Examination
EDAF30 Programming i C++

2024-01-10, 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 $8 + 6 + 9 + 18 + 9 = 50$ 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. When trying to compile the following program

```
 1  #include <iostream>
 2  using std::cout;
 3
 4  struct A {
 5      A(int x) {val = x;}
 6      void print() {cout << "A("<<val<<")";}
 7      int val;
 8  };
 9
10
11  struct B {
12      B(int x) {a=A(x);}
13      void print() {cout << "B("; a.print(); cout << ")";}
14      A a;
15  };
16
17  int main()
18  {
19      B b(10);
20      b.print();
21      cout << '\n';
22  }
```

the compiler gives the error

```
testprogram.cpp|12 col 14 error| no matching function for call to 'A::A()'
|| B(int x) {a=A(x);}
||          ^
testprogram.cpp|5 col 5 info| candidate: A::A(int)
|| A(int x) {val = x;}
|| ^
testprogram.cpp|5 col 5 info| candidate expects 1 argument, 0 provided
testprogram.cpp|4 col 8 info| candidate: constexpr A::A(const A&)
|| struct A {
||        ^
testprogram.cpp|4 col 8 info| candidate expects 1 argument, 0 provided
testprogram.cpp|4 col 8 info| candidate: constexpr A::A(A&&)
testprogram.cpp|4 col 8 info| candidate expects 1 argument, 0 provided
```

a) Explain why the compiler gives this error

b) Show how the program can be fixed, by changing *the class* `A`.

c) Show how the program can be fixed, by changing *the class* `B`.

The program is expected to print

```
B(A(10))
```

2. Some programs use a pattern, *"source/sink"*, for managing resources. Objects are alloacated by a function, a *source*. The objects are then passed on to a *sink* which processes them and *takes ownership* of the objects. A minimal example of this is the following program, where `random_int` allocates an `int` and returns an owning pointer to it:

```
int* random_int()
{
    int* res=new int;

    *res = rand();
    return res;
}
```

The function `consume_number` prints the number and then deallocates it to avoid leaking memory:

```
void consume_number(int* p)
{
    std::cout << "consuming " << *p;
    std::cout << std::endl;

    delete p;
}
```

A minimal example program:

```
void use()
{
    int* i = random_int();
    consume_number(i);
}
```

Another common approach to resource management is that the *calling function* does the allocation, and then call-by-reference is used to get an *out-parameter*:

```
void random_int(int& out)
{
    out = rand();
}
```

An example program using this together with `consume_number` from the above example:

```
void use2()
{
    int i;
    random_int(i);
    consume_number(&i);
}
```

In this case, `use()` works as expected, but `use2()` gives an error when executed. What is the differenc that causes `use2()` to fail?

3. Java has two operations for comparing objects for equality: the equality operator == and the method `Object.equals(Object)`. The following Java method shows the common semantics

```
void compareObjects(Object a, Object b) {
  if (a == b) {
    System.out.println("a and b refer to the same object");
  }

  if (a != null && a.equals(b)) {
    System.out.println("the values of a and b are equal");
  }
}
```

   This problem is about how the corresponding comparisons are implemented in C++.

   a) Show how one, in a C++ function, can determine if two parameters, `a` and `b`, are *the same object*. (I.e., give an expression that corresponds to `a == b` in Java[1].)
   b) Show how one, in a C++ function, can determine if two parameters, `a` and `b`, have *the same value*. (I.e., give an expression that corresponds to `a.equals(b)` in Java.) You may assume that the comparison is possible for the type of `a` and `b`.
   c) Give a C++ implementation that corresponds to the Java method `compareObjects`. You may limit the implementation to what is necessary to make the following program work, but state clearly what assumptions and limitations you make and what is required of the types for which your implementation of `compareObjects` works.

