

DD2387 Programsystemkonstruktion med C++

Laboration 1: Grundläggande C++

31 augusti 2009

I den här labben kommer du att lära dig att använda grundläggande C++ såsom klasser, loopar, variabler och minneshantering. Ni får jobba i grupper om högst två personer. Vid återkopplingssamtalet ska alla gruppmedlemmar kunna svara på frågor om alla delar av koden.

Läs igenom hela lydelsen innan du börjar. För att uppgifterna ska kännas mindre lösryckta hänger de ofta ihop.

Allmänna krav på labben

- Din kod ska vara modulariserad i klasser och filer.
- Ditt program ska inte läcka minne, så var noga med dina konstruktörer och destruktörer.
- Ditt program ska visa att du behärskar `const` på ett korrekt sätt.
- I denna labb ska du inte använda dig av containerklasser i STL. Du får t.ex. inte använda STL-klassen *vector* för att implementera din vektorklass.
- Se till att dina program är indenterade (*M-x indent-buffer* i emacs).
- Använd fullständiga meningar när du svarar på frågeställningarna.

Förberedelser (Om du redan gjort följande behöver du inte göra det igen.)
Skriv namn på labbkvittot som kan hämtas på kurshemsidan. För att registrera dig på kursen ska du skriva följande:

```
datan> res checkin cprog09
```

För att få rätt labbmiljö skriv:

```
datan> course join cprog09
```

När labblydelsen hänvisar till filer i *kurskatalogen* avses

```
/info/cprog09/labbar/lab1
```

Redovisning När du är klar med labben ska du skicka in din lösning för en automatisk testning. Hur det går till förklaras i epost som skickas till din nada-adress (ditt login namn@nada.kth.se). Epostbrevet skickas strax efter första schemalagda labbtillfället - det är alltså mycket viktigt att du *registrerar dig på kursen innan dess*. Observera att den automatiska testningen inte är fullständig utan det krävs även redovisning för handledare.

Uppgifter

1.1 (iteration, pekare) Det finns ett klassiskt program som brukar kallas för Hello world. Denna uppgift går ut på att skriva ett program som skriver ut olika varianter på texten `Hello world!`. Programmet ska kunna ta en text och/eller ett tal som argument. Exempelskörning:

```
datan> hello
Hello world!
datan> hello C++
Hello C++!
datan> hello 3 C++          lurigt !
Hello C++ C++ C++!
datan> hello 2
Hello world world!
datan>
```

Denna uppgift är alltså inte en övning i objektorienterad programmering utan en introduktion till C++.

Tips: Använd `argv` och `argc`. Använd `atoi` för att översätta en `char *` till ett tal (om strängen inte representerar ett tal får man noll).

Vi ska använda kompilatorn `g++` för att översätta programkoden till ett körbart program. Lägg till `g++` genom att skriva `module add gcc/4.0.2`. Om du gjort `course join` kommandot ovan kommer nödvändiga moduler adderas automatiskt nästa gång du loggar in. För att kompilera ditt program skriver du följande:

```
> g++ -c myfile1.cpp
> g++ -c myfile2.cpp
> g++ -o myprog myfile1.o myfile2.o
```

Du kan också skriva

```
> g++ -o myprog myfile1.cpp myfile2.cpp
```

1.2 (avlusare, debugger) a På kurskatalogen finns ett program `matherrors.cpp`. Kompilera programmet med flaggan `-g` för att lägga till felsökningsinformation i din körbara fil.

```
> g++ -g -o matherrors matherrors.cpp
```

Starta avlusaren (debuggern) `DDD` med:

```
> ddd matherrors
```

Sätt ut brytpunkter (*breakpoints*) där du vill att programmet ska stanna under körning genom att högerklicka på en rad och välja *Set breakpoint*. Starta programmet genom att klicka på *Run* eller välja *Run...* under Program-menyn. När programmet stannar på din brytpunkt, dubbelklicka på valfri variabel i ditt program för att se dess värde. Alternativt kan du hålla muspekaren över en variabel för att se dess värde. Vilket värde får variabeln `w`?

