

2D1387 Programsystemkonstruktion med C++  
Redovisning av laborationer 2012/13

Hicham MOHAMAD  
Namn

720101-3278  
Personnummer

Labb 1 - Grundläggande C++

hamednomati 2013/11/28  
Handledare Datum (ev. komplettering handl/datum)

Labb 2 - Kalender

HD 2013-02-01  
Handledare Datum (ev. komplettering handl/datum)

Projektuppgift - Äventyrsspel

Alexander 3/12-13 flytta spel-log från konstruktor.  
Handledare Datum (ev. komplettering handl/datum)

Ex1.1 Daniel John 2014-01-30

Ex1.2 Chij 31-01-2014

Ex1.3 \_\_\_\_\_

Ex1.4 Chij 31-01-2014

Ex1.5 \_\_\_\_\_

Ex1.6 \_\_\_\_\_

Ex1.7 \_\_\_\_\_

Ex2.1 MStahre 31/1-14

Ex2.2 \_\_\_\_\_

Ex2.3 \_\_\_\_\_

Ex2.4 \_\_\_\_\_

Ex3.1 Alexander 2014-01-14

Ex3.2 Alexander 2014-01-14

Ex3.3 Han 11/11

Ex3.4 Han 11/1

Ex3.5 \_\_\_\_\_

Ex3.6 \_\_\_\_\_

B → 45 - 59

C → 30 - 44

jag har tot → 24 p lab 3 (fid + extra)

+ 19 p Lab 1 extra

43 p

+ 5 p Lab 2 - extra

48 p

# DD2387 Programsystemkonstruktion med C++

## Laboration 1: Grundläggande C++

23 augusti 2012

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 återkopplingssamtalen ska alla gruppmedlemmar kunna svara på frågor om alla delar av koden.

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

### Allmäna 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 labbkвиттот som kan hämtas på kurshemsidan. Be om namnunderskrift efter varje redovisad labb.

När labblydelsen hänvisar till filer i **kurskatalogen**, avses

*DD2387/kurskatalog/Lab1  
/info/cprog12/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@csc.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.

På kurskatalogen finns en **README** fil med de frågor som ställs i labblydelsen. Fyll i svaren på frågorna och bifoga filen med någon av dina inskickningar, håll reda på vilken så att du kan ta fram den vid redovisning.

### Uppgifter

**1.1** (**iteration, pekare, make**) 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. Exempelkörning:

— will need compiler

① datan> hello.out  
Hello world!  
② datan> hello.C++  
Hello C++!  
③ datan> hello 3 C++      *lurigt!*  
Hello C++ C++ C++!  
④ datan> hello 2  
Hello world world!  
datan>

Denna uppgift är alltså inte en övning i objektorienterad programmering utan en introduktion till C/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 kommer använda den kompilator g++ som finns intallerad på salsdatorerna. För att kompilera din källkodsfil nedan kallad myfile.cpp skriv:

> g++ -o myprog.out myfile.cpp

Filten myprog.out kan du köra i terminal fönstret genom att skriva:

> ./myprog.out

Ofta används programmet make för att kompilera större projekt, make utgår från att det finns en fil **makefile** med regler för vad som ska göras. Kopiera makefile från kurskatalogen till den katalog du står på () och titta på filen.

> cp /info/cprog12/lab1/makefile .  
> more makefile  
.out: %.cpp  
      g++ -g -std=c++0x -Wall \$\*.cpp -o \$\*.out

Nu kan du skriva

> make myfile.out

*to turn on  
C++11 support*

Det är nyttigt att lära sig hur make fungerar (även de som använder Visual Studio) googla efter valfri kort tutorial och svara på vad andra raden gör, vad betyder \$\*? Googla även på g++ och ta reda på vad -Wall och -g gör.

### 1.2 (avlusare, debugger)

På kurskatalogen finns ett program **matherrors.cpp**.

```
int powerof(int x, int y) {  
    int res = 1;  
    for (int i = 0; i < y; i++) {  
        res *= x;  
    }  
    return res;  
}  
  
int main() {  
    int x = 10;
```

-c compile but not link  
-Wall → turn on warnings  
-o → output file  
-g → insert debugging info in your executable

```

int y = 3;
int res = powerof(x, y);
std::cout << x << " upphöjt till " << y << " är " << res << std::endl;
float z = 0.29;
int w = (int) (z * x * x);
if (z * x * x == 29)
    std::cout << z << "*" << x * x << " är 29" << std::endl;
else
    std::cout << z << "*" << x * x << " är inte 29" << std::endl;
}

```

Kompilera programmet med flaggan -g för att lägga till **felsökningsinformation** i din körbara fil.

```

> cp /info/cprog12/lab1/matherrors.cpp .
> make matherrors.out

```

Starta **avlusaren** (debuggern) DDD med:

```
> ddd matherrors.out
```

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. Experimentera lite med avlusaren, prova att stega med *Next* och *Step*.

— Varför blir värdet på variabeln w inte det man tror? *float = 6 significant digits → double = 10 significant digits*

— Hur många varv körs för-loopen i funktionen **powerof**? *ett varv*

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 filerna, sätt dig in i koden.

```

int must_follow_a(char * start, int length, char a, char b) {
    int nr = 0;
    for (int i = 0; i < length; i++, ++start) {
        if (*start == a && *(start + 1) == b) // maintainers note: DANGER!
            nr += 1;
    }
    return nr;
}

```

\* Det finns ingen main-funktionen, komplilera koden med  
 $\sim \text{std} = \text{C}++\text{0x}$   
 $\text{g}++ -c \text{must\_follow\_a.cpp}$

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/cprog12/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/DD2387/cprog11/labbor/3/Lab1`

→ `cp README.lab1 /afs/nada.kth.se/home/e/mwojcie/c++prog/labbor/lab1`  
 viu README.lab1

charlotte.no-ip.com:81/progsseminare/2011-12-26\_GCCv4.7

How to install and use GCC g++ v4.7 and C++11 on Ubuntu 12.04  
(stéphane charlotte)

GNU eu20523

vara i denna kopia  
för kurskatalog!

/info/cprog12/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.out -I /info/cprog12/cxxtest/ 1.2b.cpp must\_follow\_a.o

Kör testet

> ./test\_1.2b.out

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 genom att lägga till en ny regel i din makefile.

→ 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')

, OBOE  
Boundary  
condition

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

Varför är det så viktigt att testa randvillkoren?

→ Boundary condition

Valgrind is a memory management detector.  
It shows you memory leaks, deallocation errors etc.

### 1.3 (Temporära objekt, minnesläckor, valgrind)

Kopiera programmet A.cpp från kurskatalogen och komplettera med egna spårutskrifter i huvudprogrammet så att du förstår vad som händer. Redovisa utskrifterna vid redovisning. Var beredd att svara på varför utskrifterna ser ut som de gör. När frigörs objekten? När skapas **temporära objekt**?  
*varför utskrifterna  
ser ut som de gör?*

Programmet **valgrind** används ofta för att analysera program skrivna i C/C++. Prova det på A.out.

```
> valgrind --tool=memcheck --leak-check=yes ./A.out
```

Kopiera programmet Data.cpp från kurskatalogen.

```
Data ** foo(Data ** v, int x) {
    for (int i = 0; i < x; i++)
        → if (v[i] != 0)
            v[i] = new Data;
    return v;
}
```

Kompilera och kör valgrind på utfilen.

```
> cp /info/cprog12/lab1/Data.cpp .
> make Data.out
> valgrind --tool=memcheck --leak-check=yes ./Data.out
```

— Notera att valgrind klagar på att programmets beteende beror på en **oinitierad variabel**. Hur ser valgrinds felmeddelande ut? Kommentera bort den raden (if-satsen), blir det någon skillnad i hur mycket minne som läcker? Borde det ha blivit någon skillnad?

Ändra på sista raden till

*Data \*\* p is equivalent to Data \* p[]*

*it is an array of Data pointers*

```
Data ** p = foo(v, size);
delete [] p; // deallocate only the pointer to the vector (array)
            but nothing happens to objects Data
```

— Varför läcker det fortfarande minne?

**1.4 (operatoröverlaging, 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 **kopieringskonstruktur**. Tilldelning/kopiering av olika stora vektorer ska fungera. Implementera även **move-konstruktur** och **move-operator** så att **std::move** fungerar. Implementera en **tillfältningsoperator** som tar en **initializer\_list** som parameter.

Vektorklassen ska även **överlagra** indexoperatorn [] för snabb åtkomst av elementen.

