Examination
EDAF50 C++ Programming

2025-06-03, 08:00–13:00

Aid at the exam: one C++ book. *Not allowed:* printed copies of the lecture slides or other papers.

Assessment (preliminary): the questions give $10 + 5 + 9 + 10 + 6 + 10 = 50$ points. You need 25 points for a passing grade (3/25, 4/33, 5/42).
You must show that you know C++ and that you can use the C++ standard library. "C solutions" don't give any points, and idiomatic C++ solutions that reimplement standard library facilities may give deductions, even if they are correct.
In solutions, resource management must also be considered when relevant – your solutions must not leak memory.
Free-text anwers should be concise but complete, well motivated and written clearly, to the point, and in complete sentences. Answers may be given in swedish or english.

For all problems, you may choose to answer "I don't know", which will be worth 20% of the maximum score of that problem. If you opt to do so, the sentence "I don't know." or "Jag vet inte." must be clearly given as the only answer to that problem. (I.e., you will not get any credit for an answer "I don't know" to a subproblem.)

Please write on only one side of the paper and hand in your solutions with the papers numbered, sorted and facing the same way. Please make sure to write your anonymous code and personal identifier on each page, and that the papers are not folded or creased, as the solutions may be scanned for the marking.

1. The standard algorithm `unique` has the following description:

```
template <typename ForwardIterator>
    // Every time a consecutive group of duplicate elements appears in
    // the range [first, last), the algorithm removes all but the first
    // element. That is, unique returns an iterator new_last such that
    // the range [first, new_last) contains no two consecutive elements
    // that are duplicates. The time complexity is linear.
ForwardIterator unique(ForwardIterator first, ForwardIterator last);
```

Example: the sequence `{1, 2, 2, 4, 5, 5, 5, 5, 6}` is changed to `{1, 2, 4, 5, 6}`. Please note that the algorithm does not require the sequence to be sorted, and only considers adjacent elements. For instance `{1, 4, 1, 1, 4, 4, 4}` is changed to `{1, 4, 1, 4}`.

The effect of `std::unique` is to modify the sequence. We want to be able to iterate over the sequence of unique elements (as defined above) without copying or modifying the original. To that end, we want a function template

```
template <typename Iter>
    // Gives an iterator over the unique elements of the sequence.
    // The returned iterator references first.
    // operator++() causes the iterator to reference the next unique
    // element, or last.
    // The resulting object is comparable for equality to Iter.
unique_iter<Iter> unique_subseq(Iter first, Iter last);
```

Implement the function template `unique_subseq` and the class template `unique_itr` so that the following example works as intended.

```
template<typename It1, typename It2>
void print_elems(It1 first, It2 last)
{
    for(;first != last; ++first) cout << *first << " ";
    cout << "\n";
}

void example()
{
    std::vector<int> vi{1,1,1,1,2,2,3,4,5,5,5}; // input

    std::vector<int> expi{1,2,3,4,5};           // expected

    auto it = unique_subseq(begin(vi), end(vi));

    cout << "Actual result:   ";
    print_elems(it, end(vi));
    cout << "Expected result: ";
    print_elems(begin(expi), end(expi));
}
```

The expected output is

```
Actual result:   1 2 3 4 5
Expected result: 1 2 3 4 5
```

Hint: If you need to get the value type of an iterator `Iter` you can use either
`typename std::iterator_traits<Iter>::value_type` ,or `decltype(*std::declval<Iter>());`

NB! To simplify `unique_iter`, we use a simple manual loop printing the result instead of comparing the ranges with standard algorithms. Then we can compare the `unique_iter` to a `std::vector<int>::iterator`. Standard algorithms like `std::mismatch` requires both iterators in a range to have the same type.

2. Consider this code:

```
#include <iostream>

class Base
{
public:
  virtual void foo() const { std::cout << "Base's foo!" << std::endl; }
};

class Derived : public Base
{
public:
  void foo() { std::cout << "Derived's foo!" << std::endl; }
};

int main()
{
  Base* o1 = new Base();
  Base* o2 = new Derived();
  Derived* o3 = new Derived();

  o1->foo();
  o2->foo();
  o3->foo();
}
```

The code compiles without errors, but the output is:

```
Base's foo!
Base's foo!
Derived's foo!
```

where the programmer expected

```
Base's foo!
Derived's foo!
Derived's foo!
```

as `Base::foo()` is `virtual` and `o2` points to a `Derived` object.

Explain the behaviour. Why does not `o2->foo()` call `Derived::foo`? Why does `o3->foo()` call `Derived::foo`?