Experimentera lite med avlusaren, prova att stega med *Next* och *Step*. Hur många varv körs for-loopen i funktionen `powerof`? Varför blir det fel värde?

b På kurskatalogen finns ett program `must_follow_a.cpp`. Funktionen `must_follow_a` tar ett intervall i en teckenvektor samt två tecken som indata. Funktionen returnerar antalet förekomster där det första tecknet följs av det andra inom intervallet. Kopiera, sätt dig in i koden och kompilera koden

```
g++ -c must_follow_a.cpp
```

På kurskatalogen finns ett program `test__must_follow_a.cpp` som kan testa koden med hjälp av testramverket `cxxtest`. Detta testramverk finns installerat på `/info/cprog09/cxxtest/`. Där finns också mer dokumentation om hur ramverket fungerar och var man kan hämta hem det.

För att bygga testet måste du generera en testfil (här kallad `1.2b.cpp`) med kommandot:

```
/info/cprog09/cxxtest/cxxtestgen.py --error-printer -o 1.2b.cpp test__must_follow_a.cpp
```

Kompilera därefter den genererade testfilen (glöm inte inkludera testbiblioteket).

```
g++ -o test_1.2b -I /info/cprog09/cxxtest/ 1.2b.cpp must_follow_a.o
```

Kör testet

```
./test_1.2b
```

Gör ett nytt test genom att ändra förutsättningarna så att funktionen `must_follow_a` förväntas returnera två. För att automatisera testgenereringen kan du använda `make`.

Funktionen är medvetet buggig och kan leta utanför det givna intervallet. Denna typen av fel där man räknat fel på ett brukar kallas “`off by one`”. Gör ännu ett nytt test där funktionen fallerar. Använd följande förutsättningar:

```
vek = {'b', 'b', 'a', 'b', 'b'};"
must_follow_a(vek, 3, 'a', 'b')
```

Om funktionen hade fungerat som det var tänkt borde man inte få någon teckenföljdsförekomst. Redovisa minst tre testfunktioner vid redovisningen.

Varför är det så viktigt att testa randvillkoren?

1.3 (Temporära objekt)

Fyll i egen påhittad text i de streckade strängarna nedanför. Komplettera med egna spårutskriftsriter i huvudprogrammet. Redovisa utskrifterna vid redovisning. Var beredd att svara på varför utskrifterna ser ut som de gör. När frigörs objekten? När skapas temporära objekt?

```
#include <iostream>

class A {
public:
    A()
        {std::cout << "_____ " << std::endl; }
    A(const A & ref)
        {std::cout << "_____ " << std::endl; }
    ~A()
        {std::cout << "_____ " << std::endl; }
    A(char * s)
        {std::cout << "_____ " << s << std::endl;}
    A & operator=(const A & s)
        {std::cout << "_____ " << std::endl;
         return *this;}
};

void no_ref(A a) {}
void with_ref(const A &a) {}

int main()
{
    A a("my name is a");
    A b = a;          // vad är skillnaden
    A c(a);           // mellan dessa
    A d;              // tre tekniker?
    d = a;

    A *aa = new A[5];
    delete aa;        // Vad kommer att hända?

    no_ref(a);         // Bildas temporära objekt?
    with_ref(a);       // Bildas temporära objekt?

    return 0;
}
```

1.4 (operatoröverlagring, minneshantering) Skapa en vektorklass `Vector` för positiva heltal (`unsigned int`). Du får inte använda klassen `vector` i STL i din lösning (däremot kan du, om du vill, implementera en referenslösning för att jämföra egna tester). Låt storleken vara fixerad och bestämmas av ett argument av typen (`size_t`) till konstruktorn. Även nollstora vektorer ska kunna skapas.

Varje vektorelement ska initieras till 0.

