LUNDS TEKNISKA HÖGSKOLA Institutionen för datavetenskap

Tentamen
C++-programmering

2013–03–16, 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 $10 + 10 + 17 + 13 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42).

---

1. STL-algoritmen `is_partitioned` (ny i C++11) har följande beskrivning:

```
template <typename InputIt, typename UnaryPredicate>
bool is_partitioned(InputIt first, InputIt last, UnaryPredicate p)
// Returns true if all elements in the range [first, last) that satisfy
// the predicate p appear before all elements that don't. Also returns true
// if [first, last) is empty. The time complexity is linear.
```

   Antag att predikatet `p` är `true` för udda element, `false` för jämna element. Då är sekvensen $\{1, 3, 1, 42, 4\}$ partitionerad men inte sekvensen $\{1, 3, 42, 1, 4\}$.

   a) Skriv en funktion som returnerar `true` om alla udda tal i en vektor (av typen `vector<int>`) kommer före alla jämna tal. Du ska naturligtvis utnyttja `is_partitioned`.

   b) Implementera funktionen `is_partitioned`.

2. I ett simuleringsprogram inträffar händelser vid olika tidpunkter. En händelse beskrivs av den abstrakta klassen `Event`. Användaren utnyttjar `Event` som basklass till sina egna händelseklasser. Ett objekt av klassen `Scheduler` har en lista med alla händelser och ser till att händelserna exekveras i tidsordning. Schematiskt ser det ut så här:

```
class Event {
public:
    explicit Event(unsigned long time);
    virtual ~Event();
    unsigned long getTime() const;
    virtual void action() = 0;
};

class Scheduler {
public:
    void insertEvent(Event* e);
    void actionLoop();
};
```

(forts. nästa sida)

Användaren skapar dynamiska objekt av sina händelseklasser, gör `insertEvent` på dem och anropar sedan `actionLoop`. Denna funktion tar fram det händelseobjekt som har minst `time` och anropar `action` på det, varefter objektet tas bort. Detta fortsätter så länge det finns händelseobjekt kvar i listan. (En händelse kan medföra att nya händelser ska inträffa senare, dvs `action` kan lägga in nya objekt i händelselistan. Om det inte vore så skulle det inte bli särskilt intressant.)

Implementera klassen `Scheduler`. Det är viktigt att implementeringen är effektiv, så du ska använda en prioritetskö för att lagra pekarna på händelseobjekten. När man definierar en prioritetskö kan man ange vilken funktion som ska användas för att sortera objekten i kön: `priority_queue<T, Sequence, Compare>`. Parametrarna har följande betydelse (tabellen är hämtad från SGI:s STL-dokumentation):

Parameter	Description	Default
`T`	The type of object stored in the priority queue.	
`Sequence`	The type of the underlying container used to implement the priority queue.	`vector<T>`
`Compare`	The comparison function used to determine whether one element is smaller than another element. If `Compare(x,y)` is `true`, then `x` is smaller than `y`. The element returned by `Q.top()` is the largest element in the priority queue. That is, it has the property that, for every other element `x` in the priority queue, `Compare(Q.top(), x)` is `false`.	`less<T>`

3. En klass `String` håller reda på en följd av tecken på följande sätt:

```
class String {
public:
    ...
private:
    char* chars; // array of characters
    size_t n;    // the number of characters in the array
};
```

Du ska förse klassen med en iterator som kan användas för att hämta (men inte ändra) *ord* ur en `String`. Ett ord definieras som en följd av icke-blanka tecken (mellan orden finns blanka, inte ”whitespace”).

Du behöver bara implementera de konstruktioner som krävs för att nedanstående program ska fungera. Observera att en iterator inte får kopiera strukturen som den itererar över. Du måste också göra tillägg till klassen `String`; dessa ska du redovisa. Däremot behöver du inte implementera funktionen `getline` så att den fungerar med `String`-objekt.

```
int main() {
    using namespace std;
    String line;
    while (getline(cin, line)) {
        for (String::word_iterator wi = line.wi_begin();
                wi != line.wi_end(); ++wi) {
            cout << *wi << " "; // *wi is a std::string object
        }
        cout << endl;
    }
}
```

Exempel på indata och resultat (de raka strecken anger början och slutet på indata och utskrift):

```
|asdf ghjk qwer|  => |asdf ghjk qwer|
|   abc   def   | => |abc def|
|      |          => ||
||                => ||
```

4. En baklängesordlista är en ordlista där orden är sorterade efter sina avslutningar. Ord som slutar på `a` kommer före ord som slutar på `b`, ord som slutar på `ma` kommer före ord som slutar på `na`, och så vidare. En sådan ordlista kan vara bra att ha för korsordslösare och för poeter som ska hitta rimord.

   I filen *words.txt* finns ett antal engelska ord åtskilda av ”whitespace”. Skriv ett program som läser filen och skapar en baklängesordlista med ett ord per rad på filen *backwords.txt*. Det ska inte finnas några dubbletter i listan. Stora bokstäver ska bytas mot motsvarande små bokstäver. C-funktionen `char tolower(char ch)` konverterar en stor bokstav till motsvarande lilla bokstav; om `ch` inte är en stor bokstav returneras tecknet oförändrat.

   I programmet *ska* du använda STL-konstruktioner så långt det är möjligt (inga `for`- eller `while`-satser).