02393 Programming in C++
Module 6: Classes and Objects I

Sebastian Mödersheim

October 3, 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

Recap

- Dynamic Allocation
- Containers: vectors, stacks, ...
- Strings
- File I/O

The ++ in C++

- So far: basically C with few elements of C++
  - ★ `string, cout, int &i,...`
- C++ features for abstraction: ADTs and OOP
  - ★ Classes, Inheritance, Templates
- The extensions of C++ do not increase the set of expressible algorithms.
- So why bother?

Why Abstraction?

- People working independently on a large program
  - ★ using libraries: IrrLicht, STL, ...
  - ★ base-example

  ... they cannot possibly all talk with each other.
- Even working alone, a good structure is helpful!

We need clearly defined interfaces.

Problem of today's lecture: abstraction mechanisms are themselves a bit abstract and hard to appreciate at first.

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 to realize ADTs: the user of a class cannot directly manipulate variables, but only call functions. Aka data encapsulation
  - ★ We can change the implementation without changing the calling program.
- 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

Example: A Dictionary

Danish-English Dictionary

rød	red
grøn	green
blå	blue
gul	yellow
sort	black
hvid	white
grå	gray
...	

- Main operation: for a given keyword (e.g. Danish word) look up entry (e.g. English word).
- Inverse operation usually not (directly) supported (requires a English-to-Danish dictionary).
- Using sorting by keyword to make look-up efficient

Abstract Data Types

- Abstract from implementation details (e.g. keyword-sorted array)
- Describe operations on ADT.
- ADTs can only be constructed, accessed, and manipulated using these operations.
- Programs that uses the ADT do not need to be changed when the ADT's implementation is changed.

Map as an Abstract Data Type

Operations for ADT Map:

- $emptyMap : map$.
- $insert : map \times string \times string \rightarrow map$
- $find : map \times string \rightarrow string$
  using special string `"not_found"` if given key is not in map.
- . . . and maybe more operations if desired

with properties:

$$find(emptyMap, k) = "not\_found"$$

$$find(insert(m, k, e), k) = e$$

$$find(insert(m, k, e), k') = find(m, k') \text{ if } k \neq k'$$

what happens if use insert multiple times with same keyword?

Live Programming: Implementing the Map

Follow on svn:

svn checkout svn://repos.gbar.dtu.dk/samo/cpp2016/
with username `student` and password `yvyebbnnq532ej3b`

- We start with a very simple implementation without OO
  - ★ We don't care about efficiency for now; just something correct!
- Dynamics: creating and deleting maps
- Abstraction: make it an ADT using classes in C++
- Split into `.cpp` and `.h` files
- Templates: allowing maps between (almost) arbitrary data types

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 to realize ADTs: the user of a class cannot directly manipulate variables, but only call functions. Aka data encapsulation
  - ★ We can change the implementation without changing the calling program.
- Constructor/Destructor called when an object is created/destroyed.
  - ★ Objects can be used as local variables of a function e.g.:
    `void main{ dict d; ... }`
    Created/destroyed when entering/leaving the function
  - ★ ... or created/destroyed with `new/delete`.