Implementera tilldelningsoperator och kopieringskonstruktor. Tilldelning/kopiering av olika stora vektorer ska fungera. Vektorklassen ska även överlagra indexoperatorn `[]` för snabb åtkomst av elementen.

```
...
int x = 2;
int i = vektor[7];
vektor[3] = x;      // OBS, ska fungera!
```

Kontrollera att det är giltig åtkomst annars ska `std::out_of_range` kastas! Generellt är det ofta en god idé att låta konstruktorer som tar ett argument deklaras som `explicit`, gör så även i denna klass. Varför? Ange ett exempel där det annars kan bli dumt. Prova din vektor med filen `test_vec.cpp` i kurskatalogen. Detta testprogram kontrollerar inte all funktionalitet. Du måste själv ansvara för att skriva ett bättre testprogram som testar randvillkoren ordentligt, t.ex. genom att använda ett testramverk som `cxxtest`.

Tips: Tänk på att operatorn `[]` måste vara en konstant medlemsfunktion i vissa fall. När och varför? Hur kopierar man vektorn?

```
Vector b = a;
a[0] = 1;      // b ska inte ändras av denna sats.
```

Tänk även på vad som händer i följande fall (dvs då vektorn förväntas kopiera sig själv):

```
Vector v; v = v;
```

1.5 (mallar) a) Modifiera din vektorklass `Vector` från uppgift 1.4 så att den kan lagra en godtycklig datatyp genom att använda mallar (*templates*). Din nya klass ska dessutom kunna ändra storlek efter den skapats. Klassen ska fortfarande kasta `std::out_of_range` vid ogiltig åtkomst. Exempel på instansiering:

```
class A;
Vector<double> dvect;
Vector<A *> apvect;
Vector<int> ivect(10);
```

Defaultkonstruktorn ska skapa en tom vektor. Om man anger en storlek till konstruktorn ska elementen initieras till defaultvärdet för typen (tilldela värdet `T()` till elementen). Implementera även en konstruktor som tar två argument, dels en storlek och ett defaultvärde för alla element i vektorn.

b) Du ska också lägga till ny funktionalitet i din klass:

- `push_back(T)` lägger till ett element sist i vektorn. Det ska för det mesta ske i konstant tid.
- `insert(size_t i, T)` lägger till ett element före plats `i`. Om `i` är lika med antal element i vektorn fungerar metoden som `push_back`
- `erase(size_t, i)` tar bort ett element på plats `i`
- `clear()` tar bort alla element

- `size()` ger antal element i vektorn
- `sort(bool ascending = true)` sorterar vektorn i angiven riktning på enklast möjliga sätt. (Observera att byte av två element kräver tilldelningsoperator och att alla datatyper som ska jämföras måste definiera `operator<.`)

Se till att rena åtkomstfunktioner (`read only`) är konstantdeklarerade.

Prova din nya vektorklass med filen `test_template_vec.cpp`. Detta testprogram kontrollerar inte all funktionalitet. Du måste själv ansvara för att skriva ett bättre testprogram som testar randvillkoren ordentligt.

Redovisning

När du är klar med de båda vektorklasserna ska du skicka in respektive källkod för testning. Instruktioner för hur du skickar in koden skickas till din nada-address. På kurskatalogen finns de två testprogram (`cprog09lab14.cpp`, `cprog09lab15.cpp`) som kommer att köras när du skickar in koden. Testprogrammen förutsätter att vektorerklasserna heter `Vector` respektive `template<typename T> Vector` och ligger i filer enligt nedan. Testprogrammet läser in instruktioner från en fil och utför dem på vektorerna sedan jämförs utdatat med utadata från en referensimplementation. Se därför till att inte skriva på `std::cout` i ditt program (det går att skriva på `std::cerr` istället men då kan din implementation underkännas för att den är för långsam).