*referenslösning?*

```
...  
int x = 2;  
int i = vektor[7];  
vektor[3] = x; // OBS, ska fungera!  
Vector v2;  
v2 = {1, 2, 5};
```

Kontrollera att det är giltig åtkomst annars ska `std::out_of_range` kastas! Generellt är det ofta en god idé att låta konstruktörer som tar ett argument deklareras 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. Använd valgrind. 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:

*Expanding a vector:*

```
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 konstruktur 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:

- 1 • `push_back(T)` lägger till ett element sist i vektorn. Det ska för det mesta ske i konstant tid.  $O(1) \Rightarrow$  la recherche se fait en temps constant.
- 2 • `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`
- 3 • `erase(size_t i)` tar bort ett element på plats `i`
- 4 • `clear()` tar bort alla element
- 5 • `size()` ger antal element i vektorn

Because constructor that can be called with single argument defines an implicit conversion from the constructor's parameter type to the class type.

Template <typename T> Vector

- 6 • `sort(bool ascending = true)` sorterar vektor i angiven riktning på enklast möjliga sätt. Använd `std::sort` (datatyper som ska jämföras måste definiera `operator<`.)
- 7 • `exists(const T &)` Returnerar true om elementet finns i vektorn annars false. Använd `std::find` för att implementera funktionen.

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. Ett **minimum av test** är att skriva ett test för varje medlemsfunktion man implementerar. Glöm inte använda `valgrind`.

## Redovisning

När du är klar med de båda vektorklasserna ska du skicka in respektive källkod för testning. Instruktioner för hur du skickar in koden manuellt finns på kurs-hemsidan. Det finns ett inskickningsskript för att automatisera inskickandet se hjälppavsnitt på [kth.kattis.scrool.se](http://kth.kattis.scrool.se). Skicka in det **testprogram** (`cprog12lab14.cpp`, `cprog12lab15.cpp`) som kommer att köras när du skickar in koden. De finns på kurskatalogen. Testprogrammen förutsätter att vektorklasserna 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 utdata med utdata 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 blir för långsam).

För den första vektorklassen ska filerna `cprog12lab14.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 `cprog12lab15.cpp` ska skickas in. Testfilerna `cprog12lab14.cpp` och `cprog12lab15.cpp` ska skickas in ommodifierade.

Endast inskickad och testad kod kan redovisas. Redovisningen sker vid schemalagda labbtillfällen. Om redovisningen går bra och källkoden skickades in innan bonusdatum (står på labkvittot) får du två 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 labbkвиттот av handledaren oavsett om labben ska kompletteras eller inte.

Lycka till!

## Betygshöjande extrauppgifter

**Extrauppgift 1.1 (10p, krav för betyg C)** Skriv en dynamiskt allokerad matrisklass `Matrix`. På kursbiblioteket finns en header-fil `Matrix.h` som du ska utgå ifrån. Observera att det finns ett eller fler **medvetna designfel** i headerfilen. `Matrix.h` ska använda din vektorimplementation i `lab1`. För att man inte ska kunna lägga till extra element på en rad eller kolumn används internt en klass `matrix_row`.

```
matrix[7].push_back(1); // Ej tillåtet
```

Matrikklassen ska ha en del funktionalitet som förklaras nedan.

Åtkomst till elementen ska ges enligt följande exempel:

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

Låt matrikklassen 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 visas nedan till vänster. Till höger visas samma utskrift men med understrykningstecken istället för mellanslag (notera mellanslag sist på raden). Testa utskriften med `cxx_test` genom att skriva till en `strängström`.

<code>[ 1 2 -3 ; 5 6 7 ]</code>	<code>[_1_2_-3_ ;_5_6_7_]</code>
-------------------------------------	--------------------------------------

Implementera **aritmetik** för matriser. Implementera **tilldelningsoperator** och **kopieringskonstruktör** samt **identitet** (sätter kvadratisk matris till identitetsmatrisen), **negation** och **transponering**. Överlägra 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`? En del av operatorerna delar funktionalitet. Utnyttja detta för att underlättा implementationen.

Skriv testfall för dina metoder. Skriv även testfall som inte ska fungera (matrissernas dimension är fel). Exempel på testfall, skalär- och matrismultiplikation av 0-stora, 1-stora matriser, kvadratiska, rektangulära matriser. **Kedjeaddition** och multiplikation av matriser. Vad finns det för designfel i `Matrix.h`? Vad borde man kunna göra som man inte kan göra?

→ funzione  
composta  
f o g

På kursbiblioteket finns nio felaktiga matrisimplementationer. De ligger i underbibliotek kompilerade för ubuntu. Matriserna är kompilerade med `std::vector` och `Matrix.h` är ändrad därefter.

Skriv testfall som fångar felet i varje matris. Testfall 5 och 9 är tillståndsfel efter `utskrift` och `tilldelning`, och kan vara svåra att träffa. Samla flera testfallsanrop i en testfunktion och anropa den efter `utskrift/tilldelning`. Testa även dina testfall på din matrisimplementation (som bygger på din egen vektor). Din testkod är oberoende av vektor men testkoden måste kompileras om med de felaktiga matriserna.

→ För att bygga med en buggig matrisimplementation finns en färdig `Makefile` på kurskatalogen. Kopiera hela katalogen. Ändra eventuellt i sökvägen till `cxxtest` i Makefile. Bygg första testet med `make runtest01`. Vid redovisningen ska du kunna redogöra för alla delar i din kod, även den kod du inte skrivit (t.ex. `matrix_row`). Du ska kunna svara på hur `Matrix.h` kan förbättras och ha gissningar på vad som är galet i de olika matrisimplementationerna du testat.

**Extrauppgift 1.2 (4p)** Använd `Matrix` för att implementera en `labyrintlösare`. Låt elementen i matrisen motsvara en ruta i labyrinten. Skriv ut den slutgiltiga lösningen (utan återvändsgränder). 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 Xp)**  
Reserverad för framtidens.

**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 vektor 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 definera 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>` överlägra 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. Ytterliggare 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.

- <http://cs.yale.edu/classes/>
- <http://idt.snu.ac.kr/lectures> *seoul national university*
- [www.tondering.dk/claus/calendar.html](http://www.tondering.dk/claus/calendar.html)
- [www.hermesic.ch](http://www.hermesic.ch)
- [gl.developpez.com/tutoriel/c-util/makefile](http://gl.developpez.com/tutoriel/c-util/makefile)
- [net.pku.edu.cn/course/cs101/2011/resource/note-cpp](http://net.pku.edu.cn/course/cs101/2011/resource/note-cpp)

## 2D1387 Programsystemkonstruktion med C++

### Laboration 2: Kalender (*chapter 15 in Lippman*)

1 september 2011

I den här labben ska du visa att du att lärt dig att använda **överlagring, abstrakta basklasser** och **polymorf**. Ni får jobba i grupper om högst två personer. Vid redovisningen ska alla gruppmedlemmar kunna svara på frågor om alla delar av koden. Läs igenom hela lydelsen innan du börjar. För att uppgifterna ska känna mindre lösvryckta hänger de ofta ihop.

Allmänna krav:

- Ditt program ska visa att du behärskar nyckelord som **const** och **virtual** på ett korrekt sätt.
- 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 får använda STL.
- Innan du redovisar ska du läsa på frågeställningarna.
- Ni ska kunna redogöra för fyra olika klassdiagram och visa hur ni har använt er av polymorfi.
- Ni ska skicka in följande .h och .cpp filer till Kattis. *date, julian, gregorian, calendar, kattistime*. Dessutom vektorn och testprogrammet (<lab-namn>.cpp)
- Vid redovisning ska ni ha en webläsare öppen med den källkod ni skickat in till Kattis.

När labblydelsen hänvisar till filer i *kurskatalogen* avses

`/info/cprog11/labbar/lab2` → oldx2Hjs

Lycka till!

#### 2.1 (abstrakta basklasser, strikt virtuella funktioner)

Denna uppgift är den första av flera där du ska skapa datum- och kalenderklasser. Du ska häданefter samla allt som har med din kalender att göra i namnrymden (**namespace**) `lab2`.

Med en **abstrakt basklass** definierar man upp ett gränssnitt som nedärvt klasser måste uppfylla. Skriv en generell **basklass Date** som har följande funktioner. Om inget annat sägs är returtypen **int**.

- Avgör vilka konstruktörer som ska vara med. Behövs tilldelningsoperator?

- Åtkomst: `year()`, `month()` (första månaden ska representeras av talet 1), `day()` (den första i varje månad ska representeras av talet 1), `week_day()` (måndag = 1), `days_per_week()` (returnerar ett fixt antal dagar), `days_this_month()` (returnerar antal dagar den aktuella månaden) och `months_per_year()`. Skapa funktioner `week_day_name()` och `month_name()` med returtyp `std::string`.

⑨ • Mutatorer:

- prefix-operatorer `++` och `--` som returnerar referens till sig själv.
- `+=`, `-=` tar `int` som parameter. Välj lämplig returtyp.
- `add_year(int n = 1)` och `add_month(int n = 1)` (lägger till `n` år/månad, där `n` är positiv eller negativ). Välj lämplig returtyp och motivera vid redovisning.

- **Obs!** Basklassen ska kunna hantera alla sorters kalendersystem (persiska, kinesiska m.m.). Basklassen ska därför inte innehålla något som är specifikt för den kalender vi använder, såsom 12 månader per år, veckodagarnas namn eller dyl.

- Jämförelser (returnerar `bool`): `==`, `!=`, `<`, `<=`, `>`, `>=`, `-` (returnerar heltalsdifferensen mellan två datum)

- För att kunna jämföra olika datum som t.ex. ett persiskt datum med ett svenska datum, så ska du definiera en metod `mod Julian_day` som returnerar antal dagar sedan det modifierade julianska dygnet vilket i vår nuvarande kalender är den 17 november 1858. Mer information om just detta dygn kan fås via google, eller wikipedia, sök på "modified Julian day number".

- Man ska kunna skriva ut implementationer av datumklassen med utskriftsoperatorn. Deklarera `ostream & operator<<(ostream & os, const Date &);`. Låt utskriften vara år-månad-dag (med bindestreck).
- Operatorerna ska vara virtuella där så behövs. Låt de funktioner som inte kan definieras i basklassen vara strikt virtuella (pure virtual).

- Vilka funktioner är `const`? Vilka argument/returtyper är `const`? Vid redovisningen ska du övertyga handledaren att du kan använda `const` och `virtual`.

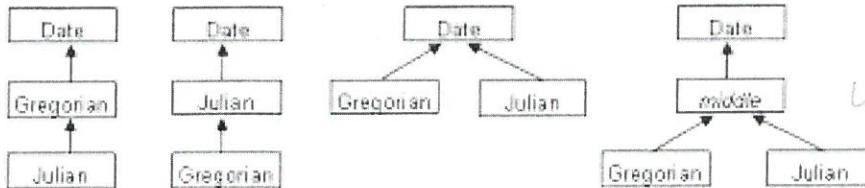
Kuriosa: Excel använder 1:a januari 1900 som referensdag. Om man öppnar Excel på PC, formaterar en cell till datum och matar in 60 så får man 29/2 1900. Den skottdagen har dock aldrig funnits. Jag hyser stor tillsikt att ni klarar er bättre än excelprogrammerarna.

## 2.2 (arv från abstrakt basklass)

**Gör följande:** skriv två klasser Julian och Gregorian för den julianska respektive gregorianska kalendern. Dessa ska ärvda (direkt eller indirekt) från klassen

- Julian until 1580
- Gregorian is the one commonly used today Feb 1582

Date i uppgiften ovan. Ta **ställning** till nedanstående fyra alternativ att ärva. Vid redovisningen ska du reflektera över för- och nackdelar för alla de fyra alternativen.



#### Funktionalitet hos klasserna:

- Implementation av basklassens virtuella funktioner vid behov
- Konstruktör som tar år, månad, dag som int.
- Man ska kunna kopiera och tilldela dateobjekt. Skillnaden mellan datumen ska vara noll efter tilldelning/kopiering.
- postfix-operatorer ++ och -- som returnerar kopia av sig själv.
- Konstruktion utan argument (**defaultkonstruktorn**) ger dagens datum. För att få dagens datum ska du inte använda systemanrop direkt utan gå via en speciell funktion, se filen **kattistime.h** på kurskatalogen.
- Korrekt hantering av skottår. I den julianska är det vart fjärde år. I den Gregorianska kalendern infaller skottdagen vart fjärde år men utgår de sekelårtal som inte är jämnt delbara med 400.
- Utskriftsoperatorn ska definieras och skriva ut datumet på formen YYYY-MM-DD där ental fylls ut med nollar (2000-01-01).
- **add\_month** ska ge samma datum i nästa månad men om det inte går så ska det plussas på 30 dagar istället. Exempel 1/9 -> 1/10 och 31/5 -> 30/6. Den 31/1 kan bli 1:a eller 2:a mars beroende på om det är skottår.
- För skottdagen ska **add\_year** ge sista februari: 29/2 2004 -> 28/2 2005. Anropar man **add\_year** den 29/2 först med 1 och därefter med -1 så ska man hamna på 28/2. Lägger man till fyra år till skottdagen ska man hamna på skottdagen igen om det nu är skottår då: 29/2 2004 -> 29/2 2008. Hade man istället lagt till ett år i taget hade man hamnat på den 28:e
- Anropar man **add\_month** med argumentet 5 så ska det vara ekivalent med om man gör **add\_month** 5 gånger. Motsvarande gäller inte alltid för **add\_year** se ovan.
- Försöker man komma åt eller skapa ogläliga dagar så ska **out\_of\_range** slängas.
- Använd engelska namn med små bokstäver på dagar och månader. Var noga med stavningen.
- **Deklaration** och **implementation** av klasserna ska skrivas i i separata header-respektive implementationsfiler (.h .cpp).

Alla implementationsdetaljer exempelvis om du har en hjälpfunktion `leap_year()` ska döljas med `protected` eller `private`.

För att underlätta räkningar kan du låta datumen representeras av ett avstånd till ett fixt datum. Du behöver inte implementera en sluten formel för datumberäkning utan det är tillåtet att använda en hårdkodad (förberäknad) tabell som t.ex. innehåller avstånd till årets början. Det räcker om din kalender klarar datum från och med år 1858 fram till och med år 2558.

Prova dina klasser med filen `datetest.cpp`. Utöka testprogrammet med egna tester. T.ex.

