Organizing Programs and Data

Lecture 5
Hartmut Kaiser
hkaiser@cct.lsu.edu
http://www.cct.lsu.edu/~hkaiser/fall_2012/csc1254.html
Abstract

We will discuss functions as the main means of organizing computation. We rework the student grades example to be more modularized.
DRY - Don’t repeat yourself (DIE – Duplication is Evil)

The probably single most fundamental tenet in programming is to avoid repetition.

Many programming constructs exist solely for that purpose (e.g. loops, functions, classes, and more).

As soon as you start repeating yourself - create a new abstraction.

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system (program)

Tightly related to the Open/Closed Principle and the Liskov Substitution Principle

http://en.wikipedia.org/wiki/Don%27t_repeat_yourself
I wish I had been taught more about the importance of variable and method naming in school. About writing code that speaks to the reader. Code that is clear and concise. Code that solves the problem in an efficient way but also communicates what it does and why from the perspective of the user.

I wish I had been taught more about how to write components that could be changed without actually changing that specific code at all. I wish I had been taught more about polymorphism and composition. About how to structure applications not with a focus on reuse but on change. About why abstractions matter. I’ve written and seen code that is super DRY but that doesn’t contain a single abstraction and that group things in huge inheritance hierarchies. Code that is about as flexible and reusable as a sunken ship.

http://joelabrahamsson.com/entry/the-dry-obsession
Building a program

Analysis
  Refine our understanding of the problem
  Think of the final use of our program

Design
  Create an overall structure for the program

Implementation
  Write code
  Debug
  Test

Go through these stages repeatedly
Writing a program: Strategy

What is the problem to be solved?
  Is the problem statement clear?
  Is the problem manageable, given the time, skills, and tools available?

Try breaking it into manageable parts
  Do we know of any tools, libraries, etc. that might help?
    Yes, even this early: iostreams, vector, etc.

Build a small, limited version solving a key part of the problem
  To bring out problems in our understanding, ideas, or tools
  Possibly change the details of the problem statement to make it manageable

If that doesn’t work
  Throw away the first version and make another limited version
  Keep doing that until we find a version that we’re happy with

Build a full scale solution
  Ideally by using part of our initial version
Writing a Program

Even experienced programmers make mistakes
  Lots of mistakes; it’s a necessary part of learning
Designing a good program is genuinely difficult
It’s often faster to let the compiler detect gross mistakes than to try to get every detail right the first time
  Concentrate on the important design choices
Building a simple, incomplete version allows us to experiment and get feedback
  Good programs are “grown”
Organizing Programs and Data

We have been using standard data structures and algorithms, where each:

- Solves a particular problem
- Is independent from the others
- Has a name

Our program (computing student grades) has only the first of these qualities

- For larger programs the goal is to achieve all three

  *Code gets unmanageable*
Organizing Programs and Data

Two facilities to break program into smaller parts
- Functions
- Data structures

Today we will look at organizing computations: i.e. functions
- Functions are a way to encapsulate a piece of work
  - Reuse: avoid redoing explicit computation
  - Naming: allow to think in more abstract terms
  - Encapsulation: hide implementation details
Organizing Computations

For example, compute overall grade from pre-computed homework, midterm and final grades

```java
// compute a student's overall grade from midterm and final exam
// grades and homework grade
double grade(double midterm, double final, double homework)
{
    return 0.2 * midterm + 0.4 * final + 0.4 * homework;
}
```

Function consists of
Return type: **double**
Name: **grade**
(Optional) list of arguments: **double midterm, final, homework**
Function body: { ... }
Alternative Function Syntax

- C++11 allow to write functions as (requires g++ 4.4, vs2010):
  
  ```
  auto grade(double midterm, double final, double homework) -> double
  {
  ... 
  }
  
  In this case no real value, but we will see sensible use cases later
  
  Exception: lambda functions, have no name, they use [] instead (requires g++ 4.4, vs2010):
  
  ```
  std::foreach(words.begin(), words.end(),
  [](std::string s) {
    std::cout << s << "\n";
  }
  );
  ```
Organizing Computations

Functions have to be *called* (invoked) to be useful.

The output statement we had earlier:

```
cout << name << ", your final grade is: " << setprecision(3)  
  << 0.2 * midterm + 0.4 * final + 0.4 * sum / count  
  << setprecision(prec) << endl;
```

Becomes now:

```
cout << name << ", your final grade is: " << setprecision(3)  
  << grade(midterm, final, sum / count)  
  << setprecision(prec) << endl;
```
Calling Functions

Must match: name, supplied argument count, argument (type) sequence

Each argument initializes a newly created instance of the corresponding parameter

The parameters behave like ordinary local variables inside the function

  Call by value: the parameters are initialized from a copy of the argument
Calling Functions: Control Flow