3. A class representing a person is defined in `user.h` as follows:

```
#ifndef USER_H
#define USER_H

#include <iostream>
#include <string>
#include <tuple>

class User {
  public:
    User() = default;
    User(const char* fname, const char* lname)
        : fn{new std::string(fname)}, ln{new std::string(lname)}
    {}
    ~User()
    {
        delete fn;
        delete ln;
    }

    friend std::ostream& operator<<(std::ostream& os, User u)
    {
        return os << *u.fn << " " << *u.ln;
    }

    bool operator==(User u) const { return *fn == *u.fn && *ln == *u.ln; }
    bool operator!=(User u) const { return !(*this == u); }
    bool operator<(User u) const
    {
        return std::tie(*ln, *fn) < std::tie(*u.ln, *u.fn);
    }

  private:
    std::string* fn{nullptr};
    std::string* ln{nullptr};
};
#endif
```

The following test program is written to test the comparison operators:

```
#include "user.h"

#include <iomanip>
#include <iostream>
#include <string>

using std::cout;

template <typename T, typename C>
bool test_cond(const std::string& msg, const T& t, const T& u, C c, bool exp)
{
    auto res = c(t, u) == exp;
    if (!res) {
        cout << "condition failed: [" << msg << "] for " << t << " and " << u;
        endl(cout);
    }
    return res;
}
```

continued on the next page...

```
int main()
{
    User u("Test", "Testsson");
    User v("Tim", "Plate");

    bool res{true};
    res &= test_cond("Equality", u, v, std::equal_to<User>(), false);
    res &= test_cond("Inquality", u, v, std::not_equal_to<User>(), true);
    res &= test_cond("Ordering (<)", u, v, std::less<User>(), false);
    res &= test_cond("Ordering (<)", v, u, std::less<User>(), true);

    if (res)
        cout << "Success.\n";
}
```

When the test program is run, the program crashes with the following output

```
user_test(36608,0x102b7c580) malloc: *** error for object 0x600000e75140: pointer being freed was not allocated
user_test(36608,0x102b7c580) malloc: *** set a breakpoint in malloc_error_break to debug
[1]    36608 abort       ./user_test
```

To investigate further, the programmer builds with `-fsantize=address`, and then the output is

```
=================================================================
==37443==ERROR: AddressSanitizer: heap-use-after-free on address 0x000108901fd7 at pc 0x0001047105dc bp 0
    ↪ x00016b6f2660 sp 0x00016b6f2658
READ of size 1 at 0x000108901fd7 thread T0
    #0 0x1047105d8 in std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>
        ↪ >::__is_long() const+0x64 (a.out:arm64+0x1000045d8)
    #1 0x1047112c0 in std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char>
        ↪ >::size() const+0x18 (a.out:arm64+0x1000052c0)
    #2 0x104711120 in bool std::__1::operator==<std::__1::allocator<char> >(std::__1::basic_string<char,
        ↪ std::__1::char_traits<char>, std::__1::allocator<char> > const&, std::__1::basic_string<char,
        ↪  std::__1::char_traits<char>, std::__1::allocator<char> > const&)+0x3c (a.out:arm64+0
        ↪ x100005120)
    #3 0x104711058 in User::operator==(User) const+0x7c (a.out:arm64+0x100005058)
    #4 0x10471175c in User::operator!=(User) const+0x118 (a.out:arm64+0x10000575c)
    #5 0x10471153c in std::__1::not_equal_to<User>::operator()(User const&, User const&) const+0x11c (a.
        ↪ out:arm64+0x10000553c)
    #6 0x10470e778 in bool test_cond<User, std::__1::not_equal_to<User> >(std::__1::basic_string<char, std
        ↪ ::__1::char_traits<char>, std::__1::allocator<char> > const&, User const&, User const&, std::
        ↪ __1::not_equal_to<User>, bool)+0x138 (a.out:arm64+0x100002778)
    #7 0x10470de3c in main+0x284 (a.out:arm64+0x100001e3c)
    #8 0x1048890f0 in start+0x204 (dyld:arm64e+0x50f0)

0x000108901fd7 is located 23 bytes inside of 24-byte region [0x000108901fc0,0x000108901fd8)
freed by thread T0 here:
    #0 0x104c7aacc in wrap__ZdlPv+0x74 (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4aacc)
    #1 0x10470eebc in User::~User()+0x60 (a.out:arm64+0x100002ebc)
    #2 0x10470ed48 in User::~User()+0x18 (a.out:arm64+0x100002d48)
    #3 0x104710d3c in std::__1::equal_to<User>::operator()(User const&, User const&) const+0x12c (a.out:
        ↪ arm64+0x100004d3c)
    #4 0x10470e3f0 in bool test_cond<User, std::__1::equal_to<User> >(std::__1::basic_string<char, std::
        ↪ __1::char_traits<char>, std::__1::allocator<char> > const&, User const&, User const&, std::
        ↪ __1::equal_to<User>, bool)+0x138 (a.out:arm64+0x1000023f0)
    #5 0x10470ddb8 in main+0x200 (a.out:arm64+0x100001db8)
    #6 0x1048890f0 in start+0x204 (dyld:arm64e+0x50f0)

previously allocated by thread T0 here:
    #0 0x104c7a6b4 in wrap_Znwm+0x74 (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4a6b4)
    #1 0x10470ed90 in User::User(char const*, char const*)+0x34 (a.out:arm64+0x100002d90)
    #2 0x10470e2a4 in User::User(char const*, char const*)+0x28 (a.out:arm64+0x1000022a4)
    #3 0x10470dd74 in main+0x1bc (a.out:arm64+0x100001d74)
    #4 0x1048890f0 in start+0x204 (dyld:arm64e+0x50f0)

SUMMARY: AddressSanitizer: heap-use-after-free (a.out:arm64+0x1000045d8) in std::__1::basic_string<char,
    ↪ std::__1::char_traits<char>, std::__1::allocator<char> >::__is_long() const+0x64
```