```
Gregorian g;          // dagens datum
Julian j;             // också dagens datum
std::cout << "Today it is " << g << " gregorian and " << j << " julian";
if (g - j == 0) std::cout << ". It is the same date" << std::endl;
g = j;
if (g - j == 0) std::cout << "It is still the same date" << std::endl;
```

Vilket ger utskriften

```
Today it is 2006-08-01 gregorian and 2006-07-19 julian. It is the same date
It is still the same date
```

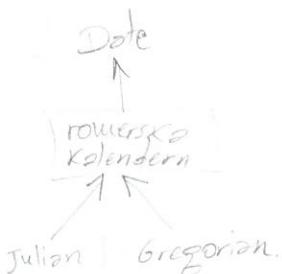
Dagens datum är alltså inte samma i kalendrarna. Följande gäller:

1/1 1900 Greg	= 20/12 1899 Jul
1/1 1900 Jul	= 13/1 1900 Greg
16/11 1858 Greg	= 4/11 1858 Jul modified julian day -1
17/11 1858 Greg	= 5/11 1858 Jul modified julian day 0
18/11 1858 Greg	= 6/11 1858 Jul modified julian day 1

Det finns flera **datumkonverterare** mellan gregorianska och julianska kalendern på nätet om man vill kontrollera fler datumskillnader.

**Historisk bakgrund** Den Julianska kalendern infördes av Julius Caesar år 46 f.Kr. Detta år infogade Julius Caesar hela 90 dagar till den **romerska kalendern** för att återföra månaderna till deras rätta ställe på året med hänsyn till årstiderna. I den **Julianska kalendern** infinner sig skottår vart fjärde år.

På grund av tillkortakommanden i skottårsberäkningen i den Julianska kalendern hade påskens vid tiden för påven Gregorius XIII förskjutits tio dagar (förskjutningen var cirka 3/4 dag per århundrade). Gregorius ansåg att det var dags att räta upp detta och proklamerade att torsdagen den 4:e oktober 1582 skulle följas av fredagen den 15:e oktober. Det var bara katolska länder som följde påbudet. I Sverige bestämdes det att man skulle införa **gregorianska kalendern** succesivt mellan år 1700 och 1740 genom att ta bort skottdagarna var fjärde år vilket inte fungerade så bra i praktiken. Bland annat glömde man bort att man inte skulle ha skottår 1708.



with map < > can  
be faster !?

## EventDate

### 2.3 (polymorfi, pekare till basklass, strömmar)

I den här uppgiften kommer du att skriva en klass **Calendar** som:

- internt håller en lista med händelser (såsom julafton, födelsedag) kopplade till datum
- har en defaultkonstruktor som sätter aktuellt datum till nuvarande datum.
- kan tilldelas en annan kalender (som kan vara instansierad med en annan datumtyp)
- har en metod **bool set\_date(int år, månad, dag)** som sätter om aktuellt datum. Om parametrarna bildar ett ogiltigt datum ska **false** returneras.
- har metoden **bool add\_event**, som kan ta en till fyra parametrar: händelse (string) och parametrarna dag, månad, år (int). Om någon av de senare parametrarna saknas så används aktuellt datum. Om datumet är ogiltigt ska **false** returneras. Om händelsen (strängen) redan finns i kalendern på det angivna datumet ska **false** returneras och ingenting nytt läggas in.
- **remove\_event** med samma parametrar som **add\_event**. Om det inte går att ta bort händelsen returneras **false**.

Använd STL för att hantera dina händelser. Använd **std::string** till strängarna som representerar händelser. En dålig lösning är att ha datum och händelse i två parallella vektorer med **gemensamt index**. Varför? Förklara dig vid redovisningstillfället.

Låt din kalenderklass vara en **mallklass** som tar en klass nedärvid från **Date** som argument. När du använder datum ska du endast använda dig av funktioner som ligger i basklassen **Date**. På så sätt håller du kalenderklassen datumberoende. Några exempel på hur din klass ska bete sig:

```
std::cout << "-----" << std::endl;
Calendar<Gregorian> cal;
cal.set_date(2000, 12, 2);
cal.add_event("Basketträning", 4, 12, 2000);
cal.add_event("Basketträning", 11, 12, 2000);
cal.add_event("Nyårsfrukost", 1, 1, 2001);
cal.add_event("Första advent", 1);           // år = 2000, månad = 12
cal.add_event("Vårdagjämning", 20, 3);        // år = 2000
cal.add_event("Julafton", 24, 12);
cal.add_event("Kalle Anka hälsar god jul", 24); // också på julafton
cal.add_event("Julafton", 24); // En likadan händelse samma datum ska
                             // ignoreras och inte sättas in i kalendern
cal.add_event("Min första cykel", 20, 12, 2000);
cal.remove_event("Basketträning", 4);

std::cout << cal; // OBS! Vårdagjämning och första advent är
                  // före nuvarande datum och skrivs inte ut
std::cout << "-----" << std::endl;
cal.remove_event("Vårdagjämning", 20, 3, 2000);
cal.remove_event("Kalle Anka hälsar god jul", 24, 12, 2000);
```