För den första vektorklassen ska filerna `cprog09lab14.cpp`, `vector.h`, `vector.cpp` och en README-fil med personuppgifter och svar på frågorna. Det finns ett skelett till en sådan README fil på kurskatalogen. För den templetiserade vektorklassen ska all källkod ligga i `vector.h` och enbart den filen samt `cprog09lab15.cpp` ska skickas in. Testfilerna `cprog09lab14.cpp` och `cprog09lab15.cpp` ska skickas in omodifierade.

Endast inskickad och testad kod kan redovisas. Redovisningen sker vid schemalagda labbtillfällen. Om redovisningen går bra och källkoden skickades in innan bonusdatum (står på labbkvittot) får du tre bonuspoäng. Det är mycket vanligt att man får komplettera sin labb innan den blir godkänd. Se för din egen skull till att få en underskrift på labbkvittot av handledaren oavsett om labben ska kompletteras eller inte.

Lycka till!

Betygshöjande extrauppgifter

Extrauppgift 1.1 (4p) Skriv en dynamiskt allokerad matrisklass `Matrix` för heltal genom att använda `Vector<int>`. En matris är alltid rektangulär eller kvadratisk, det är inte tillåtet att kunna lägga till ett extra element på en rad eller kolumn. Klassen ska ha en del funktionalitet som förklaras nedan.

Åtkomst till elementen ska ges enligt följande exempel:

```
int x = matris[7][2];
matris[3][1] = x;
```

Låt matrisklassen definiera

```
std::ostream &operator<<(std::ostream &, const Matrix &)
```

så att man kan skriva ut matrisen med `cout` med rader och kolumner.

Definiera även en inmatningsoperator (operator») så att man kan mata in värden i en matris på matlab format [1 2 0; 2 5 -1; 4 10 -1] ingen felhantering behöver implementeras för felaktig inmatning. Första tecknet är alltid [

```
Matrix m;  
std::cin >> m;
```

Användaren matar in [1 2 -3 ; 5 6 7]

```
std::cout << m << std::endl;
```

Utskriften blir:

```
[ 1 2 -3  
; 5 6 7 ]
```

Skriv ett testprogram som visar all funktionalitet. Använd `cxxtest` ramverk. Om du även gör extrauppgift 1.3 ska du redovisa dessa tester tillsammans (t.ex. `m1[2][3]*m2`).

Extrauppgift 1.2 (5p) Använd `Matrix` för att implementera en labyrintlösare. Låt elementen i matrisen motsvara en ruta i labyrinten. Prova din labyrintlösare med filen `maze.cpp`. Skapa en funktion `read(const char **data)` som initierar matrisen med en labyrint. För den som vill ha större/andra labyrinter finns en labyrintgenerator `maze_generator.cpp` som genererar C++-syntax.

Extrauppgift 1.3 (6p, krav för betyg C) Utöka funktionaliteten så att man kan hantera aritmetik för matriser. Antingen kan du bygga ut din befintliga klass eller skriva en ny klass och använda dig av arv eller aggregat. Var beredd att försvara ditt designbeslut vid redovisningen.

Implementera tilldelningsoperator och kopieringskonstruktor samt identitet (sätter kvadratisk matris till identitetsmatrisen), negation och transponering. Överlagra operatorer för matrisaritmetik

`+`, `-` och `*` samt `*` för skalärmultiplikation.

Tips: Tänk över retur- och argumenttyper: vilka är `const` och vilka kan inte vara referenser? Vilka funktioner är `const`? Många av de föreslagna operatorerna delar funktionalitet. Utnyttja detta för att underlätta implementationen.

Skriv testfall för dina metoder. Skriv även testfall som inte ska fungera (matrisernas dimension är fel). Exempel på testfall, skalär- och matrismultiplikation av 0-stora, 1-stora matriser, kvadratiska, rektangulära matriser. Kedjeaddition och multiplikation av matriser `(m1*(m2+k*m3-m4)*m5)`. Om du även gör 1.1 ska du redovisa dessa tester tillsammans (t.ex. `m1[2][3]*m2`).

