Tentamen
C++-programmering

2015–03–21, 8.00–13.00

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 $10 + 10 + 18 + 12 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42).

---

1. Standardalgoritmen `remove_if` har följande beskrivning:

```
template <typename ForwardIterator, typename UnaryPredicate>
ForwardIterator remove_if(ForwardIterator first, ForwardIterator last,
                          UnaryPredicate pred);
```

   Transforms the range `[first,last)` into a range with all the elements for which `pred` returns `true` removed, and returns an iterator to the new end of that range.

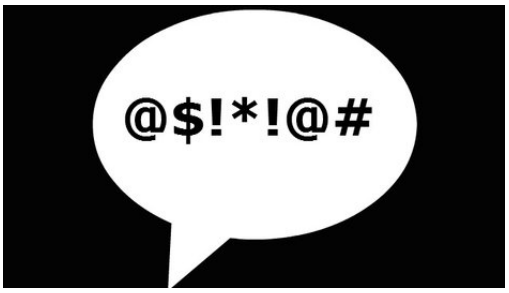
   The function cannot alter the properties of the object containing the range of elements (i.e., it cannot alter the size of an array or a container). The removal is done by replacing the elements for which `pred` returns `true` by the next element for which it does not, and signaling the new size of the shortened range by returning an iterator to the element that should be considered its new past-the-end element.

   The relative order of the elements not removed is preserved, while the elements between the returned iterator and `last` are left in a valid but unspecified state.

   a) `v` är en `vector<int>`. Skriv en programrad (eller två) som tar bort alla tal < 0 från vektorn och ändrar vektorns storlek. Använd `remove_if` och en annan funktion.

   b) Implementera funktionen `remove_if`.

2. Svordomar kan se ut så här:



   Skriv en funktion som tar en sträng som parameter och ändrar den genom att byta ut alla vokaler (aeiouyåääöAEIOUYÅÄÖ) mot tecken som slumpmässigt väljs bland tecknen `@$!*#&%`.

3. ”Bulgarisk patiens” är ett läggspel med enkla regler:[1]

   - Tag N kort, fördela dem slumpmässigt på ett antal högar (1..N stycken, antalet väljs slumpmässigt). Det ska vara minst ett kort i varje hög. Vi förutsätter i fortsättningen att N är ett triangeltal, det vill säga att $N = 1 + 2 + \cdots + k$ för något k.
   - Upprepa tills antalet kort i högarna är $1, 2, \ldots, k$ i någon ordning:
     - Tag ett kort från varje hög, lägg dem i en ny hög.

   I nedanstående exempel visas först antalet kort i högarna när 10 kort fördelats på 6 högar. Sedan visas antalet kort i högarna efter varje drag och till sist antalet drag. ”Tomma” högar finns inte med.

```
2 1 2 1 2 2
1 1 1 1 6
5 5
4 4 2
3 3 1 3
2 2 2 4
1 1 1 3 4
2 3 5
1 2 4 3
8 moves.
```

   Utskriften har genererats av följande program (med `n` = 10):

```
int main() {
    cout << "Number of cards: ";
    int n;
    cin >> n;
    BulgarianSolitaire bs(n);
    cout << bs << endl;
    int moves = 0;
    while (!bs.atGoal()) {
        bs.move();
        cout << bs << endl;
        ++moves;
    }
    cout << moves << " moves." << endl;
}
```

   Implementera klassen `BulgarianSolitaire`. Anvisningar:

   - Antalet kort i högarna *ska* representeras av en `vector<int>`. Tomma högar ska inte vara med (det vill säga vektorn får inte innehålla några nollor).
   - Använd standardalgoritmer så långt som möjligt. Det finns inga krav på effektivitet hos programmet.

[1] `http://en.wikipedia.org/wiki/Bulgarian_solitaire`

4. Som du vet kan man kopiera från ett område och få de kopierade värdena inlagda sist i en STL-behållare enligt följande exempel (här kopierar man till en vektor, men det fungerar med alla behållare som har operationen `push_back`):

```
int main() {
    int from[] = {9, 9, 2, 7, 8, 4, 5, 1, 6, 3};
    vector<int> to = {1, 3, 5, 7};
    copy(begin(from), end(from), back_inserter(to));
    copy(to.begin(), to.end(),
            ostream_iterator<int>(cout, " ")); // 1 3 5 7 9 9 2 7 8 4 5 1 6 3
    cout << endl;
}
```

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

Skriv en klass `sort_insert_iterator` och en funktion `sort_inserter` som fungerar som en `back_inserter` men som inte lägger in ett värde sist (med `push_back`) utan i stället sorterar in värdet på rätt plats i växande ordning (med `insert`). Värden ska jämföras med operatorn <. Om man i ovanstående program använder en `sort_inserter` i stället för en `back_inserter` ska utskriften bli `1 1 2 3 3 4 5 5 6 7 7 8 9 9`. Det ska fungera även om `to` är en `list` i stället för en `vector`.

Klassen ska naturligtvis vara en template-klass och funktionen en template-funktion. Bara de operationer som förekommer i funktionen `copy` ska implementeras:

```
template<typename In, typename Out>
Out copy(In first, In last, Out dest) {
    while (first != last) {
        *dest = *first;
        ++dest;
        ++first;
    }
    return dest;
}
```

Ledning: operationerna * och ++ gör ingenting (mer än att returnera objektet som de utförs på).