```

cal.set_date(2000, 11, 2);
if (! cal.remove_event("Julafton", 24)) {
    std::cout << " cal.remove_event(\"Julafton\", 24) tar inte" << std::endl
    << " bort något eftersom aktuell månad är november" << std::endl;
}
std::cout << "-----" << std::endl;
std::cout << cal;

```

Man ska kunna skriva ut kalendern m.hj.a utskriftsoperatorn. Enbart händelser efter aktuellt datum ska skrivas ut. Kalenderns händelser ska vara sorterade i första hand på datum och i andra hand på **insättningsordning**. Formatet ska vara på formen **datum : händelse** med etterföljande retur, se nedan. OBS! nollar framför ental (enligt Date) samt mellanslag före och efter kolon

```

-----
2000-12-11 : Basketträning
2000-12-20 : Min första cykel
2000-12-24 : Julafton
2000-12-24 : Kalle Anka hälsar god jul
2001-01-01 : Nyårsfrukost
-----
cal.remove_event("Julafton", 24) tar inte
bort något eftersom aktuell månad är november
-----
2000-12-01 : Första advent
2000-12-11 : Basketträning
2000-12-20 : Min första cykel
2000-12-24 : Julafton
2001-01-01 : Nyårsfrukost

```

Testa din kod innan du skickar in den. Endast inskickad och testad kod kan redovisas. Redovisningen sker vid schemalagda labbtillfällen. Om redovisningen går bra och källkoden skickades in innan bonusdatum (står på labkvittot) får du två 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 labkvittot av handledaren oavsett om labben ska kompletteras eller inte.

### Betygshöjande extrauppgifter

*Lap2\_sp\_ex22  
kalendrar.h  
kalendrar.cpp*

Extrauppgift 2.1 (5p) Lägg till följande funktionalitet hos kalendern:

- Skriv metoden `bool move_event(const Date & from, const Date & to, std::string event)` som kan flytta händelser i kalendern genom att först plocka ut och sedan lägga till händelsen på det nya datumet. Om händelsen inte finns i `from` alternativt redan finns i `to` returneras false.
- Skriv metoden `bool add_related_event(const Date & rel_date, int days, std::string rel_event, std::string new_event)`. En händelse ska kunna relateras till en annan händelse genom att man anger det antal dagar som skiljer dem åt. Flyttas händelsen kommer den relativa händelsen att flyttas också. Tas den bort så tas den relativa händelsen också bort.

- recurring events*
- Ge möjlighet att lägga in återkommande händelser, t.ex. att julafaston sker 24:e december varje år och fäktnings är det varje fredag. Man ska kunna ange hur länge händelsen ska återkomma annars är defaultvärdet hundra.
  - Ge möjlighet att lägga till födelsedagar så att kalendern räknar ut hur gamla födelsedagsbarnen är. I vår gregorianska kalender gäller att födelsedagar 29/2 fyller år 28/2 nästa år.
  - Skriv ett demonstrationsprogram som visar den nya funktionaliteten. Skriv ut både instruktioner och utskrifter så att det går att följa utan att behöva slå upp i demonstrationsprogrammet vad som händer.

**Extrauppgift 2.2 (6p)** Implementera ytterliggare två utmatningsformat för din kalender. För att ändra format på utskriften använder man `cal.set_format(Calendar::format)`, där `Calendar::format` är en av tre `enumtyper`, `list`, `cal` resv och `iCalendar` definierade i `Calendar`.

Det ena formatet liknar unix-programmet `cal`. December 2007 ska se ut så här med några speciella datum inlagda och aktuellt datum satt till 2/12.

december 2007						
må	ti	on	to	fr	lö	sö
					1	< 2>
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20*	21	22	23
24*	25	26	27	28	29	30
31						

2007-12-20: Min andra cykel  
2007-12-24: Julafaston

Det andra formatet är `iCalendar` (rfc2445) formatet som kan läsas in till andra kalenderprogram. Nedan är ett exempel. Ändra PRODID till någon unik text. Varje händelse är ett VEVENT med BEGIN/END. DSTART och DTEND innehåller datum och tider (efter T:et), sätt dem som entimmespass med början från klockan 8. SUMMARY innehåller en textbeskrivning över händelsen.

first event

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Simsalabim AB//En bra kalender made by ...//
```

second event

```
BEGIN:VEVENT
DTSTART:20071220T080000
DTEND:20071220T090000
SUMMARY:Min andra cykel
END:VEVENT
BEGIN:VEVENT
DTSTART:20071224T080000
DTEND:20071224T090000
SUMMARY:Julafaston
END:VEVENT
END:VCALENDAR
```

Redovisa genom att låta ett grafiskt kalenderprogram (t.ex. sunbird, google calendar m.m.) läsa in din kalenders ical-formaterade utskrift och visa upp vid redovisningen.

**Extrauppgift 2.3 (6p)** Implementera ett gränssnitt till din kalender. Låt kalenderns funktioner representeras av ett menysystem med flera menyer (menyn kan vara under kalendern):

Werk interface  
user\_interface.cpp

december 2007							Huvud meny:
må	ti	on	to	fr	lö	sö	-----
					1 < 2>		1. Välj datum
3	4	5	6	7	8	9	2. Välj månad
10	11	12	13	14	15	16	3. Formatmeny
17	18	19	20*	21	22	23	4. Händelsemeny
24*	25	26	27	28	29	30	q. Avsluta
31							
2000-12-20: Min första cykel							Ditt val: _
2000-12-24: Julafhton							

Låt undermenyer och faktiska menyval dela samma (abstrakta) basklass **Action**. Låt undermenyklassen ha en medlem **add** som lägger till ett objekt av typ **Action**. Använd **mallar och funktionspekare** för att få dina menyval att samarbeta med kalendern. Tänk på att olika menyval ger upphov till funktioner med olika många argument och med olika argumenttyper. Denna uppgift ska lösas utan upprepning av kod genom att utnyttja mallar och klasser. Man ska enkelt kunna lägga till eller flytta menyval.

Så här skulle det kunna se ut men stirra dig inte blind på just den här lösningsvarianten.

```
Menu::Output new_meeting("add a new meeting");
Menu::Output change_meeting("change meeting time");
Menu::Output evaluate_meeting("evaluate meeting");
...
Menu::Input enter_date("enter date: ");
Menu::Input enter_what("enter a description of the activity: ");
...
Menu::Event<int, std::string, Cal::Calendar,
           bool (Cal::Calendar::*)(int, std::string)>
evaluate_meeting_event(evaluate_meeting, age, name, c, &Cal::Calendar::meeting);
...
Menu::Submenu meeting_menu("Manage meetings");
meeting_menu.add( ... );
...
Menu::Submenu main_menu("Main menu");
main_menu.add( ... );
...
```

# DD2387 Programsystemkonstruktion med C++

## Projektuppgift: Äventyrsspel

30 oktober 2012

Bonuspoäng: 4p om redovisningen sker i tid

I den här labben kommer du att lära dig att skriva ett lite större program med hjälp av arv, polymorfi och standardklasserna (STL). Ni får jobba i grupper om högst två personer. Vid redovisningen ska alla gruppmedlemmar kunna svara på frågor om alla delar av koden. Läs igenom hela lydelsen innan du börjar. För att uppgifterna ska känna mindre lösvryckta hänger de ofta ihop.

Spelet är ditt sätt att visa att du behärskar centrala delar av språket C++. Att du vet hur **polymorfism** och **virtual** fungerar, att du kan sätta restriktioner på data med **public/private/friends/const**, att du kan hantera **minnesallokering** rätt och att du kan använda STL.

Allmänna krav:

- Nyckelord som **const**, **virtual**, etc. ska användas på ett korrekt sätt.
- Din kod ska vara modulariserad i klasser och filer.
- Ditt program ska inte läcka minne, så var noga med dina konstruktörer och destruktörer.
- I denna labb ska du använda STL
- Du måste använda en makefile för att bygga dina klasser så att byggningen inte tar så lång tid. Ett exempel på en **makefile** hittar du på kurskatalogen.
- Uppgiften är löst hållen för att inte påverka din kreativitet negativt. Se dock till att du uppfyller kraven i 3.2. År du osäker på omfattningen av spelet så kan du fråga kursledare eller övningsassistent.
- Extrauppgifterna har ibland förslag på användning av nya standarden C++11, det är inga krav men rekommenderas varmt. Överhuvudtaget rekommenderas att prova på C++11 specifika saker som lamdafunktioner, for (each) loopar, auto m.m.

---

Vid redovisning ska följande vara förberett:

- En editor där din makefile är öppnad.
- Ett kommandoskal (shell) där prompten står i labbkatalogen.
- Alla program ska vara indenterade (*M-x indent-buffer* i emacs).
- Koden ska vara färdigkompilerad.
- Ni ska veta i vilken fil varje uppgift finns.
- Ni ska ha några skisser över kodstrukturen tillgängliga.
- Läst på och vara förberedd på frågeställningarna i 3.2

- Komprimerat källkoden i en arkivfil. Ett sätt att komprimera är att använda javas jar-program t.ex. `jar cvf source.zip *.cpp *.h makefile`

Se till att labhandledaren skriver under på ditt kvittenspapper.

Lycka till!

*Dungeons & Dragons*

Spelledare

**Introduktion:** Ett **rollspel** var från början ett **brädspel** där flera personer deltog. Spelet utspelade sig ofta i en fiktiv värld (*fantasy*) där spelarna mötte trollkarlar, krigare, drakar och andra varelser. Alla varelser i spelet hade egenskaper såsom styrka, uthållighet och magi. **Spelet leddes av en spelledare** som berättade för spelarna vem de mötte och vad som utspelade sig. Ett exempel på denna typ av brädspel är Drakar och Demoner (*Dungeons & Dragons*).