Använd `cxxtest` och redovisa dina tester vid redovisning.

Extrauppgift 1.4 (5p) Använd mallar för att implementera en klass `Hypercube` som hanterar liksidiga matriser med godtycklig dimension. Ta hjälp av `Matrix` eller `Vector` för implementationen. Exempel:

```

Hypercube<3> n(7);    // kub med 7*7*7 element
Hypercube<6> m(5);    // sex dimensioner, 5*5*...*5 element
m[1][3][2][1][4][0] = 7;

Hypercube<3> t(5);
t = m[1][3][2];      // tilldela med del av m
t[1][4][0] = 2;      // ändra t, ändra inte m
std::cout << m[1][3][2][1][4][0] << std::endl; // 7
std::cout << t[1][4][0] << std::endl;         // 2

```

När du har löst uppgiften, tänk efter hur du kan göra en elegant lösning på 10-15 rader. Redovisa helst en elegant lösning men en elegant tankegång kan också godkännas.

Extrauppgift 1.5 (6p) Implementera en specialisering `Vector<bool>` som använder så lite minne som möjligt, dvs representerar en `bool` med en bit. Använd någon stor heltalstyp (såsom `unsigned int`) för att spara bitarna. Observera att du ska kunna spara ett godtyckligt antal bitar. Skapa funktionalitet så att vektorn kan konverteras till och ifrån ett heltal (i mån av plats). Implementera all funktionalitet från `Vector` som t.ex. `size`. Du behöver inte implementera `insert` och `erase` om du inte vill.

Extrauppgift 1.6 (5p, krav för betyg A) Skapa en iteratorklass till `Vector<bool>` som uppfyller kraven för en random-access-iterator (dvs har pekarliknande beteende). Ärv från `std::iterator<...>` (genom `#include <iterator>`) för att lättare få rätt typdefinitioner. Låt din `iterator` ärva från din `const_iterator` eftersom den förra ska kunna konverteras till den senare. Läs på om `iterator_traits<T>` och definiera de typdefinitioner du behöver (kanske `typedef bool value_type`). Du behöver inte implementera `reverse_iterator` eller `const_reverse_iterator` om du inte vill. Exempel som ska fungera:

```

Vector<bool> v(31);    // Skapa en 31 stor vektor
v[3] = true;
Vector<bool> w;        // tom vektor
std::copy(v.begin(), v.end(), std::back_inserter(w));
std::cout << std::distance(v.begin(), v.end());
// konstant iterator och konvertering
Vector<bool>::const_iterator it = v.begin();
std::advance(it, 2);

```

Det kan vara mycket svårt att få till så att `sort(v.begin(), v.end())` fungerar. Det krävs inte men du bör kunna resonera om vad som saknas i din lösning.

Extrauppgift 1.7 (3p) Låt `Vector<bool>` överlagra operatorer för booleska operationer: `&`, `|` och `^`. Använd datorns hårdvara i största möjliga utsträckning genom att använda booleska operationer på `unsigned int`.

Skapa även minst tre funktioner `weight` som räknar antalet satta bitar (ettor). Använd de booleska operationerna för detta, d.v.s. gör beräkningen utan att titta på varje bit separat. Ett sätt att göra det är att mappa 256 bitmönster i en array och helt enkelt slå upp antalet bitar i varje byte. Ett annat sätt är att räkna ettor och ta bort den högraste ettan i varje iteration. Ytterligare en

variant är att invertera alla ettor och nollor innan man räknar högraste ettan. En mer matematisk variant är följande tvåradare som utan slinga räknar ettor i ett 32-bitars tal.

```
xCount = x - ((x >> 1) & 033333333333) - ((x >> 2) & 011111111111);  
return ((xCount + (xCount >> 3)) & 030707070707) % 63;
```

Redovisa några tester där respektive variant fungerar bäst. Tänk också på vad som händer om vektorerna har olika storlek.