```c
int square(int x) {
    return x * x;
}

int main() {
    i = 0;
    while (i < 100) {
        square(i);
        ++i;
    }
}
```
Finding Medians

Encapsulate finding a median

```cpp
// compute the median of a vector<double>
// note: calling this function copies the whole vector
double median(vector<double> vec)
{
    auto size = vec.size(); // auto size = ...
    if (size == 0)
        throw domain_error("vector is empty, median undefined");

    sort(vec.begin(), vec.end());
    auto mid = size / 2; // auto size = ...
    return size % 2 == 0 ? (vec[mid] + vec[mid-1]) / 2 : vec[mid];
}
```
Reimplementing Grading Policy

Grading policy:

// Compute a student's overall grade from midterm and
// final exam grades and all homework grades.
// This function does not copy the vector argument
// (as median does so for us).
double grade(double midterm, double final, vector<double> const & hw)
{
    if (hw.size() == 0)
        throw domain_error("student has done no homework");
    return grade(midterm, final, median(hw));
}
Reimplementing Grading Policy

References:

```cpp
vector<double> const& hw;

Const reference to a vector of double
Defines a new name for an object (here argument)

vector<double> homework;
vector<double>& hw = homework;

// 'hw' is a synonym for 'homework'

Anything we do with hw is equivalent to doing the same thing to homework (and v.v.)

Adding the const creates a read-only synonym
```
Reimplementing Grading Policy

References

Reference to a reference is still a reference to the original object

```cpp
vector<double>& hw1 = hw;
vector<double> const& chw = hw1;
```

Constness is preserved:

```cpp
vector<double>& hw1 = chw;    // error!
```

Reference as parameter asks to get access to the original object (the argument)

No copying!
Reimplementing Grading Policy

Function named `grade()`
- Same name as existing function: overloading
- No ambiguity as their parameters have different types

Error checking
- Checked same error as in `median()` function

Better error message
Reading Homework Grades

Reading an arbitrary number of doubles into a vector

```cpp
// read homework grades from an input stream into a vector
istream& read_hw(istream& in, vector<double>& hw)
{
    // to be filled...
    return in;
}
```

Returns 2 things:

- Whether input operation was successful
- The data itself
Non-const references tell the intend to change the functions argument:

```cpp
vector<double> homework;
read_hw(cin, homework);
```

The function changes the data and the state of `cin`

Returns another reference, the same object which was passed:

```cpp
// now this:
if (read_hw(cin, homework)) { /* ... */ }
```

```cpp
// is equivalent to:
read_hw(cin, homework);
if (cin) { /* ... */ }
```
Reading Homework Grades

First attempt:

```cpp
istream& read_hw(istream& in, vector<double>& hw) {
    double x;
    while (in >> x)
        hw.push_back(x);
    return in;
}
```

Not quite right? Why?

What is if `hw` already holds elements?
What if `cin` is in an error state?
What about this solution:

```cpp
// read homework grades from an input stream
// into a vector<double>
istream& read_hw(istream& in, vector<double>& hw)
{
  if (in) {
    hw.clear(); // get rid of previous contents

    // read homework grades
    double x;
    while (in >> x)
      hw.push_back(x);

    // clear the stream so that input will work for
    // the next student
    in.clear();
  }
  return in;
}
```
Reading Homework Grades

`istream::clear()` vs. `vector<>::clear()`

Different semantics!

- `istream::clear()` resets error state
- `vector<>::clear()` deletes all entries
Calling Conventions

Fundamentally different treatment of `vector<double>`:

- `median()`
  - `vector<double>`: copies all data
- `grade()`
  - `vector<double> const&`: no copy, read-only access
- `read_hw()`
  - `vector<double>&`: no copy, read-write access
Calling Conventions

Call by value

Default behavior
Copies the data passed as an argument
Proper thing to do for
  *Small data types (int, double, etc.)*
  *If a copy is required anyways*

Any modification to the parameters will not be reflected for the arguments
Calling Conventions

Call by const reference

Parameter needs explicit annotation (const&)

Does not copy the data itself, but creates a read-only reference to the original data

Proper thing to do for

Larger data types (efficiency)

Function does not change the value of parameters
Calling Conventions

Call by (non-const) reference
Parameter needs explicit annotation (&)
Does not copy the data itself, but creates a read-write reference to the original data
Proper thing to do for
- Any data types (not only for efficiency)
- Function does change the value of parameters
  Way to ‘return’ several things from a functions

Any modification to the parameters will be reflected for the arguments
Arguments need to be ‘lvalues’, so this will not work:

```
// function returning the vector of homework grades
vector<double> get_homework() {...}

// this does not work, as return values are not 'lvalues'
read_hw(cin, get_homework());
```