Du ska i den här labben implementera ett fullskaligt äventyrrsspel. Dina **klasser** kommer att representera troll, drakar m.m. Varelsernas egenskaper kommer att representeras av **klassernas medlemmar**. Spelledarens roll kommer att skötas av en **kommendotolk**, dvs ett textbaserat gränssnitt där du anger vad du vill göra. Efter du gjort ditt drag kommer spelledaren låta övriga varelser göra sitt.

Om du inte gillar drakar och demoner kan du istället skriva ett valfritt äventyrrsspel, t.ex. om hur du **pallar** äpplen av dina grannar på fredagskvällarna.

### 3.1 (arvsstruktur)

I denna uppgift kommer du att definiera klasshierarkier och objekt. Sist i uppgiften kommer objekten samverka med varandra.

- Börja med att välja ett lämpligt namn på den **namnrymd** som ska innehålla klasser och data i äventyrrsspelet.

a) Gör en klasshierarki för **aktörerna** i ett äventyrrsspel med **attribut** och **metoder**:



- Skapa en **klasshierarki för aktörerna**. Låt några av klasserna vara instansierbara och representera aktörer i spelet. *Exempel:* en trollkarl är en människa som är en aktör.
- Låt varje aktörklass ha en **egenskap** (gärna fler) specifik för den aktör de representerar. Några av **värdena** för varje aktör ska variera över spelets gång. *Exempel:* en trollkarl har ett värde för magi som minskar då han troller. Alla varelser har **kroppsspoäng** (liv). Genom att äta spenat kan en människa få mer kroppsspoäng, dock inte över ett maximalt värde. Alla varelser har **styrka**.
- Låt aktörerna ha en **mängd val** de kan göra då det är deras tur. Låt valen bero på var de är, vem de har i närheten och hur starka deras egenskaper är. *Exempel:* ett skadat troll letar mat, ett friskt troll **springer** runt slumprövigt, ett troll i närheten av en människa **blir argt**.

Förslag på funktioner i **basklassen** för aktörerna:

- **type()** – returnerar namnet på arten, t.ex. trollkarl eller drake
- **name()** – returnerar namnet på varelserna, t.ex. Merlin
- **action()** – aktörens tur att agera  
t.ex. *letar mat*, *springer*, *blir arg*, ...

- `go(direction)` – gå åt håll
- `fight(Character)` – slåss med
- `pick_up(Object)` – ta upp sak
- `drop(Object)` – släpp sak på marken
- `talk_to(Character)` – konversera med

b) Gör en klasshierarki för miljön med attribut och metoder:

- Skapa en klasshierarki som representerar miljön i ditt spel. Låt några av klasserna vara instansierbara och representera olika varianter på miljöer. *Exempel:* rum → inomhusmiljö → miljö, djungel → utomhustyp → miljö, strand → utomhustyp → miljö, hav → vatten → utomhustyp → miljö.
- Låt de olika typerna av miljöer ha utgångar åt olika håll. Ett miljöobjekt måste kunna svara på vilka riktningar som spelarna kan gå. Bortom varje utgång ska man komma till ytterligare ett miljöobjekt. *Exempel:* rum har bara fyra riktningar, skog har åtta. Vissa riktningar är blockerade av väggar, murar, berg och träd.
- Låt varje instans av en miljöklass ha en beskrivande text. Om du låter texten beskriva utgångar och föremål så blir spelet mer levande och roligare att spela.
- Olika typer av miljöer kan ha olika funktionalitet. På stranden kan man sola sig och bli mer attraktiv, i djungeln ser man bara gula ögon om natten, helig mark är fredad från strid, kvicksand får man inte stå still för länge på för då trillar man ner i en annan miljö.

Förslag på funktioner i basklassen för miljöerna:

- `directions()` – returnera vilka utgångar som finns `get_exits_name()`
- `neighbor(direction)` – returnera granne (t.ex. referens till objekt) i given riktning
- `description()` – returnera beskrivning av vad miljön innehåller, vilka föremål man kan ta och vilka aktörer som befinner sig på platsen.
- `enter(Character)` – aktör kommer till platsen
- `leave(Character)` – aktör går från platsen
- `pick_up(Object)` – någon tar upp ett föremål som finns på platsen
- `drop(Object)` – någon lägger ner ett föremål på platsen

c) Gör en klasshierarki för föremål med attribut och metoder:

- Skapa en hierarki av klasser som representerar föremålen i ditt äventyrs-spel. *Exempel:* börs → behållare → föremål, ring → föremål

- Låt alla föremål ha egenskaper. Alla föremål måste kunna svara på frågor om deras egenskaper, såsom dess pris, vikt, volym etc. *Exempel:* En ryggsäck har plats för ett visst antal saker/viss volym och går sönder om man lastar den för tungt, en ring kostar 500 silverpengar, ett svärd väger 2 kg.

Förslag på funktioner i basklasserna för föremålen:

- `weight()` – vikt
- `volume()` – volym
- `price()` – pris

Förslag på funktioner för behållare:

- `hold_weight()` – vikt innan behållaren går sönder
- `hold_volume()` – volym innan behållaren blir full
- `add(Object)` – lägg objekt i behållaren
- `remove(Object)` – ta bort objekt från behållaren

*spelplan*  
d) Sätt ihop en handfull miljöinstanser så att de bildar en liten spelplan. Instansiera några aktörer och lägg dem i en vektor `Vector<Character *>`. Lägg även ut instanser av föremål, varav någon behållare, i de olika miljöerna.

Iterera över vektorn och låt varje aktör utföra funktionen `action()`. Låt aktörerna plocka upp objekt, lägga ned objekt, gå genom dörrar, prata med varandra, bli arga och släss etc. Skriv ut information om hur spelet fortlöper, aktörernas namn och typ och vad de gör.

*Tips:* Du kan behöva en slumpgenerator `rand()` och en slumpinitierare `srand()` (se `man rand`). Om man använder `rand()` utan `srand()` får man alltid samma slumptalsföljd, vilket kan vara användbart när man letar fel.

**3.2** Du ska nu skapa ett riktigt, spelbart äventyrsspel för en spelare. Följande funktionalitet ska finnas:

- Ge en introduktion före spelet börjar där bakgrundshistorien berättas, målet står specificerat och några kommandon omnämns.
- Spelet ska ha en kommandotolk som sköter all inmatning till spelet. Speltolk tar ett kommando från tangentbordet och utför handlingen. Kommandotolken bör klara alla funktioner som aktörernas basklass specificerar. *Exempel:* `pick up sword, go north, buy shield`.
- Alternativt: visa spelplanen med teckengrafik. Låt användaren manövrera genom att hämta in ett tecken med `getch`.
- Låt eventuella strider ske i omgångar tills någon part flyr eller avlider. Striderna kan t.ex. styras med tärningar.

+ `rand()`  
`srand()`

- Låt minst en händelse bero på ytter/tidigare omständigheter. Exempel: dörren öppnas bara du har rätt nyckel, lönndörren visar sig bara om du försöker gå i den riktningen, vakten släpper in dig bara om du tidigare talat med en speciell instans av en aktör, fallluckan öppnar sig bara om du väger tillräckligt mycket.
- Spelet ska ha ett mål. När målet är uppfyllt ska spelet avslutas.
- Du som kan spelet (och vet var den hemliga nyckeln m.m. finns) ska kunna spela spelet från start till mål under en redovisning.
- Du ska kunna visa upp några virtuellt nedärvda metoder vars implementering väsentligen skiljer sig och du ska kunna diskutera designskillnaden mellan att deklarera metoder virtuella eller inte. Därför måste spelet ha en viss omfattning, ett alltför tunnt spel kan underkänna. Om du kan visa detta med enbart en arvstruktur med flera intressanta virtuella ary så är det tillräckligt. Alla tre arvsstrukturer kan göras intressanta. En del kanske vill ha många olika sorters aktörer (Harry Potters kompisar, lärar, husdjur, fiender, porträtt, spöken m.m.) eller invecklade strider (expelliarmus) andra kanske vill ha roliga föremål och gåtor (da Vinci koden) eller spännande miljöer (vid-behovs rum, lustiga huset m.m.) Om du känner dig osäker, ta kontakt med din övningsledare.
- Du ska lösa hur olika objekt når varandra. Håller rummen reda på vad som är i dem och vet aktörerna vilket rum de är i? Vet objekten vem som håller i dem. Man kan ha endast en global vektor där allting finns och som man letar upp vad man vill ha. Eller så har man flera behållare som pekar eller refererar till samma objekt, t.ex. att både rummen och aktörerna håller reda på var aktören befinner sig. Man kan också tänka sig överföring av ägande, sjön ger bort excalibur till prins Arthur (jämför med autopointers/uniquepointers funktionalitet). I princip kan man jämföra designproblemet med hur objekt når varandra som valet mellan enkellänkad och dubbellänkad lista. Det är enklare att nå vissa saker med en dubbellänkad lista men man måste komma ihåg att uppdatera både next och previous.

Lönndörr = secret door

Falllucka = trapdoor

Expelliarmus = spell used to disarrange another wizard, typically by causing the victim's wand to fly out of reach.

Sword in the stone of King Arthur

## Inför redovisning

Förbered några handritade skisser för att underlätta förståelsen av er design. Om du gjort UML-diagram påtala detta vid redovisning.

