Lösningsförslag, Tentamen i C++

2017–04–20

1. Eftersom funktioner i en subklass inte överlagrar funktioner med samma namn i superklassen (utan döljer dem) så har klassen `D` bara en funktion `f(double)`. Alltså kommer anropet `f(2)` att göra en implicit konvertering av värdet 2 till en `double`, och programmet skriver ut

```
3.3
3.6
```

2. a) In `print1` the container is `const`, but in the class `operator[]` is non-`const`. Add the function:

```
T operator[](size_t i) const { return arr[i]; }
```

or, possibly return `const T&` if lifetimes allow and the object may be expensive or impossible to copy.

b) Add the definition of the iterator type and of the `begin` and `end` functions.

```
using const_iterator = const T*;
const_iterator begin() const { return arr; }
const_iterator end() const { return arr + N; }
```

c) Yes. range for uses `begin()` and `end()` and, as the loop variable is a copy, using the `begin() const` and `end() const` is OK.

d) Add a constructor taking a `std::initializer_list`

```
template <typename T, size_t N>
class array {
public:
    ...
    array() = default;
    array(std::initializer_list<T>) ;
    ...
};

template <typename T, size_t N>
array<T,N>::array(std::initializer_list<T> il)
{
    if(il.size() != N)
        throw std::range_error("size mismatch");
    size_type t = 0;
    for(auto x : il) {
        arr[t++]=x;
    }
}
```

3. **a**. I konstruktorn görs ingen initiering av variabeln `a`, utan istället görs en *tilldelning*. Eftersom objektet måste konstrueras fullständigt innan funktionskroppen för konstruktorn körs så anropas implicit default-konstruktorn `A::A()`, men denna finns inte.

b. Man kan lösa problemet i `A` genom att ge den en default-konstruktor. Två exempel:

```
A(int x=0) {val = x;}
```

eller

```
A() = default;
A(int x) {val = x;}
```

Här fungerar det med tilldening i konstruktorkroppen, men man bör alltid skriva konstruktorer med initiering, d v s `A(int x=0) :val{x} {}`.

c. För att lösa probemet i `B` gör man initiering av medlemmen `a` i stället för tilldelning:

```
B(int x) :a{A(x)} {}
```

4. a) En funktion som läser från en ström, partitionerar och skriver ut

```
void test_partition(std::istream& is)
{
    std::vector<int> v;
    int i;
    while(is >> i) {
        v.push_back(i);
    }
    auto r = partition(v.begin(), v.end(), [](int x) {return x%2==0;});

    cout << "result:\n";
    for(auto x : v) {
        cout << x << " ";
    }
    cout << "\n";
}
```

Som alternativ till lambda-uttrycket kan en funktionspekare eller ett funktions-objekt användas. (Variabeln `r` används i uppgift b.)

b) Lägg till följande sist i `test_partition`.

```
    std::sort(v.begin(), r,std::greater<int>{});
    std::sort(r,v.end(), std::greater<int>{});

    cout << "sorted:\n";
    for(auto x : v) {
        cout << x << " ";
    }
    cout << "\n";
```

c) implementation av `partition`:

```
template <typename T, typename C>
T mypartition(T b, T e, C c)
{
    auto pos = b;
    while(b != e){
        if(c(*b)) {
            std::iter_swap(pos++,b);
        }
        ++b;
    }
    return pos;
}
```

5.
```
size_type Editor::find_left_par(size_type pos) const {
      char thechar = text.at(pos);
      char matching;
      int depth{1};
      switch(thechar) {
          case ')':
              matching='(';
              break;
          case ']':
              matching='[';
              break;
          case '}':
              matching='{';
              break;
          default:
              return std::string::npos;
      }
      ssize_t p = pos;  // NB! signed
      do {
          char c = text.at(p);
          if(c == matching && --depth == 0) return p;
          else if(c == thechar)  ++depth;
          --p;
      } while (p >= 0); // posiiton 0 is legal in a string
      return std::string::npos;
  }
```