic
1	29.8.	Introduction
2	5.9.	Basic C++
3	12.9.	Data Types, Pointers
4	19.9.	
5	26.9.	Libraries and Interfaces; Containers
6	3.10.	Classes and Objects I
7	10.10.	Classes and Objects II
		Efterårsferie
8	24.10.	Classes and Objects III
9	31.10.	Recursive Programming
10	7.11.	Lists
11	14.11.	Trees
12	21.11.	Novel C++ features
13	28.11.	Summary
	5.12.	Exam

Last week: Dictionary without and with OOP

```
struct map{

  vector<string> keys;
  vector<string> entries;
};

string find(
  const map &d,
  string key);
void insert(map &d,
  string key,
  string entry);
//
```

```
class map{
private:
  vector<string> keys;
  vector<string> entries;

public:
  string find(
    //map &d,
    string key) const;
  void insert(//map &d,
    string key,
    string entry);
};
```

OOP Basics—Summary

- A class consists of
  - ★ a record (similar to `struct`) of member variables
  - ★ methods: functions that work on one such a record.
- Object: instance of a class. Basically just a block of memory to hold one record of all member variables.
- Typically, methods are public, variables are private.
  - ★ Allows for ADTs/data encapsulation:: the user of a class cannot directly manipulate variables, but only call functions.
  - ★ Class implementation can change without changing the program that uses the class.
- Some special methods:
  - ★ Constructor: called when an object is created, i.e.
    - ▶ as a parameter or local variable of a function
    - ▶ or when created with new
  - ★ Destructor: called when an object is deallocated, i.e.
    - ▶ when a function finishes, and thus the scope of all its local variables and parameters ends
    - ▶ or when calling `delete` for an object created with `new`.
  - ★ Later: copy constructor and assignment operator

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...
- Example for many features of OOP/C++ and programming

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...
- Example for many features of OOP/C++ and programming
  - ★ Templates

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...
- Example for many features of OOP/C++ and programming
  - ★ Templates
  - ★ Copy constructor and assignment operator

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...
- Example for many features of OOP/C++ and programming
  - ★ Templates
  - ★ Copy constructor and assignment operator
  - ★ Overloading the standard operators

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...
- Example for many features of OOP/C++ and programming
  - ★ Templates
  - ★ Copy constructor and assignment operator
  - ★ Overloading the standard operators
  - ★ Dynamic memory allocation

Live Programming: Implementing the Vector Class

The vector class from the standard template library is a very convenient variant of the old **array**.
Its actual implementation is quite complex:
http://www.sgi.com/tech/stl/stl_vector.h
We now step by step develop our own version `vec`:

- Helps to understand the usage of the `vector` class!
- Helps to understand other container classes like `map`, `set`, ...
- Example for many features of OOP/C++ and programming
  - ★ Templates
  - ★ Copy constructor and assignment operator
  - ★ Overloading the standard operators
  - ★ Dynamic memory allocation
  - ★ Iterators

Copying vectors

- What will happen in the following code snippet?
- How are vectors copied?

```
vec f(vec v){...}
int main(){
  vec v1;
  ...
  vec v2=v1;
  ...
  v1=v2;
  ...
  v2=f(v1);
}
```

Copying vectors

- What will happen in the following code snippet?
- How are vectors copied?

```
vec f(vec v){...}
int main(){
  vec v1;
  ...
  vec v2=v1;
  ...
  v1=v2;
  ...
  v2=f(v1);
}
```

- Default behavior: C++ makes a copy of the member variables.

Copying vectors

- What will happen in the following code snippet?
- How are vectors copied?

```
vec f(vec v){...}
int main(){
  vec v1;
  ...
  vec v2=v1;
  ...
  v1=v2;
  ...
  v2=f(v1);
}
```

- Default behavior: C++ makes a copy of the member variables.
- How can we change that default behavior?

Copy Constructor

- Defining a copy constructor for class `vec`.
  `vec(const vec & v)`
  - ★ additional constructor that constructs a vector given an existing one.
  - ★ General form `classname(const classname & v)`
- Will be called:

```
vec f(vec v){...}
  //here for v (with argument v1 from main)
int main(){
  vec v1;
  ...
  vec v2=v1; // Here for v2 (with argument v1)
  ...
  v1=v2;
  ...
  v2=f(v1);
}
```

Assignment Operator

- Defining an assignment operator for class `vec`.

  `vec & operator=(const vec & v)`

  - ★ The "overwrite" the present vector with vector `v`.
  - ★ Result: reference to the present vector
  - ★ Pitfalls:
    - Check for self-assignment (so that `v=v;` does not crash)
    - Remember to de-allocate any allocated space of the old vector before overwriting variables.
  - ★ In general: `classname & operator=(const classname & v)`
- Will be called:

```
vec f(vec v){...}
int main(){
  vec v1;
  vec v2=v1;
  v1=v2; // Here for v1 (with argument v2)
  v2=f(v1); //Here for v2 (with the result
}           // of f(v1) as argument)
```

Abstract collection

- Dynamic size.
- Access through `operator[]`.
- Iterators (`begin` and `end`).
- Entries ordered from first to last.
- Traversing entries with `i++` and `i--`.
- Access entry with `*i`.