- Hur sker minnesallokeringen? Var görs allokering och destruktion?
- Läcker programmet minne? Kör en gång med valgrind.
- Är alla read-only metoder const-deklarerade?
- Beskriv klasshierarkin? Visa klassdiagram.
- Hur ser slingan ut som hanterar händelser. Hur hanteras händelser?

valgrind --tool=memcheck --leak-check=yes ./camelot  
 valgrind --leak-check=full --show-reachable=yes --num-callers=20 --track-fds=yes ./camelot

- Vad är det som håller reda på var spelaren är? Vad håller reda på alla andra objekt i spelet? Hur ser det ut i minnet, visa en minnesbild.
- Hur kopplas miljöerna ihop? Visa minnesbild.
- Hur hittar man saker/grannar? Hur sker uppslagningen?
- På vilket sätt skiljer sig karaktärer i spelet?
- Hur sker inmatning? Hur sker parsningen av det som inmatas?
- Hur fungerar action-metoden?

### Betygshöjande extrauppgifter

map

Extrauppgift 3.1 (6p krav för betyg D) Använd pekare till medlemsfunktioner och lambdafunktioner i ditt program.

En teknik för att undvika stora if-satser är att lägga alternativen tillsammans med funktionspekare i en map. Skriv kommandotolkten med hjälp av funktionspekare och pekare till medlemsfunktioner. Skapa en std::map och låt namnet på varje kommando och/eller kortkommando vara nyckel till en pekare till kommandot. Skapa ytterligare en std::map och låt namnet på en aktör vara nyckeln till det objekt aktören representeras av. När ett kommando verkar på en aktör, använd funktionspekaren genom objektet. Olika kommandon kan ge upphov till funktionspekare av olika typer. Ibland vill man ha ropa på funktioner med olika antal element. Ett simpelt sätt är att använda en map på första nyckelordet och skicka vidare resten av argumenten till en ny f-sats/map-uppslagning. I nya C++11 standarden finns det variadic templates, som tillhandahåller ett mer avancerat sätt att hantera problemet.

77

Syntaxen för en medlemspekare är krångligare (pekaren till objektet måste med) än en vanlig funktionspekare och därfor ska den användas så att ni i framtiden inte backar för krånglig syntax.

Använd även en lamdafunktion någonstans i ditt program. Om du har en elegantare lösning på denna extrauppgift med funktorer och/eller templates kan du redovisa den elegantare lösningen. En annan lösning som ligger nära till hands är en map med omväxlande funktionspekare och lamdafunktioner men om du redovisar en alternativ lösning så visa även ett exempel på en medlemsfunktionspekare (deklarering/anrop) så att handledaren övertygas om att du behärskar syntaxen.

Extrauppgift 3.2 (4p) Spelets position ska gå att spara till fil och ladda från fil. Använd strömmar (*streams*). Exempel: save my\_game1, load game7.

Man ska kunna utvidga ditt spel så att det går att ladda ett nytt spel under spelets gång utan att det läcker minne.

Extrauppgift 3.3 (5p) Låt spelets karta och aktörer definieras av en fil. Ladda all information såsom miljöer, karta (miljöernas förhållande till varandra), aktörer och föremål från fil. Observera att det bara är objekten och deras egenskaper som ska läsas från filen medan objektens funktioner och beteenden ligger i klasserna de instansierar.

**Filformatet** kan t.ex. se ut som

MIL1:Du står i ett liten grotta. En svag belysning  
avslöjar ett stort hål i golvet.:MIL2:OBJ2:AKT3  
MIL2:Du befinner dig i en trång gång mellan två  
salar. Väggarna är väta och hala.:MIL1,MIL3:OBJ1:  
MIL3:Du är i ett stor sal där tre stora bord står  
dukade.:MIL2::AKT1,AKT2  
AKT1:TROLL:Gruff-Gruff:kroppspoäng=17,iq=3:  
OBJ4,OBJ5  
OBJ1:BEHÅLLARE:OBJ6,OBJ7:en liten ryggsäck:10kg,  
10liter,2daler  
OBJ2:PENGAR:en mängd daler:0kg,0liter,10daler  
OBJ3:ENHET:ett bredsvärd med förgylld egg:3kg,  
1liter,350daler  
...

Spelar: äldre svenskt  
silver- eller kopparuyt.

Egg: skarp kant på skärande verktyg  
t.ex. kniv, yxa, svärd

Filens ska kollas så att den innehåller konsistenta användningar av objekt: utgångar måste vara till rum som finns, föremål som används ska vara definierade etc. Hur ska man hantera händelser som beror på yttre/tidigare omständigheter?

*Tips:* Gör denna uppgift tillsammans med extrauppgift 3.2 så får du mycket på köpet.

Dynamic  
memory  
ptr

**Extrauppgift 3.4 (5p) krav för betyg B** Inför minst tre sorters objekt som kan konstrueras (och destrueras) under spelets gång och inte endast när spelet tar slut. Du får själv välja om det ska vara aktörer, föremål eller miljöer. Ett exempel för aktörer kan vara att det då och då vaknar upp **vampyrer** som börjar gå omkring i världen. Dessa vampyrer kan slås ihjäl någon helt annanstans i spelet av spelaren eller av någon annan vampyrdräpare som råkar gå förbi. Ett annat exempel för föremål kan vara ett **appelträd** där spelaren eller andra busungar kan palla mogna eller omogna frukter. Äpplena kanske man kan ge vidare till någon annan, t.ex. sin söndagsskolefröken som gör äppelpaj av dem. **Dynamiska miljöer** som det kan skapas flera av, isflak som flyter förbi? Tjänsterum på byråkrativerket som ibland infinner sig på slumpmässigt väningsplan

nya object

dröpa = slay, kill

... Redovisa hur spelet håller reda på dessa objekt som skapas och dör under spelets gång. Du ska kunna argumentera för att din lösning håller för att utvidgas med fler sorter av den här typen av objekt.

I C++11 finns **shared\_ptr** och **weak\_ptr** som du kan experimentera med. Minst en av dina dynamiskt allokerade objekt ska dock allokeras och deallokeras med new och delete så att du kan visa dina färdigheter i att hantera dynamiskt allokerat minne. Testa minnesläckor med valgrind.

**Extrauppgift 3.5 (9p)** Gör ett grafiskt händelseorienterat (t.ex. klickbart) gränssnitt till spelet. **Varning** - räkna med åtskilliga arbetstimmar, det är kanske den mest dyrköpta extrapoänguppgiften. **Koden måste vara väl strukturerad** använd t.ex. designmönstret **model-view-controller**. Grafikkoden ska i huvudsak vara åtskild från den övriga koden och uppdelad i klasser. Kom ihåg att labben ska imponera med dina kunskaper i C++ inte dina kunskaper i användandet av grafikrutiner. Extrauppgiften med en map med funktionspekare

Grafik

kan inte göras som beskrivet på kommandotolken. Gör istället en annan map med medlemsfunktionspekare som ersätter någon annan stor if-sats i koden.

*Li; mutex  
std::; thread*

**Extrauppgift 3.6 (9p)** Gör ett trådat nätverksspel. Inför ett 'tick' som går ett visst antal gånger per sekund genom spelet. Låt alla aktörer som inte är spelare agera efter ett visst antal 'ticks'. Exempel:

A troll has arrived from the east  
[tre sekunder går]  
A troll says Yum, yum I will eat you  
A troll hits you  
...

Gör det möjligt att låta fler än en spelare spela spelet. **Varning** - räkna med åtskilliga arbetsstimmar, det är en ganska dyr extrapoänguppgift. **Koden måste vara väl strukturerad**. Nätverkskoden ska i huvudsak vara åtskild från den övriga koden och uppdelad i klasser. Kom ihåg att labben ska imponera med dina kunskaper i C++ inte dina kunskaper i användandet av tråd- och **nätverksbibliotek**. Använd gärna trådbiblioteket i nya C++-standarden.

