Tentamen
EDAF30 Programmering i C++

2020–01–15, 14:00–19:00

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

Assessment (preliminary): the questions give $9 + 8 + 18 + 15 = 50$ 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.
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.

Please write on only one side of the paper and hand in your solutions with the papers sorted and facing the same way, as the solutions may be scanned for the marking.

---

1. The following code fragment does not compile.

```
 1  class Foo{
 2  public:
 3      Foo(int i) {x = i;}
 4  private:
 5      int x;
 6  };
 7
 8  class Bar{
 9  public:
10      Bar(int i) {a = i;}
11  private:
12      Foo a;
13  };
```

   The compiler error is

```
foobar.cc: In constructor 'Bar::Bar(int)':
foobar.cc:10:16: error: no matching function for call to 'Foo::Foo()'
     Bar(int i) {a = i;}
                ^
foobar.cc:10:16: note: candidates are:
foobar.cc:3:5: note: Foo::Foo(int)
     Foo(int i) {x = i;}
     ^
foobar.cc:3:5: note:   candidate expects 1 argument, 0 provided
foobar.cc:1:7: note: Foo::Foo(const Foo&)
 class Foo{
       ^
foobar.cc:1:7: note:   candidate expects 1 argument, 0 provided
```

   a) Explain why the compiler gives this error. What is the root cause and why does that lead to this particular error?

   b) Change the code fragment to make it correct.

---

2. 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
```

3. Morse code (named after Samuel Morse, an inventor of the telegraph) is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes. For example, the letter *A* is represented by `.-`, *B* is `-...` (where the length of a dash is three times the length of a dot). We limit our scope to the letters A – Z, and disregard spaces between words.

   Your task is to write a class that handles encoding and decoding of morse code, with code sequences represented as strings of dots and dashes. Assume that the definition of the morse alphabet is available in a file named `morse.def` (in the current directory), where each line has a letter (in uppercase, morse code is case insensitive) and its code:

```
A .-
B -...
C -.-.
D -..
E .
F ..-.
G --.
H ....
I ..
J .---
K -.-
L .-..
```

   etc.

   Implement a class `Morse_code`, so that the following program

```
#include <iostream>
#include <morse.h>

int main()
{
    Morse_code mc{"morse.def"};
    std::cout << mc.encode("Hello Morse") << '\n';
    std::cout << mc.decode("... --- ...") << '\n';
    std::cout << mc.decode(".... ----") << '\n'; // ---- is not a valid code
    std::cout << mc.decode(mc.encode("loopback test")) << '\n';
}
```

   produces the output

```
.... .  .-..  .-..  ---  --  ---  .-.  ...  .
SOS
H?
LOOPBACKTEST
```

   - For encoding, any character except letters in `{A-Z,a-z}` (i.e., the letters in `morse.def` and the corresponding lower-case letters) should be ignored. In the output, the code sequences should be separated by whitespace.
   - For decoding, unrecognized code sequences should be represented by a `?` in the output. As the code does not distinguish between upper and lower-case letters you may choose if your output should be upper- or lower-case.
   - You may assume that the morse code definitions file is correctly formatted and you don't need to implement any error handling for reading the file.

   To change the case of a letter, the following functions (defined in `<cctype>`) can be used:

```
int toupper( int ch );
int tolower( int ch );
```

   where `ch` is character to be converted. If the value of `ch` is not representable as `unsigned char` and does not equal `EOF`, the behavior is undefined. The functions return the converted character or `ch` if no upper/lower-case version is defined.

4. Consider the following program:

```
#include<iostream>
#include<initializer_list>

template <typename T>
class Vektor{
public:
    Vektor(size_t sz) :size{sz},p{new T[size]{0}} {}
    Vektor(std::initializer_list<T> l) :Vektor(l.size()) {assign(l);}
    ~Vektor() {delete[] p;}
    T& operator[](size_t i) {return p[i];}
    size_t length() const {return size;}
    T* begin() {return p;}
    T* end() {return p+size;}
    const T* cbegin() const {return p;}
    const T* cend() const {return p+size;}

private:
    size_t size;
    T* p;
    void assign(const std::initializer_list<T>& l);
};

template <typename T>
void Vektor<T>::assign(const std::initializer_list<T>& l)
{
    size_t idx{0};

    for(const auto& e : l) {
        p[idx++] = e;
    }
}

template <typename T>
void add(const Vektor<T> c1, const Vektor<T> c2, Vektor<T> c3)
{
    auto it1 = c1.cbegin();
    auto it2 = c2.cbegin();
    auto it3 = c3.begin();

    while( (it1 != c1.cend() || it2 != c2.cend()) && it3 != c3.end()){
        T tmp1{};
        if(it1 != c1.cend()){
            tmp1 += *it1;
            ++it1;
        }
        T tmp2{};
        if(it2 != c2.cend()){
            tmp2 += *it2;
            ++it2;
        }
        *it3 = tmp1 + tmp2;
        ++it3;
    }
}
```

the program continues on the next page...

```
int main()
{
    Vektor<int> v1{1,2,3,4,5,6};
    Vektor<int> v2{10,20,30,40};
    Vektor<int> v3(v1.length());

    add(v1,v2,v3);
    for(auto e: v3){
        std::cout << e << " ";
    }
    std::cout << std::endl;
}
```

When the program is excuted it produces the output

```
15523872 0 33 44 5 6
```

instead of the expected:

```
11 22 33 44 5 6
```

a) Explain what happens when the program is executed.

b) Can the program be corrected so that it produces the expected output *by only changing the function template* `add`?

   If yes, show a corrected function template. If no, motivate.

c) Can the program be corrected so that it produces the expected output *by only changing the class template* `Vektor`?

   If yes, show a corrected function template. If no, motivate.

d) In the function template `add`, the temporary variables in the loop are declared as

```
T tmp1{};
T tmp2{};
```

   Would it change the behaviour of the function if they were instead declared as

```
T tmp1;
T tmp2;
```

   If yes, how? If no, motivate.

e) Show how the member function `assign` can be implemented using the algorithm `std::copy` instead of the hand-written loop.

   The declaration of `std::copy` is

```
template< class InputIt, class OutputIt >
OutputIt copy( InputIt first, InputIt last, OutputIt d_first );
```

   The return value is an iterator to the destination range, one past the last element written.