Tentamen
C++-programmering

2017–03–17, 14.00–19.00

Hjälpmedel: En valfri C++-bok. OH-bilderna från föreläsningarna är *inte* tillåtna.

Du ska i dina lösningar visa att du behärskar C++ och att du kan använda C++ standardklasser. ”C-lösningar” ger inga poäng, även om de är korrekta.

Uppgifterna ger preliminärt $12 + 14 + 10 + 6 + 8 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42).

---

1. STL-funktionen `adjacent_difference` har följande beskrivning (adjacent betyder intilliggande):

```
/*
 * Computes the differences between adjacent values in the range
 * [first, last) using operator-() and writes the result to result.
 * Returns an iterator pointing just behind the values written to result.
 *
 */
template<typename InputIterator, typename OutputIterator>
OutputIterator adjacent_difference(InputIterator first, InputIterator last,
        OutputIterator result);
```

   Det första värdet i talföljden flyttas oförändrat till `result`. Exempel på användning:

```
int a[] = {1, 1, 2, 3, 5, 9};
int b[6];
adjacent_difference(a, a + 6, b); // b becomes {1, 0, 1, 1, 2, 4}
```

   a) Implementera `adjacent_difference`.

   b) Använd `adjacent_difference` i ett program som läser ett antal heltal och producerar ett differensschema. Om talen `1 1 2 3 5 9` läses ska utskriften ha följande form:

```
1 1 2 3 5 9
0 1 1 2 4
1 0 1 2
-1 1 1
2 0
-2
```

2. Europas alla rojalister behöver verktyg för att hålla reda på tronföljden i de olika kungahusen i världsdelen. Din uppgift blir därför att hjälpa dem med programvara för att generera listor över tronföljden i de olika länderna.

   a) Skriv en klass `Ruler` som kan användas för att hålla reda på och läsa in/skriva ut uppgifter om en enskild regents regeringstid. Uppgifter som sparas om en regent är i tur och ordning:

      - namn (innehåller *ej* whitespace)
      - ordningsnummer (vanliga arabiska siffror används)
      - nation
      - årtal, trontillträde
      - årtal, tronfrånträde (eller 0 om regenten fortfarande sitter på tronen)

   Klassen ska fungera tillsammans med följande enkla huvudprogram:

```
int main() {
    Ruler r1("Gustav-Adolf",6,"Sweden",1950,1973);
    std::cout << r1 << std::endl;

    Ruler r2;
    std::cin >> r2;
    std::cout << r2 << std::endl;
}
```

   Ett exempel på en körning av programmet kan se ut så här:

```
$ ./ruler_simple
Gustav-Adolf 6 of Sweden, 1950-1973
Karl 12 Sweden 1697 1718
Karl 12 of Sweden, 1697-1718
```

   b) En textfil innehåller uppgifter om europeiska regenter i osorterad ordning. Skriv ett huvudprogram som läser in uppgifterna och skriver ut en sorterad tronföljdslista på standard output i enlighet med följande instruktioner:

      - Ange namnet på filen som ett argument på kommandoraden.
      - Varje rad i filen innehåller uppgifter om en regents regeringstid på följande format: `namn ordningsnummer nation tillträdesår frånträdesår`
      - I utskriften ska regenterna sorteras i första hand efter nation och i andra hand efter tillträdesår samt skrivas ut med en regent per rad.
      - Använd klassen `Ruler` för att lagra information om enskilda regenter samt för inläsning och utskrift av regentdata.
      - Använd standardbibliotekets algoritmer, containrar och iteratorer när så är möjligt (och rimligt).
      - Redovisa (implementera) även eventuella nödvändiga hjälpfunktioner eller tillägg till klassen `Ruler` du kan behöva.

3. Som du vet kan man kopiera från ett område och få de kopierade talen inlagda sist i en behållare enligt följande exempel (här är behållaren en vektor, men det fungerar med alla behållare som har operationen `push_back`):

```
int a[] = { 1, 2, 3, 4, 5 };
vector<int> v;
copy(a, a + 5, back_inserter(v));
```

   Funktionen `back_inserter` skapar ett `back_insert_iterator`-objekt. När operationen = anropas på detta objekt (inifrån `copy`) anropas `v.push_back`.

   Implementera klassen `back_insert_iterator` och funktionen `back_inserter`. De operationer som förekommer i följande funktion måste implementeras:

```
template<typename In, typename Out>
Out copy(In first, In last, Out dest) {
   for (; first != last; ++first) {
      *dest = *first;
      ++dest; // dest++ skall också fungera
   }
}
```

4. Betrakta följande kodfragment:

```
class A {
public:
    A(int i) { x = i; }
    int x;
};

class B {
public:
    B(int i) { a = i; }
    A a;
};
```

   Vid kompilering av koden får man ett kompileringsfel på raden "`B(int i) { a = i; }`".

   a) Förklara varför kompileringsfelet uppstår.

   b) Hur ska kodfragmentet ovan ändras så att det blir korrekt?

5. Betrakta följande klassdefinition:

```
class Foo : public SuperFoo {
  Bar* fBar1;
  Bar* fBar2;
  ~Foo() {delete fBar1; delete fBar2;}
  // ... medlemmar utan betydelse för uppgiften
};
```

   där objektet är ägare till de objekt som pekarna pekar på. En invariant för klassen `Foo` är att båda pekarna `fBar1` och `fBar2` måste vara giltiga pekare (d v s peka på objekt).

   Klassen `Bar` har medlemmarna `Bar::Bar(const Bar&)` och `Bar& Bar::operator=(const Bar&)` och är inte polymorf.

   Skriv en tilldelningsoperator för klassen Foo: `Foo& Foo::operator=(const Foo& that)`.

   Ledning: Tänk på minnesläckor, att minnesallokeringar kan ge exceptions och på att superklassen, `SuperFoo`, också kan ha datamedlemmar.