LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Eric Elfving 2017-08-16

Computer examination in

TDDD38 Advanced Programming in C++

Date 2017-08-16

Time 14-19

Department IDA

Course code TDDD38

Exam code DAT1

Examiner

Klas Arvidsson (klas.arvidsson@liu.se)

Administrator

Anna Grabska Eklund, 28 2362

Teacher on call

Eric Elfving (eric.elfving@liu.se, 013-28 2419)
Will primarily answer exam questions using the student client.
Will only visit the exam rooms for system-related problems.

Allowed Aids (tillåtna hjälpmedel)

An English-* dictionary may be brought to the exam.
No other printed or electronic material are allowed.
The cppreference.com reference is available in the exam system.

Grading

The exam has a total of 25 points.
0-10 for grade U/FX
11-14 for grade 3/C
15-18 for grade 4/B
19-25 for grade 5/A.

Special instructions

- All communication with staff during the exam can be done in both English and Swedish.
- Don't log out at any time during the exam, only when you have finished.
- Given files are found in subdirectory `given_files` (write protected). The exam will be available as a pdf in this directory at the start of the exam.
- Files for examination must be submitted via the Student Client, see separate instructions (`given_files/student_client.pdf`)!
- When using standard library components, such as algorithms and containers, try to chose "best fit" regarding the problem to solve. Avoid unrelated/unnecessary computations and unnecessary data structures.
- C style coding may cause point reduction where C++ alternatives are available.
- Your code should compile. Commented out regions of non-compiling code may still give some points. Resource leaks and undefined behavior is important to fix.

1. **Theory questions** Answers may be given in either Swedish or English. Write your answers to all theory questions in a single textfile and submit it as `ASSIGNMENT #1`.

   (a) Give a reason for why the language needs the `virtual` keyword from a viewpoint of the zero overhead principle? [1p]

   (b) The `<functional>` header has a lot of function objects that implement simple operations such as `std::less` implementing `operator<`. Why do we need these function objects? [1p]

   (c) Exceptions should be caught by reference. Give two reasons why. [1p]

   (d) Suppose that we have a class `C` with an int member `i` with a constructor that initializes `i` with an int value (see below). A problem with this implementation is that we are also able to call this constructor with values that are convertible to int. Give the declarations needed in the class to give compilation errors when the constructor is called with values of type that is convertible to int (i.e. double, long, bool, char etc.). [1p]

```
class C
{
    int i;
public:
    C( int v ) : i{v} {}
};
```

   (e) What is the difference between the `noexcept` operator and the `noexcept` specifier? [1p]

2. Copy the file `program2.cc` from the `given_files` directory and add your code according to specifications below and comments in the given file. No modifications are allowed in the given file other than explicitly specified. [5p]

   Walkways and pavements are often paved with some type of slab. The task in this assignment is to create a polymorphic class hierarchy to represent slabs of different material.

   Create an abstract base class `Slab` with three direct subclasses `Concrete`, `Rock` and `Brick`.

   - `Slab` stores color (a string such as "Gray"), weight (double) and size (a string such as "25x12x4"). It has the following public member functions:
     - `get_color()` returns the color
     - `get_weight()` returns the weight
     - `get_size()` returns the size
     - `clone()` returns a pointer to a copy of the current object
   - Concrete slabs can be tumbled (makes it look older). Objects of type `Concrete` stores whether it is tumbled or not (bool) and has a member function `tumbled()` that returns this value. By default, all `Concrete` objects are NOT tumbled.
   - `Rock` stores which kind of rock it is (string such as "Granite" or "Marble"). A member function `get_type()` returns this string. The color can not be set and should always be "Nature".
   - `Brick` has no specific members other than those provided by `Slab`.

   All values are to be set at construction and will never change. It should not be possible to modify the information after construction.
   Since this is a polymorphic class hierarchy, objects should be handled with pointers and it mustn't be possible to copy an object by any other means than the `clone` function.
   The member functions specified above is the only allowed interface. You are not allowed to add other public members.

