LINKÖPING UNIVERSITY
Department of Computer and Information Science
Software and Systems
Eric Elfving 2017-04-20

---

Computer examination in

# **TDDD38** Advanced Programming in C++

---

**Date** 2017-04-20

**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.

## Theory questions

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

1. Given the following snippet of code, give a reason for having the *using declaration* on line 4 instead of qualifying the call on line 5 (i.e. `return std::begin(range)`). [1p]

```
1 template <typename Range>
2 auto iter(Range && range)
3 {
4     using std::begin;
5     return begin(range);
6 }
```

2. Given the following primary template declaration, give a declaration of a specialization of `foo` for type `char`. [1p]

```
template <typename T>
struct foo;
```

3. Assuming `v` is a `std::vector<int>`, why is the following code wrong? [1p]

```
static_assert(v.size() > 0, "vector is empty!");
```

4. Assume you have definitions for the following functions: [1p]

```
void fun(int, short);
void fun(int, double);
```

   The call `fun(2,3)` will give an ambiguous overload. Why?

5. Describe what `length` below does and give a valid example of a call to length. [1p]

```
template <typename ...Args>
int length(Args && ... args)
{
    return sizeof...(args);
}
```

## Practical questions

6. The file `given_files/program6.cc` contains some given code. Copy the file and add implementation as stated below and submit it as `ASSIGNMENT #6`. [5p]

   The main task in this assignment is to read a number of images (or at least data on images) from the user and print an estimated total size of the given images. All images have a size (width and height) but different image formats represent the image data in different ways and will thus require different storage space.

   The given program reads descriptions for different images and tries to create objects to represent the given input. Your task is to create classes to represent different image formats. You are required to create an abstract base class `Image` with three derived classes `PNG`, `JPG` and `BMP`. `PNG` and `BMP` will only require the width and the height, `JPG` also need a level of quality (given as an integral value between 1 and 100).

   All concrete classes must have a member function `parse` that takes an `Image_Data` (given class). If the data represent an image of the current class, `parse` will create a new object dynamically and return a pointer to it, otherwise it will return `nullptr`. All classes must also implement a function `size` which return the size according to the table below (`int`):

   **BMP** `width * height * 3`

   **PNG** `width * height * 1.5`

   **JPG** `width * height * 3 * (quality/100.0)`

   Add code to the end of main to calculate and print the total size of all images entered by the user according to the example below.

```
Enter one line for each image on the format "type [type specific data]". Exit with "q".

JPG 5000 3000 50
BMP 100 100
TIFF 300 355
!!! TIFF is an invalid file format !!!
PNG 1000 1000
JPG 100 200 100

Total size: 9050000 bytes!
```

7. Write your code on a file named `program7.cc` and submit it as `ASSIGNMENT #7`. [5p]

   In this exercise, you will implement a simple compression algorithm using the standard library components. It is extra important for you to show your knowledge and skills in the standard library and tools available, usage of normal iteration statements will give deductions.

   Your task is to read an unknown number of words and replace the 10 most common words with shorter symbols, thus reducing the number of characters in the text. Your program will also print a list of words that have been replaced so that it's possible to go back to the original text.

   Algorithm:

   1. Read all words from STDIN to a `vector<string>` (called `words` below).
   2. Create a frequency table of the words.
   3. Sort the frequency table in descending order of frequency.
   4. Copy the ten most common words to a new `vector<string>`, `replace`.
   5. Replace the words in `words` that also exist in `replace`. A word that exists in `replace` is replaced by the symbol `$N`, given that `N` is the index of the word in `replace`. The 10 words will then be replaced with the symbols `$0` to `$9`.
   6. Print a translation table with the format `$0=word1;$1=word2;...;$9=word10;` on one line on cout.
   7. Print all words in `words` to cout separated with one space. This will remove all other kind of whitespace that happened to exist in the given output (expected behavior).

   Steps 2 and 3 might require new data structures. That is okay, just make sure not to copy more data then necessary.