   Main program:

```
#include <string>
#include <vector>
#include <utility>
#include "compareObjects.h"
int main()
{
  int x{10};
  int y{10};
  int& r=x;

  compareObjects("x,x", x, x);
  compareObjects("x,y", x, y);
  compareObjects("x,r", x, r);
  compareObjects("y,r", y, r);

  std::vector<int> v1{x,y};
  std::vector<int> v2{10,r};
  std::vector<int> v3{10,5};

  compareObjects("v1,v2)", v1, v2);
  compareObjects("v1,v3)", v1, v3);

  std::pair<int,std::string> p1{10,"Hello"};
  std::pair<int,std::string> p2{10,"Hello"};
  std::pair<int,std::string> p3{10, "Hej"};

  compareObjects("p1,p2", p1,p2);
  compareObjects("p1,p3", p1,p3);
}
```

   Expected output:

```
compareObjects(x,x):
- the objects are equal
- the same object
compareObjects(x,y):
- the objects are equal
- different objects
compareObjects(x,r):
- the objects are equal
- the same object
compareObjects(y,r):
- the objects are equal
- different objects
compareObjects(v1,v2)):
- the objects are equal
- different objects
compareObjects(v1,v3)):
- the objects are not equal
- different objects
compareObjects(p1,p2):
- the objects are equal
- different objects
compareObjects(p1,p3):
- the objects are not equal
- different objects
```

   Note: in the above example, compareObjects take an extra string argument to make the expected output clearer. You may leave this out and just give an implementation similar to the Java method.

[1] but disregarding Java's special semantics for comparison of primitive types.

4. This question studies an algorithm with the following declaration:

```
template <typename InputIt, typename OutputIt, typename Pred>
std::pair<InputIt, OutputIt>
copy_while(InputIt first, InputIt last, OutputIt out, Pred p)
```

It takes an iterator range, an output iterator, and a unary predicate, and copies elements from the input range to the output until the first element for which the predicate returns `false`.
The return value is one past the last elements copied, in the respective range. I.e., the InputIt refers to the first element not satisfying the predicate (or `last` if the entire range was copied) and the OutputIt is one past the last element written.

a) First, we will use `copy_while`. Write a function
`std::vector<int> take_while_sum_less_than(std::vector<int>& v, int n)`
that uses `copy_while` to move[2] the first k elements from `v` to a new `std::vector`, such that the sum of the moved elements is less than n, and k is as large as possible.
For example, with `v = {1,2,3,4,5,6,7,8,9}`, and $n = 15$, the result should contain `{1,2,3,4}` and after the call, `v = {5,6,7,8,9}`.
With `v={10,11,12}` and $n = 10$, the returned vector should be empty and v unchanged.
In code,
```
    std::vector<int> v{1,2,3,4,5,6,7,8,9};
    auto w = take_while_sum_less_than(v, 15);
```
should result in the two vectors containing the following elements:
```
    v: 5 6 7 8 9
    w: 1 2 3 4
```
Hint: The predicate can in this case be a stateful function, keeping track of the sum of the copied elements.

b) Implement `copy_while`. An example use of it is

```
void example()
{
    std::vector<int> v{1,3,5,2,4,6};
    std::ostream_iterator<int> out(std::cout, " ");

    std::cout << "The first sequence of odd numbers is: ";
    auto res = copy_while(begin(v), end(v), out, [](int x){return x%2;});
    std::cout << "\n";
    std::cout << "The first even number is " << *res.first << std::endl;
}
```

which should output

```
The first sequence of odd numbers is: 1 3 5
The first even number is 2
```

[2] Here `move` has nothing to do with move semantics, it simply means copy to the destination and remove from the source.

5. In many algorithms, such as `find_if`, `copy_if`, and `remove_if`, you use a predicate to identify the element(s) to be found, copied, or removed.

   Sometimes, it is convenient to have a function that creates such predicates. One example of such a "predicate factory" is a function `is_char_from(std::string s)` which takes a string `s` as argument and returns a predicate that takes a character as argument and returns `true` if that character is in the string `s`. The function has the following declaration:

   ```
   std::function<bool(char)> is_char_from(std::string s);
   ```

   Implement `is_char_from(std::string)` as defined above so that the following unit test works.

   ```
   #include <cassert>
   void test_is_char_from()
   {
       auto f = is_char_from("abc");
       assert(f('a'));
       assert(f('b'));
       assert(f('c'));
       assert(!f('d'));
       assert(!f('e'));
       std::cout << "is_char_from OK\n";
   }
   ```

   Hint: The class `std::function<bool(char)>` has a constructor that can be called with anything that can be called as a function that takes a `char` and returns `bool`.