#### **Extrauppgift 3.7 (4p)**

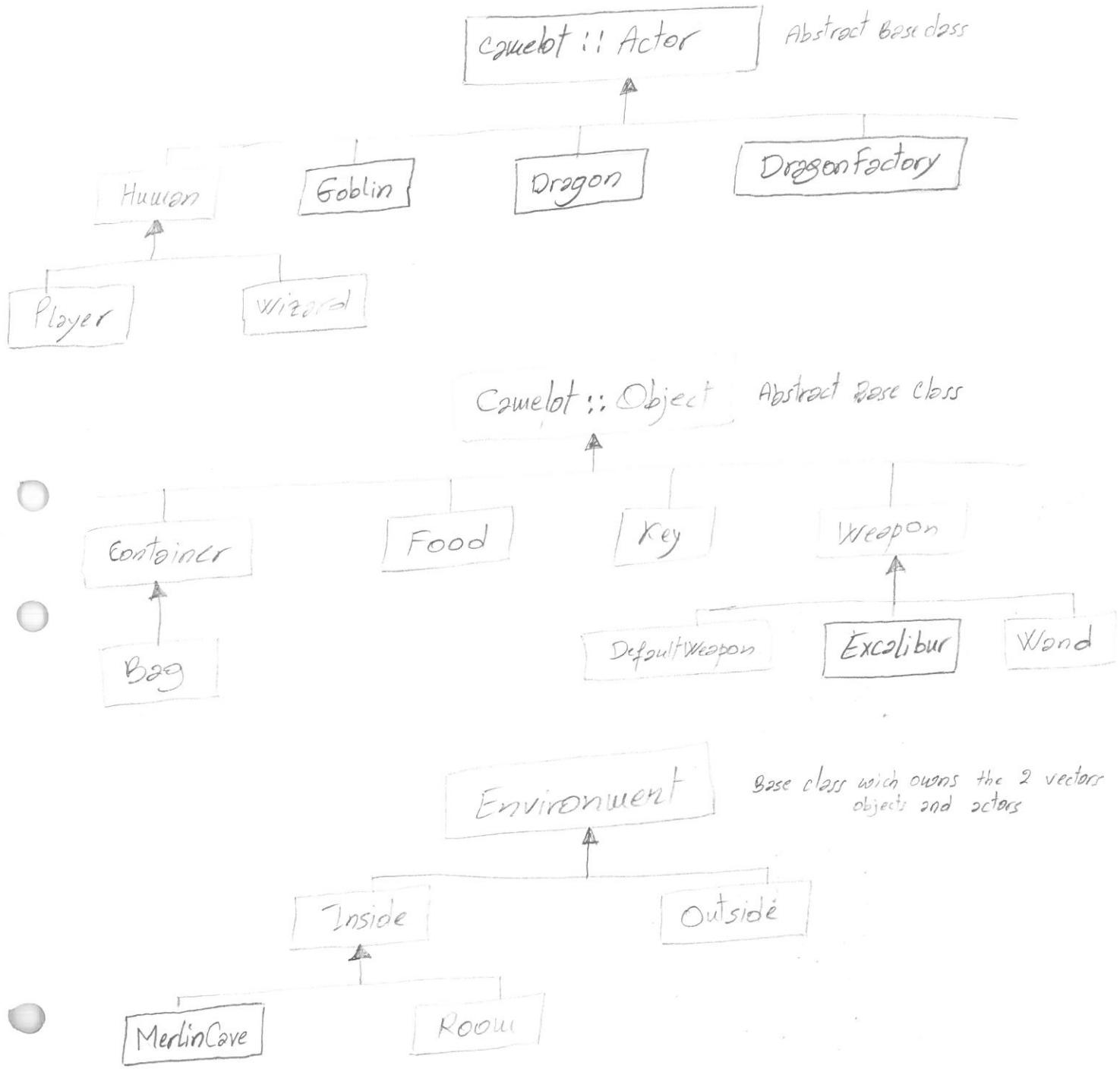
**UML** (Unified Modeling Language) kan man använda för att dokumentera ett kodprojekt. På [www.uml.org](http://www.uml.org) finns länkar till UML-verktyg och UML-tutorials. I den här extrauppgiften ska du dels visa upp korrekta klassdiagram (det finns verktyg som kan generera dem från koden), dels ska du skriva minst **fem** scenarion och rita tillhörande **sekvensdiagram**, dels ska du rita minst ett **tillståndsdiagram**.

Ett scenario är en textbeskrivning där man steg för steg i textform beskriver ett användarscenario. T.ex. aktören plockar upp ett **aztekiskt guldmyntrum** från en piratkista. En dov röst hörs som förbannar aktören. Ett spöke kommer fram ur skuggorna och slåss med aktören. Spöket dör.

Till varje scenario hör ett eller flera sekvensdiagram.

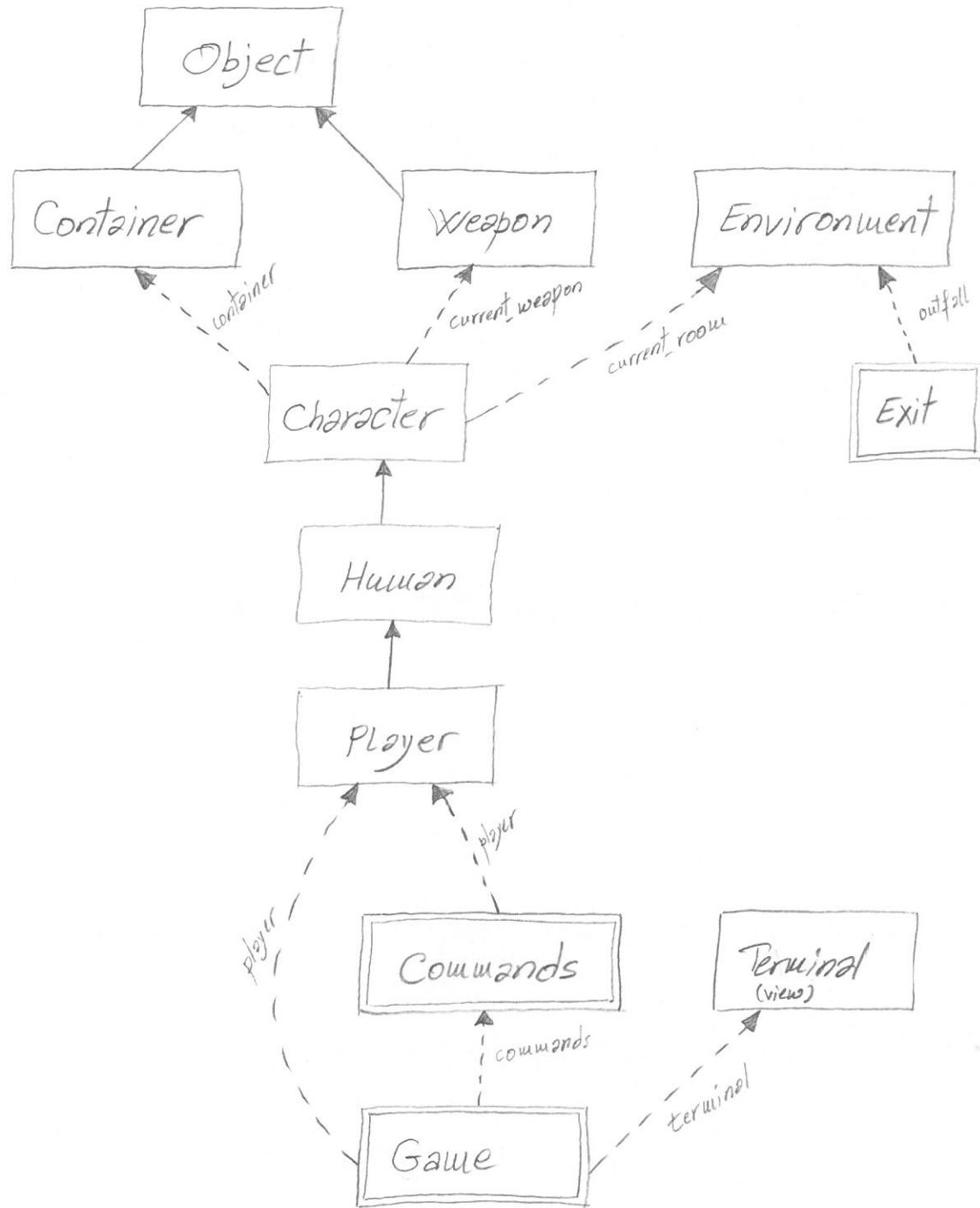
Tänk på att varje scenario bara kan illustrera en gren av en if-sats, eller ett visst antal varv i en loop.

Identifiera och redovisa de scenarien som är spelmässigt eller programmeringstekniskt intressanta i just ditt spel. Programmeringstekniskt intressanta kan t.ex. vara ett scenario där man vinner. Spelmässigt intressanta kan vara ett scenario där man får en avgörande ledtråd till hur man ska klara spelet.



- view → Terminal
- commands (parser)
- game
- Exit





The UML specifies 2 types of scope for members; instance and classifier!

— Classifier members: are commonly recognized as "STATIC" in many program languages. The scope is the class itself.

- Attribute values are equal for all instances.
- Method invocation does not affect the instance's state.

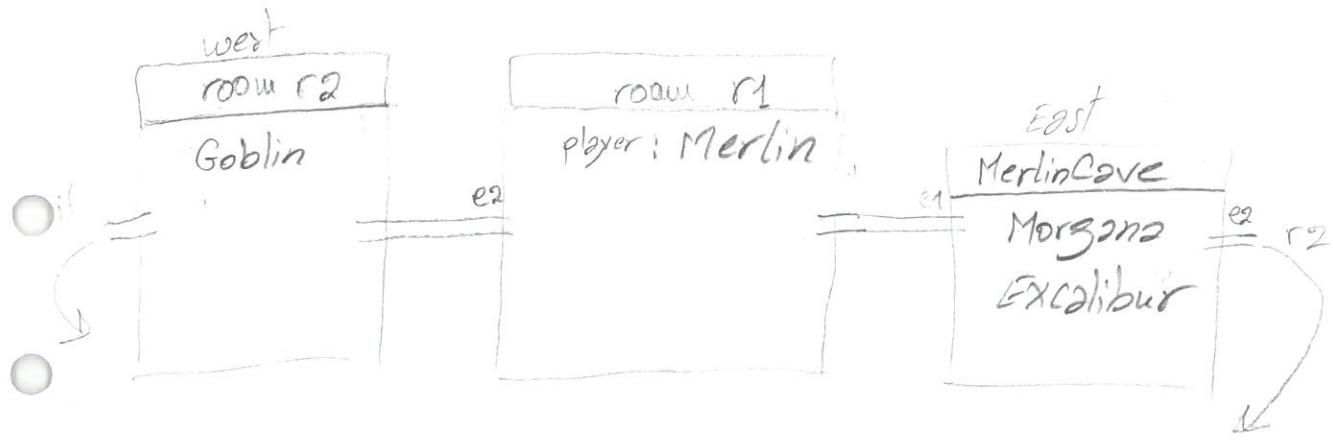
— Instance members: are scoped to a specific instance.

