Tentamen
C++-programmering

2015–05–08, 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 $6 + 9 + 18 + 5 + 12 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42).

---

1. I ett bildbehandlingsprogram behöver man hålla reda på många bilder samtidigt. Det har visat sig att programmet är mycket långsamt. Man har skalat bort bit efter bit av programmet för att ta reda på vad det beror på, och det visar sig att nedanstående lilla program tar mer än 5 sekunder att exekvera.

```
class Image {
public:
    Image(int ww, int hh) : w(ww), h(hh), pixels(new int[w * h]) {}
    ~Image() { delete[] pixels; }
    Image(const Image& rhs) : w(rhs.w), h(rhs.h), pixels(new int[w * h]) {
        copy(rhs.pixels, rhs.pixels + w * h, pixels);
    }
private:
    int w; // image width
    int h; // - height
    int* pixels;
};

Image readImage(int nbr) {
    Image temp(1000, 1000);
    // ... read the image into temp
    return temp;
}

int main() {
    vector<Image> images;
    for (int i = 0; i != 1000; ++i) {
        images.push_back(readImage(i));
    }
}
```

   a) Genom att lägga till en rad i `main`-funktionen kan man få ner exekveringstiden till ungefär hälften. Vilken rad ska man lägga till? Förklara vad förbättringen beror på.

   b) Genom att göra ett tillägg till klassen `Image` kan man få ner exekveringstiden till några hundradels sekunder. Vilket tillägg ska man göra? Förklara vad förbättringen beror på. Anmärkning: tillägget som ska göras är en konstruktion som infördes i C++11.

2. I standardbiblioteket finns en funktion `max_element(first, last)` som returnerar en iterator som pekar på platsen för det största värdet i följden `[first, last)`. Skriv en liknande funktion som returnerar två värden: dels *värdet* av det största elementet, dels *antalet gånger* som detta element förekommer (noll om följden är tom). Tips: typen på det värde som en iterator av typen `It` pekar på får man reda på med `iterator_traits<It>::value_type`.

   Funktionen får bara göra ett pass över indata. Ange vilken typ av iterator som funktionen använder.

3. I Java finns en klass `BigInteger` som beskriver ”stora heltal” som kan innehålla ett godtyckligt antal siffror. Skriv en klass `BI` som är en förenklad version av denna klass. Till exempel hanterar klassen bara positiva tal. Man ska kunna använda klassen så här:

```
int main() {
    BI b1;
    BI b2(999987);
    BI b3("123456789000123456789");
    cout << b1 << " " << b2 << " " << b3 << endl; // 0 999987 123456789000123456789

    BI b4 = b2;
    BI b5;
    b5 = b3;
    cout << b4 << " " << b5 << endl; // 999987 123456789000123456789

    b2 += BI(13);
    cout << b2 << endl; // 1000000

    cout << 111111111 + b3 << endl; // 123456789000234567900
    cout << b3 + 111111111 << endl; // 123456789000234567900

    cout << "Print a number: ";
    cin >> b1;          // for example 111222333444555666
    cout << b1 << endl; // 111222333444555666
}
```

   Följande regler *ska* följas:

   - Det är viktigare att programmet är lättläst än att det är effektivt.
   - Du ska visa både `.h`-filen (utan några implementeringar) och `.cc`-filen (med alla implementeringar). Du behöver dock inte ange någon ”include guard” eller några `include`-direktiv.
   - Du ska bara implementera de funktioner som krävs för att programmet i exemplet ska fungera. Men försök generalisera parametrarna så att man till exempel kan skriva `b2 += 13` eller `b2 + b3`.
   - Representera talen i basen 10 genom att lagra de decimala siffrorna i en `vector<unsigned char>`. Detta är inte minneseffektivt, men det gör implementeringen enklare. Talets *minst* signifikanta siffra ska lagras i det första vektorelementet.
   - Vektorn ska alltid innehålla exakt lika många element som det finns siffror i talet. Inledande icke-signifikanta nollor ska inte finnas med.
   - Om något fel inträffar (felaktigt tecken i ett konstruktoranrop eller vid inläsning) ska ett exception av typen `illegal_number` kastas.

4. Antag att man i uppgift 3 vill lagra talets siffror i en dynamiskt allokerad array i stället för i en vektor. Visa vilka ändringar som man behöver göra i `.h`-filen. Du ska inte implementera något.

5. Vid Morsesignalering, som inte används mycket nuförtiden, kodas texten som ska överföras enligt Morsealfabetet. Varje tecken kodas som en följd av korta och långa signaler (”punkter” och ”streck”). Till exempel motsvaras tecknet `a` av `.-`, b av `-...`, och så vidare. Vi begränsar oss i fortsättningen till bokstäverna a–z och struntar i mellanrum mellan ord.

   Skriv en klass `MorseCode` som kan översätta en Morsesignal (punkter och streck) till text. Exempel på användning:

```
int main() {
    MorseCode mc;
    string code = ".... . .--- -.. ..- --. .-.. .- -.. .";
    cout << mc.decode(code) << endl; // hejduglade
}
```

   Anvisningar:

   - Definitionerna av koderna finns i en fil *morse.def* i den aktuella katalogen. Filen har följande format:

```
a .-
b -...
c -.-.
d -..
e .
f ..-.
... osv
z --..
```

   - En Morsekod som inte finns i tabellen ska översättas till `?`. T ex ska `mc.decode("---- .....")` ge resultatet `"??"`.