3. The file `given_files/program3.cc` contains an implementation of selection sort for sorting a `vector<int>`. Copy that file and make the function more like the algorithms in STL by making the following modifications: [5p]

   - Generalize the parameter list so that it takes two iterators spanning the range of values to be sorted and one comparison callable that allows the user to modify how to compare elements.
   - Modify the implementation as much as needed, you may change as much as you like as long as it still implements selection sort.
   - For full points, the minimum requirement on the iterator type should be ForwardIterator.
   - The comparison should be defaulted as `std::less`.
   - Create a main function that tests your implementation with at least two different containers (with different content). For full marks the following example compiles:

     ```
     forward_list<string> lst {"hello", "this", "is", "a", "test"};
     ssort(begin(lst), end(lst),
           [](string a, string b){
              return a.length() < b.length();
           });
     ```

4. In this exercise, your task is to create a function `logic_combine` which takes three arguments (two iterators and a policy) and converts each element in the given range to bool and combines the values. The conversion and the type of combination is decided by the given policy. The algorithm should work almost like `std::accumulate`; given a start value and a range of values, apply a function that reduces the range of values into a scalar value. [5p]

   You are also to create three different policies to modify how the algorithm works. All policies have the following functions and members:

   - A starting value (of type bool)
   - `prefix` takes an element from the given range and returns a boolean value (casts to bool)
   - `combine` takes two boolean values and combines them with some logical operation
   - `done` takes the current combined value and returns true if it's unnecessary to continue checking more elements.
   - An ending value - the value returned if done returns true.

   A pseudo code implementation of `logic_combine` would look something like this:

```
Given policy P
result := P.startValue
for each element e:
    val := P.policy(e)
    result := P.combine(result, val)
    if P.done(result):
        return P.endValue
return result
```

   The following policies are to be implemented:

   `All` starts with value `true`, does nothing specific in prefix (just casts) and combines values with `&&`. Returns false when the result of a combination is false.

   `Any` starts with false, does nothing in prefix, combines with `||` and returns true if the result of combination is true at any time.

   `AtLeast` is a bit different. An object of type AtLeast is constructed with an int `N` that is used as a counter. When `N` `true` values have been found, `done` returns true and in that case the result value is true.

   `logic_combine` should by default use policy `All`.

   See `given_files/program4.cc` for examples of usage and expected results. Please note that your function should work with any container that stores elements that are convertible to `bool`.

5. Write all your code in a file named `program5.cc`. [5p]

   In the `given_files` directory, there is a file named `data1.txt` containing values for this assignment.

   Please note that it is important to use the standard library as much as possible in this assignment. Usage of hand-written repetition statements are penalized. Also make sure to use the best fitting algorithm, `for_each` is not a valid solution to all problems. Use lambdas or function objects from the STL to modify the algorithms, not normal functions.

   Read all data from STDIN and print to STDOUT.

   1. Read the data as integers `int` from cin and store the values in a suitable sequential container. This container is the only storage container allowed in this assignment.
   2. Print number of values read and all the read values to cout, separated by spaces.

      ```
      147 values read.
      5 61 2 62 63 64 65 ... 7 6 5 4 3 2 1
      ```

   3. Sort the values in decending order.
   4. Remove all duplicates.
   5. Print the remaining values.

      ```
      Unique values in decending order:
      111 110 109 108 107 106 105 ... 7 6 5 4 3 2 1
      ```

   6. Calculate the number of values representing 5% of the total (rounded down). With 111 values 5% represents 5.55 numbers, which should be rounded down to 5.
   7. Remove the bottom 5% and the top 5% of all values. For the given 111 unique values, 10 will be removed. Print the remaining values.

      ```
      Smallest and largest values removed.
      106 105 104 103 102 ... 10 9 8 7 6
      ```

   8. Calculate the mean value of the remaining elements.

      ```
      Mean value: 56.0
      ```

   9. For each element calculate the absolute value of the difference between the element and the mean value. Then print the sum of all the differences. If the container is called X, the following is to be calculated:

      $$\sum_{\forall x_i \in X} |x_i - \bar{X}|$$

      ```
      Sum of differences: 2550.0
      ```