LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Eric Elfving 2018-04-05

Computer examination in

TDDD38 Advanced Programming in C++

Date 2018-04-05

Time 08-13

Department IDA

Course code TDDD38

Exam code DAT1

Examiner

Eric Elfving (eric.elfving@liu.se)

Administrator

Anna Grabska Eklund, 28 2362

Teacher on call

Eric Elfving (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.
- 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.

Theory questions

1. Answers may be given in either Swedish or English. Write all your answers in one text file and submit as assignment 1.

   (a) Why is it usually wrong to implement `operator<<` (formatted stream output) as a member function? [1p]

   (b) There are several operators that only can be written as non-static member functions, name two. [1p]

   (c) What is an inline namespace? [1p]

   (d) What is wrong in the following example: [1p]

```
string & fun()
{
    string str {"hello"};
    return str;
}

int main()
{
    cout << fun().length();
}
```

   (e) What is bad with this code: [1p]

```
class Base {
    Base()
    {
        // ...
        fun();
    }
    virtual void fun() = 0;
};
```

Practical questions

2. Until reflections are added to the language, we have quite crude ways of querying the compiler about the types involved in expressions. In this assignment, you are about to create some tools to help with type information. When you are done, we'll be able to get type name and size for two common types and have an easy time extending for more types. [5p]

   Getting the type name is rather simple. In the given code in file `given_files/program2.cc`, there is a primary template called `Type_Name`. Copy this file to your working directory and add two specializations to `Type_Name`, one for `int` that has a `name` function that returns the string "int" and a specialization for `std::string` that has a `name` function that returns "string".

   For size, the simplest solution is just to call the `sizeof`-operator, but we want more information that that. For any type that has a `size` member function, we want that function to be called at runtime to find the number of elements of the actual object and multiply that with the size of the value type using `sizeof` (assume that there is a `value_type` type member). If the type doesn't have a size member, just call sizeof.

   Create a template class `Meta_Type` that has one static member function `name` that calls `Type_Name::name` and a member function `size` that returns the size (number of elements or size of the type) according to the previous definition.

   Overload the formatted output operator (`operator<<`) for `Meta_Type` to make the given main function compile and work according to the comments.

3. When printing data to some stream, it is common to want to wrap the text at some specified line length. In this assignment, you are going to write an *OutputIterator* that simulates this behavior. [5p]

   Create a class `LineWrapper` having the five common member types required by an iterator (`value_type, iterator_category, reference, pointer` and `difference_type`), all can be an alias for `void` except `iterator_category` which should be `std::output_iterator_tag`.

   Internally, an object of type `LineWrapper` keeps track of an output stream, a maximum length and the length of the current line. The stream and the max length can be set in the constructor (should be defaulted as `std::cout` and `80`) while the current length always start at 0.

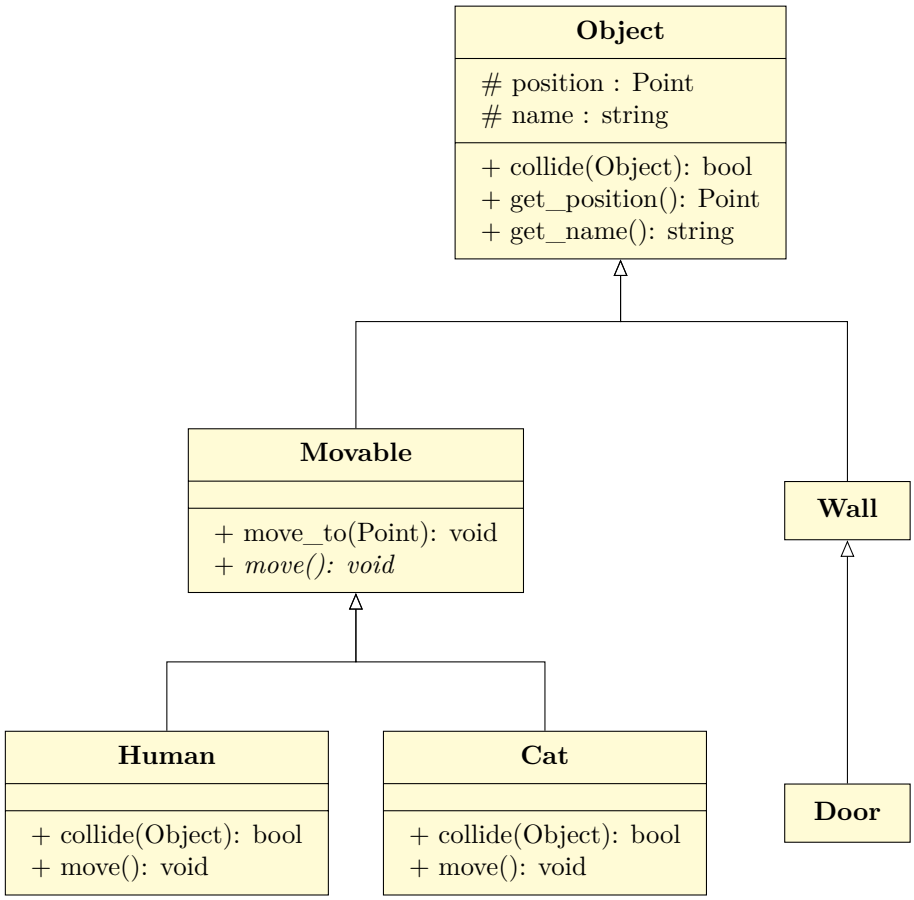
   Overload the operators required on OutputIterator according to `http://en.cppreference.com/w/cpp/concept/OutputIterator`. For hints on implementation, study the usage in the implementation of copy at `http://en.cppreference.com/w/cpp/algorithm/copy`.

   When you are done, the given code at `given_files/program3.cc` should work without any modifications together with your iterator.

4. Your team is about to create a game and you are responsible of creating a part of the class hierarchy that describes objects that you want to draw to the screen. At the moment, your game can draw people (class `Human`), cats, walls and doors. Create a base class `Object` that stores and lets the user access the object's position and name. It should also have a member function collide that checks if this object collides with another object. By default, this is done by checking if the two objects has the same position, but this can be overridden by subclasses. [5p]

   There is a direct subclass to `Object` called `Movable` to represent objects that can move. `Movable` has a member function `move_to` that sets the position and a member function `move` that all subclasses have to implement (all have different behavior). Create two subclasses to `Movable`; `Cat` and `Human`. In this simple version, cats move horizontally (increase x in move) and will collide with doors and walls that are at the same location. Human objects move vertically (only increase y in move) and will only collide with Wall objects if they are in the same location (can open Doors).

   It should not be possible to copy any objects in the class hierarchy.



   There are also some implementation to be done as a simple test program, see comments in the given code at `given_files/program4.cc`.

5. The BBP Formula (named after Bailey, Borwein and Plouffe), seen below, can be used to calculate π. [5p]

$$\pi = \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left(\frac{1}{16} \right)^n$$

The file `given_files/program5.cc` contains a program that let the user for a number of terms and gives an approximation of π for that many terms. Copy the given file and refactor the code so that it uses the STL as much as possible. The main goal is to make the code more readable by selecting algorithms that express your intent in each calculation. A full solution shouldn't require any standard iteration statements.