a) Explain what the problem is; where in the execution does the program crash, and what in the code causes the problem? Explain both the fatal error which directly causes the crash, and the underlying problem in the code that leads to this error.

b) Show how to change the class `User` so that the class works as expected, and this kind of error cannot happen.

4. Your task is to write a class `counter` that behaves just like an int, but also works as an accumulating function. The latter means that with a `counter` variable intialised to 100:

   ```
   counter c{100};
   ```

   calling it as a function

   ```
   c(10);
   ```

   increments its value by the argument, so in this case, `c` has the value 110 after the call. Calling it without an argument increments the counter by one.

   Implement the class ***`counter`****, so that the following example works as intended. Only implement what is needed by the example.*

   ```
   void example()
   {
       counter c;
       cout << "1: c = " << c << '\n';

       c = 17;
       cout << "2: c = " << c << '\n';

       int x = c;
       cout << "3: x = " << x << '\n';

       c();
       cout << "4: c = " << c << '\n';

       c(24);
       cout << "5: c = " << c << '\n';

       c = c + 100;
       cout << "6: c = " << c << '\n';

       if(c > 100){
           cout << "correct: " << c << " > 100\n";
       }
   }
   ```

   The expected output is

   ```
   1: c = 0
   2: c = 17
   3: x = 17
   4: c = 18
   5: c = 42
   6: c = 142
   correct: 142 > 100
   ```

5. The following program was written to learn how to concatenate a string and an integer, but it does not work as intended.

```
#include <iostream>
#include <string>

std::string concat(const char* str, int i)
{
    return std::string(str + i);
}

int main()
{
    std::string s1 = concat("testing", 1);
    std::string s2 = concat("test", 2);
    std::string s3 = concat("testing, testing, testing", 5);

    std::cout << "s1: " << s1 << '\n';
    std::cout << "s2: " << s2 << '\n';
    std::cout << "s3: " << s3 << '\n';
}
```

The program compiles without errors or warnings (with `-Wall -Wextra`), but when executed produces the following output.

```
s1: esting
s2: st
s3: ng, testing, testing
```

a) Explain the behaviour of the program.

b) Write another function concat2, that actually does the desired concatenation. That is, the program

```
int main()
{
    std::string s1 = concat2("testing", 1);
    std::string s2 = concat2("test", 2);
    std::string s3 = concat2("testing, testing, testing", 5);

    std::cout << "s1: " << s1 << '\n';
    std::cout << "s2: " << s2 << '\n';
    std::cout << "s3: " << s3 << '\n';
}
```

should produce the output

```
s1: testing1
s2: test2
s3: testing, testing, testing5
```

6. The `grep` utility searches its given input files and selects the lines that match one or more patterns. Here, the task is to implement a very limited version of grep, `filter_lines`, that searches an input file and outputs the lines containing a given phrase on `std::cout`.
The main program is as follows

```
int main(int argc, const char* argv[])
{
    if(argc != 3) {
        std::cerr << "Usage: filter_lines <phrase> <filename>\n";
        return 2;
    }
    std::string phrase = argv[1];
    std::string fname = argv[2];
    return print_matching_lines(phrase, fname);
}
```

Your task is to implement the function `print_matching_lines` so that it works with the above main program. The function shall

- open a file named `fname`,
- print each line in that file, that contains `phrase`, on `std::cout`, and
- return 0 (zero) on success, or print a message explaning what went wrong on `std::cerr` and return 1 on error.

Example usage: if the program `filter_lines` is run on the LaTeX source of this problem, with the phrase being `the function` (i.e., `./filter_lines "the function" filter_lines.tex`)
the output should be

```
Your task is to implement the function \code{print_matching_lines} so that it
phrase being \code{the function} (i.e.,
\code{./filter_lines "the function" filter_lines.tex})
Your task is to implement the function \code{print_matching_lines} so that it
phrase being \code{the function} (i.e.,
\code{./filter_lines "the function" filter_lines.tex})
\emph{Answer with an implementation of the function \code{print_matching_lines}.}
```

Answer with an implementation of the function `print_matching_lines`.