   The directory `given_files` contains a program `uncompress` that takes the output from your program and can be used to test your program. The following command should give the contents of the file `textfile.txt` as output (if you disregard difference in whitespace) if everything works.

   ```
   cat textfile.txt | a.out | given_files/uncompress
   ```

8. Copy `program8.cc` from directory `given_files` to your working directory, modify it according to descriptions below and submit as `ASSIGNMENT #8`. [5p]

   A class `Number` is given. An object of type `Number` stores an integer value, which can be initialized or set by a double or a double compatible value, which will then be truncated (automatic conversion from `double` to `int`).

   Modify `Number` to be a class template, with three template parameters:

   - one for the type of the stored value, so this type can be chosen, e.g. as short, int, long int, long long int, or any other suitable type.
   - one for the parameter type of the constructor and `set_value()`, so this type can be chosen, e.g. as float, double, long double, or any other suitable type with rounding possible.
   - one for a rounding policy, so it can be chosen how values given to the constructor or to `set_value()` are to be rounded when the stored value is initialized or set, e.g. round down, round up, round towards zero, etc.

   Define three policy classes, named `Round_Down`, `Round_Up`, and `Round_Towards_Zero`, each with a member function `round()`, taking the value to be rounded as argument:

   - `Round_Down` round down (towards negative infinity), 9.5 is rounded to 9, and -9.5 is rounded to -10. This is the same as the `floor` function.
   - `Round_Up` round up (towards positive infinity), 9.5 is rounded to 10, and -9.5 is rounded to -9. This is the same as the `ceil` function.
   - `Round_Towards_Zero` round towards zero, 9.5 is rounded to 9, and -9.5 is rounded to -9.

   `Round_Towards_Zero` is the default rounding policy for `Number`.

   Modify `Number` to use the rounding policy when required and possibly in other respects when made a template.

   Modify the given test code so that the program gives the following output:

```
Value      Down       Up         Towards 0

+9.67      +9         +10        +9
+9.50      +9         +10        +9
+9.35      +9         +10        +9
+9.00      +9         +9         +9
+0.00      +0         +0         +0
-9.00      -9         -9         -9
-9.25      -10        -9         -9
-9.50      -10        -9         -9
-9.67      -10        -9         -9
```

   Hint: the `floor()` and `ceil()` standard library functions, come with `<cmath>` and can be handy when implementing the rounding policies.

9. Copy file `program9.cc` from the directory `given_files` to your working directory. Add your own code to that file and submit as `ASSIGNMENT #9`. The given file contains a test program. [5p]

   Construct a container class named `Ordered_Vector`, where the elements are stored in order according to some comparison.

   `Ordered_Vector` shall be a class template with two template parameters:

   - one for the element type
   - one for the comparison to be used for ordering the elements – `std::less` shall be default

   `std::vector` shall be a base class of `Ordered_Vector`. In this way most of the properties of `Ordered_Vector` will be obtained from `std::vector`.

   `Ordered_Vector` shall have the following properties:

   - default constructor
   - copy and move constructor
   - copy and move assignment operator
   - destructor
   - `size()` – same as `vector<T>::size`
   - `empty()` – same as `vector<T>::empty`
   - `clear()` – same as `vector<T>::clear`
   - `operator[]` – same as `vector<T>::operator[]`, but only the `const` version.
   - `const_iterator` – same as `vector<T>::const_iterator`
   - `begin()` – same as `vector<T>::begin`, but only the version that gives `const_iterator`.
   - `end()` – same as `vector<T>::end`, but only the version that gives `const_iterator`.
   - `cbegin()` - same as `vector<T>::cbegin`
   - `cend()` - same as `vector<T>::cend`
   - `insert(x)` – a member function which takes a value of the element type in question and inserts the value in order into the `Ordered_Vector` object.

   Note, using sorting, e.g. algorithm `std::sort`, to order the elements is not allowed! Elements must be inserted in order!

   No other `std::vector` members than those mentioned above are allowed be part of the public interface of `Ordered_Vector`.

   Hint: The using declaration (class scope context) can be very helpful...