Tentamen
EDAF30 Programmering i C++

2015–01–14, 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 även att du kan använda C++ standardklasser. Rena ”C-lösningar” på problem som med fördel kan lösas enligt mera objektorienterade principer kan därför ge poängavdrag, även om de är korrekta.

Uppgifterna ger preliminärt $10 + 12 + 20 + 8 = 50$ poäng. För godkänt krävs preliminärt 25 poäng (betygsgränser 3/25, 4/33, 5/42). Observera att ordningsföljden eller poängantalet för uppgifterna inte nödvändigtvis avspeglar deras svårighet.

---

1. Slumptal kan användas för att kryptera texter. Man behöver en slumptalsgenerator som kan initieras med ett ”slumptalsfrö” så att den kan upprepa följden av slumptal. Använd funktionerna som beskrivs i manualsidan på sidan 4.

   Kryptering av en text går till på följande sätt:

   - välj en krypteringsnyckel, ett `int`-tal `key`,
   - skapa en slumptalsgenerator med slumptalsfröet `key`,
   - för varje tecken i texten: dra ett slumptal, addera det till tecknet.

   Vi förutsätter att teckenkoderna för tecknen ligger i intervallet $[0, 255]$ och att slumptalen ska ligga i samma intervall. För att också de krypterade tecknen ska hålla sig inom intervallet ska additionen göras ”cykliskt”, dvs att om summan av teckenkoden för tecknet och slumptalet blir större än 255 så blir det krypterade tecknet summan-256. Exempel (teckenkoderna har skrivits i decimal form):

Tecken:	A	t	t	a	c	k	!
Teckenkod:	65	116	116	97	99	107	33
Slumptal:	4	207	6	1	12	255	8
Krypterad kod:	69	67	122	98	111	106	41
Krypterat tecken:	E	C	z	b	o	j	)

   Skriv en funktion som krypterar en text utgående från krypteringsnyckeln `key`. Funktionen ska ha följande rubrik:

   ```
   string encrypt(const string& text, unsigned int key);
   ```

2. Betrakta följande vektorklass:

```
template <typename T>
class Vector {
public:
    Vector(size_t s) : v(new T[s]), sz(s) {
        for (size_t i = 0; i != sz; ++i) {
            v[i] = T();
        }
    }
    ~Vector() { delete[] v; }
    size_t size() const { return sz; }
    T get(size_t i) const { return v[i]; }
    void put(size_t i, T val) { v[i] = val; }
private:
    T* v;
    size_t sz;
};
```

a) När klassen används i följande program får man ett oväntat resultat:[1]

```
void print(Vector<int> v) {
    for (size_t i = 0; i != v.size(); ++i) {
        cout << v.get(i) << " ";
    }
    cout << endl;
}

int main() {
    Vector<int> v1(10);
    for (size_t i = 0; i != v1.size(); i++) {
        v1.put(i, i + 1);
    }
    print(v1);    // utskrift 1 2 3 4 5 6 7 8 9 10
    Vector<int> v2(5);
    print(v2);    // utskrift 0 0 0 0 0
    print(v1);    // utskrift 0 805306368 0 805306368 3 6 7 8 9 10
}
```

Den sista `print(v1)`-satsen borde ha gett samma utskrift som den första! Förklara varför det har blivit fel (du behöver inte förklara vad de stora värdena står för) och korrigera *klassen* så att programmet fungerar som förväntat.

Ledning: Felet är relaterat till hur man hanterar minne i C++.

b) Det är inte elegant att behöva utnyttja funktionerna `get` och `put` för att komma åt vektorelementen. Lägg till konstruktioner i klassen så att man kan komma åt vektorelementen med indexering, till exempel `v1[i] = i + 1`. Lägg också till en funktion så att man kan skriva ut vektorer, till exempel `cout << v2 << endl << v1 << endl`.

[1] Programmet är kört på en Mac med Xcode version 6.1.1.

3. En klass `Polynomial` som beskriver polynom i en variabel används på följande sätt:

```
int main() {
    Polynomial p1;
    p1.add_term(-2, 1);
    p1.add_term(3.5, 0);
    p1.add_term(8.2, 4);
    cout << p1 << endl;      // 3.5 - 2x + 8.2x^4
    cout << p1(2.0) << endl; // compute value for x = 2.0, 130.7
    Polynomial p2;
    p2.add_term(10, 7);
    p2.add_term(1, 0);
    p2.add_term(1, 2);
    p2.add_term(-8.2, 4);
    cout << p2 << endl;      // 1 + x^2 - 8.2x^4 + 10x^7
    p1 += p2;
    cout << p1 << endl;      // 4.5 - 2x + x^2 + 10x^7
    cout << p1 + p2 << endl; //4.5 -2x + 2x^2 -8.2x^4 + 20x^7
}
```

   Man kan alltså:

   - skapa ett tomt polynom,
   - lägga till en term med koefficient och gradtal,
   - skriva ut ett polynom,
   - beräkna polynomets värde i en punkt,
   - addera polynom med `+=` och `+`,

   Implementera klassen enligt följande anvisningar:

   - Polynomet *ska* representeras av ett objekt av standardklassen `list` med element som innehåller koefficient och gradtal (skriv en egen klass för att representera dessa eller använd standardklassen `pair`) för de termer vars koefficient inte är noll. Listan *ska* vara sorterad efter växande gradtal.
   - I `add_term` kan du förutsätta att det inte redan finns en term med angivet gradtal.
   - Utskriften behöver inte göras så elegant som i exemplet. Det räcker med att koefficient och gradtal skrivs ut för varje term, med en term per rad.

4. Skriv ett program som läser in en textfil och räknar hur många *unika* ord som förekommer i filen. Ord som förekommer flera gånger i filen ska alltså bara räknas en gång.

   Anvisningar:

   - Inled programmet med att fråga användaren efter namnet på filen som innehåller texten.
   - Orden i filen är följder av bokstäver åtskilda av whitespace (blanktecken eller radslut). Inga andra tecken finns i filen.
   - För enkelhetens skull betraktas stora och små bokstäver som olika bokstäver. ”Hej” och ”hej” betraktas alltså som två olika ord.
   - Du kan förutsätta att filen existerar och att den kan läsas.

RAND(3) BSD Library Functions Manual RAND(3)

NAME

rand, rand_r, srand, sranddev -- bad random number generator

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <stdlib.h>

int
rand(void);

int
rand_r(unsigned *seed);

void
srand(unsigned seed);

void
sranddev(void);
```

DESCRIPTION

These interfaces are obsoleted by random(3).

The rand() function computes a sequence of pseudo-random integers in the range of 0 to RAND_MAX (as defined by the header file <stdlib.h>).

The srand() function sets its argument seed as the seed for a new sequence of pseudo-random numbers to be returned by rand(). These sequences are repeatable by calling srand() with the same seed value.

If no seed value is provided, the functions are automatically seeded with a value of 1.

The sranddev() function initializes a seed, using the random(4) random number device which returns good random numbers, suitable for cryptographic use.

The rand_r() function provides the same functionality as rand(). A pointer to the context value seed must be supplied by the caller.

SEE ALSO

random(3), random(4)

STANDARDS

The rand() and srand() functions conform to ISO/IEC 9899:1990 (‘‘ISO C90’’).

The rand_r() function is as proposed in the POSIX.4a Draft #6 document.