LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Eric Elfving 2016-06-02

Computer examination in

TDDD38 Advanced Programming in C++

Date 2016-06-02

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 primarialy 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.

Theory questions

Answers may be given in either Swedish or English. Submit your answers to all theory questions in one text file called `THEORY.TXT` and submit it as `Assignment #1`.

1. Give one code example of where the compiler uses ADL for name lookup. [1p]
2. Give, with either text or a short (1-5 simple statements each) code example, two examples of undefined behavior. [1p]
3. Why should you prefer usage of the free function `std::begin` instead of using the member function? [1p]
4. What is the difference between implementation defined behavior and undefined behavior? [1p]
5. What is meant by the zero-overhead rule? [1p]

Programming exercises

6. Copy the file `assignment6.cc` from `given_files` and make additions to your copy. Submit your solution as `Assignment #6`. [5p]

   There is a built-in problem with some of the integral types, `int` in particular, in that we get undefined beavior when going outside the range of the type. Your task in this assignment is to create a type `Modular` that is templated on the internal storage type and the value range (given as two template non-type parameters). The value range shall be defaulted as the default value range for the given type given by `numeric_limits` from the `<limits>` header. Your class shall support the following operations:

   - Explicit constructor for the underlying type. If the value given is outside the current value range, the value shall be set to the lowest value.
   - Assignment from underlying type. If the value is outside the value range, the assignment shall not modify the object but throw an `std::domain_error` exception.
   - Implicit conversion to the underlying type.
   - Incrementing operators (both postfix and prefix versions) to step the value by one.
   - Addition with another `Modular` object, possibly having another value range. The value range of the result shall be the same as the left hand side object.

   The given file has a main program that tests these operations.

7. Copy the file `assignment7.cc` from `given_files` and make additions to your copy. Submit your solution as `Assignment #7`. [5p]

   In this exercise, it's extra important for you to use the standard library in a good way. To get full marks, use the correct algorithm for the given task – `for_each` is not a valid solution for all tasks and you won't need any hand-rolled loops.

   The given file has a simple function to test if a value is a prime number and a simple type alias called `num_type`.

   Your task is to create a program that generates a random number with its prime factors by following the algorithm below.

   1. Create a `vector<num_type>` with space for 10 elements
   2. Fill the vector with random values in the range $[2, 75]$
   3. Remove (with the help of the given `is_prime` function) the numbers that aren't prime numbers
   4. Sort the remaining values
   5. Print the remaining numbers to standard output
   6. Calculate the product of the remaining numbers
   7. Print the product to standard output

8. Copy `assignment8.cc` from `given_files` to your working directory. Add your own code according to instructions below and submit as `Assignment #8`. [5p]

   The following function for sorting `int` values in range `[begin, end)` is given:

```
void sort(int* begin, int* end)
{
    for ( ; begin +1 != end; ++begin)
    {
        int* min = begin;
        for (int* pos = begin + 1; pos != end; ++pos)
            if (*pos < *min)
                min = pos;
        std::iter_swap(begin, min);
    }
}
```

   It could be used as follows for sorting the elements of an array:

```
int num[]{ 7, 5, 10, 432, 1, 2, 3, 4, 5 };
sort(begin(num), end(num));
```

   Generalize sort in the following ways:

   - any type fulfilling the requirements shall be possible to use
   - the ordering operation, now `<`, shall be replaced by an ordering policy
   - make it work also for real iterators, not only for plain pointers (such as `int*`)

   Do this by encapsulating sort in a template class named `Sort`. Sort shall have two template parameters, one for the type of values to sort, and the other for the ordering policy, see below. Leave the problem of making it work for iterators until later!

   Define two sort policy classes, `Ascending` (normal sort order) and `Descending`:

   - With policy Ascending elements shall be sorted in ascending order, i.e. as the given code do. This shall be the default policy for Sort.
   - With policy Descending elements shall be sorted in descending order.

   After this, `Sort` should be possible to use on arrays in the following ways (Note: these examples gives an important hint about implementation):

```
Sort<int, Ascending>::sort(begin(num), end(num));
Sort<int, Descending>::sort(begin(num), end(num));
Sort<int>::sort(begin(num), end(num));   // default used
```

   Now, modify Sort so iterators, e.g. container iterators, can be used:

```
vector<int> v(begin(num), end(num));
...
Sort<int, Descending>::sort(begin(v), end(v));
```

   Uncomment the corresponding test code in `main()` for the final test.

9. Write your code in a file named `assignment9.cc` and submit your solution as `Assignment #9`. [5p]

A company creating saunas wants your help to create a polymorphic class hierarchy to handle their three basic types of sauna heaters; Steam room heaters, wood-burning heaters and the standard electrical heaters.
Create one abstract base class `Sauna` with three direct subclasses; `Steam_Heater`, `Wood_Heater` and `Electrical_Heater`. `Sauna` has the following properties:

- temperature, should be accessible and modifiable with public member functions.
- default temperature, the thermostat temperature setting. If unset, it shall be room temperature (22 degrees). Is set at construction and never changed.
- whether the heater is on or not. A newly created heater is always off. When a heater is turned on, the temperature is set to the default temperature. When the heater is turned off the temperature is reset to room temperature.

`Sauna` has a public function `get_clone` that calls a non-public pure virtual function `clone` that returns a new object of the dynamic type of the current object which `get_clone` then returns. This is the only way of publicly copying objects in the hierarchy. Also create a virtual function `str` that returns a string that names the current object.

- `Steam_Heater` has a default temperature of 38 degrees and has a `str` function returning the string "Steam sauna".
- `Wood_Heater` hasn't got a default temperature and has a `str` function returning "Wood-burning sauna". Since wood-burning heaters don't have a thermostat, an extra member function `add_log` is added that increases the current temperature with 5 degrees. Physically there would be a maximum temperature but you can ignore this.
- `Electrical_Heater` has a default setting of 85 degrees and the `str` function return "Electrical heater".

Overload the formatted output operator for `Sauna` to print the following information:

- the type of the sauna
- if the sauna is on:
  - The string `[ON]` followed by the temperature
- else: the string `[OFF]`

An example of above rules with a wood-burning sauna heater at 76 degrees:
`Wood-burning sauna [ON] 76 degrees`

Create a program with a vector referring to one object of each concrete type. Pass a copy of each separate object to another function that does the following:

1. print the sauna to `cout`
2. turn it on
3. if it is a wood-burning heater, add 10 logs
4. print it again

Handle resources in a good way, your program should not have any memory leaks.