Tentamen
C++-programmering

2014–03–15, 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 $15 + 15 + 5 + 15 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42).

---

1. I laboration 1 har du implementerat klassen `List`, som beskriver en enkellänkad lista av heltal. Detta är en del av klassens specifikation:

```
class List {
public:
    // ... constructor, destructor, operations

    List(const List&) = delete;
    List& operator=(const List&) = delete;
private:
    /* a list node */
    struct Node {
        int value;  // the node value
        Node* next; // pointer to the next node, nullptr in the last node
        Node(int v, Node* n) : value(v), next(n) {}
    };

    Node* first; // pointer to the first node
};
```

   a) I de båda deklarationerna i `public`-delen av klassen står det `= delete`. Vad medför dessa specifikationer?

   b) Om man tar bort `delete`-specifikationerna måste man implementera de båda funktionerna. Gör det.

   c) Ge ett exempel där de båda funktionerna används.

2. Om ”lyckliga tal” från Wikipedia (förkortat): A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers, while those that do not end in 1 are unhappy numbers (or sad numbers).

   For example, 19 is happy, as the associated sequence is:

$$1^2 + 9^2 = 82$$
$$8^2 + 2^2 = 68$$
$$6^2 + 8^2 = 100$$
$$1^2 + 0^2 + 0^2 = 1$$

   The 143 happy numbers up to 1,000 are:

   1 7 10 13 19 23 28 31 32 44 49 68 70 79 82 86 91 94 97 100 103 109 129 130 133 139 167 176 188 190 192 193 203 208 219 226 230 236 239 262 263 280 291 293 301 302 310 313 319 320 326 329 331 338 356 362 365 367 368 376 379 383 386 391 392 397 404 409 440 446 464 469 478 487 490 496 536 556 563 565 566 608 617 622 623 632 635 637 638 644 649 653 655 656 665 671 673 680 683 694 700 709 716 736 739 748 761 763 784 790 793 802 806 818 820 833 836 847 860 863 874 881 888 899 901 904 907 910 912 913 921 923 931 932 937 940 946 964 970 973 989 998 1000

   The happiness of a number is preserved by rearranging the digits, and by inserting or removing any number of zeros anywhere in the number. The unique combinations of the happy numbers below 1,000 follow (the rest are just rearrangements and/or insertions of zero digits):

   1 7 13 19 23 28 44 49 68 79 129 133 139 167 188 226 236 239 338 356 367 368 379 446 469 478 556 566 888 899

   a) Skriv en funktion som avgör om ett (positivt) heltal är ett lyckligt tal.

   b) Skriv ett program som skriver ut de unika lyckliga talen ≤ 1000 (som den andra listan ovan). Du ska naturligtvis använda funktionen från uppgift a.

3. I ett program ska en dators minne simuleras med en klass med följande specifikation:

```
struct AddressingError {}; // exception, thrown when an invalid address is given

class Memory {
public:
    using address = unsigned long; // memory address
    using byte = unsigned char;    // memory contents

    Memory(address size); // Creates a memory with size cells
    ~Memory();            // Destroys the memory

    /* Reads a cell. Returns a random number if nothing has been
       written in the cell */
    byte read(address addr) const;

    /* Writes a cell */
    void write(address addr, byte b);
};
```

   Implementera klassen. Minnet kan vara mycket stort, många gigabyte, men bara en liten del av minnescellerna används. Det finns inga stora krav på slumpmässighet på slumptalet som kan returneras av `read` (så `rand()` går bra att använda).

4. Filerna i standardbiblioteket som innehåller funktionerna för sortering (`sort`, `stable_sort` och liknande) har av någon anledning försvunnit. För att vi ska kunna sortera heltalsvektorer (arrays) har vi skrivit en funktion som sorterar en vektor med urvalssortering:

```
void my_sort(int* beg, int* end) {
    for (; beg != end - 1; ++beg) {
        int* min = beg;
        for (int* pos = beg + 1; pos != end; ++pos) {
            if (*pos < *min) {
                min = pos;
            }
        }
        std::swap(*beg, *min);
    }
}
```

   a) Ge ett exempel på anrop av funktionen.

   b) Funktionen innehåller ett mindre fel (det finns ett specialfall som funktionen inte klarar). Korrigera det felet.

   c) Skriv om funktionen så att den kan 1) sortera annat än heltal, 2) sortera annat än arrays, 3) jämföra värden på annat sätt än med $<$-operatorn. Det ska fortfarande vara möjligt att anropa funktionen med två parametrar. Ingen kod ska dupliceras mellan de båda funktionerna.
   Exempel:

```
vector<double> v = {3.5, 2.3, 1.0, 4.6, 4.5};
my_sort(v.begin(), v.end());

my_sort(v.begin(), v.end(), [](double a, double b) { return a > b; });
```

   d) Klassen `Point` har funktionerna `get_x()` och `get_y()`. Skriv ett anrop av `my_sort` som sorterar en vektor med `Point`-objekt efter x-koordinat och, om x-koordinaterna är lika, efter y-koordinat.

   e) Kan man använda `my_sort` för att sortera länkade listor (standardklassen `list`)? Om inte, korrigera funktionen så att det blir möjligt.