- Attribute values may vary between instances.
- Method invocation may affect the instance's state (i.e., change instance's attributes)

To indicate a classifier scope for a member, its name must be underlined. Otherwise, instance scope is assumed by default.

Dependency: is a weaker form of bond which indicates that one class depends on another because it uses it at some point in time. One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.

This is different from an association, where an attribute of the dependent class is an instance of the independent class.



- RolePlay
- Generate a text map / graphical map ("map.txt") ("map.png")
- warrior, assassin
- setw(int)

1. The player input a line of instructions.
2. The programme divides the line into individual words.
3. The programme interprets the words and check if the instruction makes sense.
4. The programme executes the command, if it makes sense, or by default informs the player that the command made no sense.
5. the programme loops back to asking for a command to be input by the player.

Hur rymmer värger i map?

- En global variable t.ex. allrum
- Vectors, men vi måste uppdatera!

unique\_ptr

GameCollGraph.h

shared\_ptr

auto\_ptr

Ex uppgift 1 : if → map

- if sätterna . vi kan ta bort dem genom att använda map (dictionary i Python)
- Pointer till medlem funktioner \*()
- New, delete

UML

- Klassdiagram (Diagram)

- Entity Relation

- Flödes diagram

- USE cases

sekvens diagram (Dokumentation  
uppgift)

scrum ?!

load

static

private

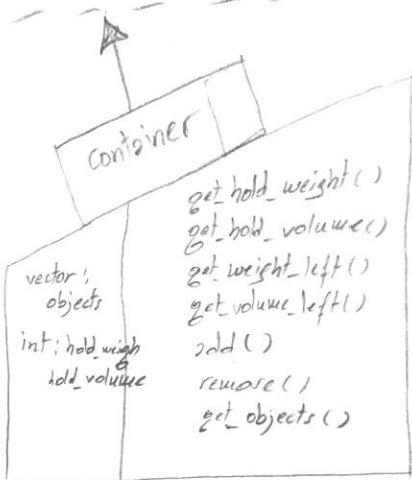
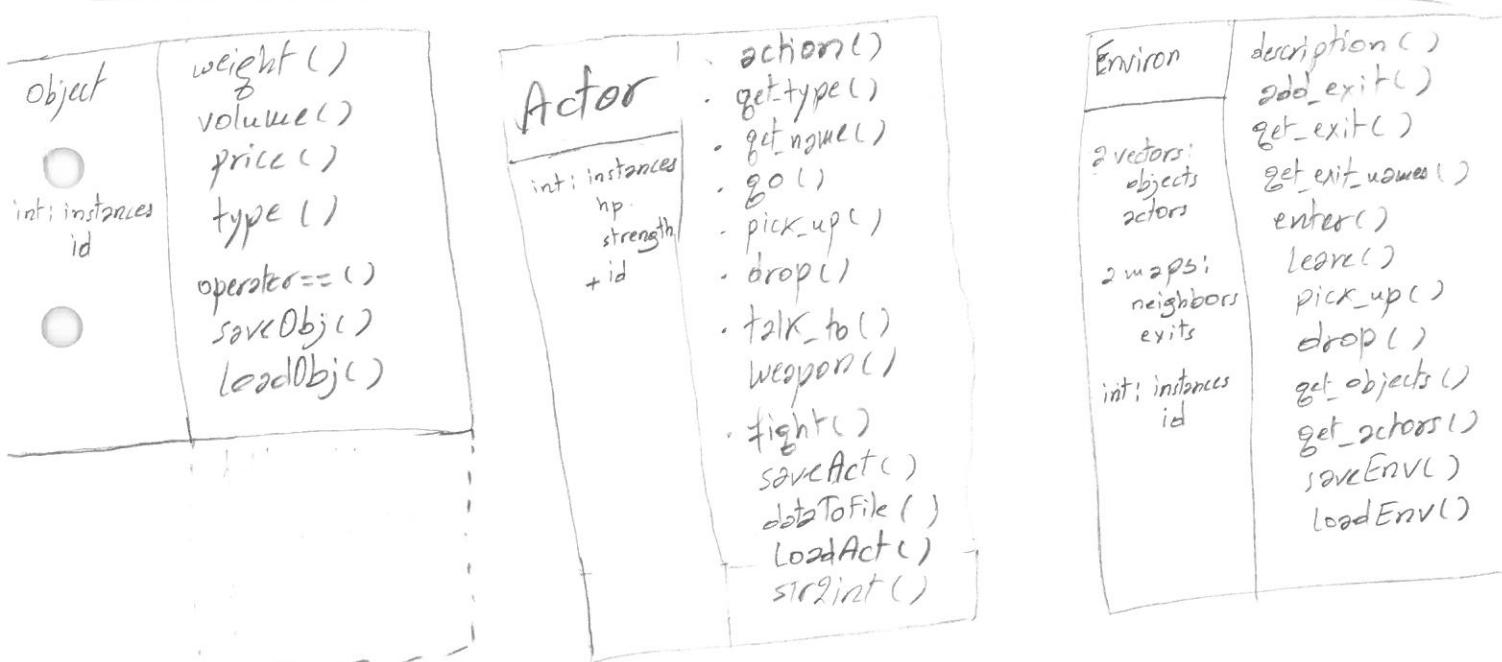
id.

recurse

Göron Bourne, MENA.

- Emergence of STL - std::vector (3-3.1.2) STL
- unique\_ptr, weak\_ptr, shared\_ptr.
- m@ns.vestin@raysearchlabs.se

## Dungeon Crawl Stone Soup manual





C++

8.11.2013

# Text Åventursspel (Lab 3)

module add gcc/4.8.1 } Thread  
module avail gcc

~~diff~~

## Objektorienterad Programmering

Logik står ~~in~~ i ett annat plats!?

Logik: slingor, villkor (while, if, --)

vi måste visa Dynamisk Bindning (Dynamic Binding)  
istället att använda if, switch, --

Actors, Items, Environment → Avstruktur

const = allt som är READ ONLY

virtual, private, public, --

virtual & Runtime

JAVA är alltid "virtual" ⇒ det kostar tid!  
men i C++ vi kan designa (välja)  
med FINAL i java kan vi inte årva.

N.B. Virtual ⇒ Runtime, inte compile time

- Vad kan man göra med virtual metoder?

Dynamisk Bindning: when we call a virtual function through a pointer or reference, the call will be dynamically bound.

- Hur rymmer hängar shop?

2 strategier:

- Genom en global variabel t.ex. allrum

- Genom vectors, men vi måste UPPDATERA dem.

N.B. smart Pointers: (earlier & safer)

shared\_ptr, unique\_ptr (instead of

- see Dynamic Memory in the book p450 Auto\_ptr)

## UML (Unified Modeling Language) diagrams

- Klassdiagramm (Doxysena)
- Entity Relationship (ER) model: is a data model for describing a DATABASE in a abstract way
- Flödesdiagramm (Flowchart): is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.
- Use Case Diagram: is a representation of a user's interaction with the system and depicting the specifications of a use case
- Sequence Diagram: is a kind of interaction diagram that shows how processes operate with one another and in what order.

## Extra Uppgift 1 if satser → map

- Vi kan ta bort if-satserna genom att använda map (dictionary in Python)
- Memory leak: using new, which allocates an object in dynamic memory and returns a pointer to that object. we forgot to free the memory by delete.
- Pointer till medlem funktioner  $\text{type}(*\text{p})(\text{type}, \text{type})$   
to declare a pointer that can point at a function, we declare a pointer in place of the function name

Ex: bool f(string s, int b);  $\rightarrow$  bool (\*pf)(string s, int b);

Lambdas

```

int y=3;
[ ]() {cout << "Hej" } ;
[ ](int x) {cout << x+1} ;
[&y] {cout << y} ;

```

A lambda expression has the form:

[capture list] (parameter list) [→ return type] {function body}

- We can omit either or both of the parameter list and return type but must always include the capture list and function body.
- A lambda expression represents a callable unit of code. It can be thought of as an UNNAMED, INLINE function.
- We call a lambda the same way we call a function by using the call operator.

EX: auto f = [ ] {return 42;} ;

cout << f() << endl; // prints 42

Access Control Specifiers: public, private, protected are also meaningful in the context of inheritance.  
The following table describes the access rights of inherited methods.

- note that the resulting access is always the most restrictive of the two.

		public	pub	prot	priv
		protected	prot	prot	priv
Member Access	private	n/a	n/a	n/a	n/a
	public	public	protected	private	private

public  
protected  
private

## General rules for Access control specifiers

- Private methods of the base class are not accessible to a derived class (unless the derived class is a FRIEND of the base class).
- if the subclass is derived publically then :
  1. Public methods of the base class are accessible to the derived class.
  2. Protected methods of the base class are accessible to derived classes and friends only.

## Inventory

Alla .o filer borde hamna i en egen mapp så det inte blir så stödigt

Använd **medlemspekare** istället för statiska funktionspekare

Förslag: Låt varje environment innehålla en lista över **Exit-objekt**, där ett Exit-objekt innehåller typ "direction", "new\_environment\*", "locked", "symmetric\_lock" och "unlocking\_item". Kanske även någon liten beskrivning av utgången och ett eventuellt lås. På det här viset kan vi implementera icke-symmetriska lås, t.ex. dörrar som gått i baklås.

Dessutom tror jag det blir lättare att hålla rätt på pekarna, jämfört med att varje Exit ska innehålla ett par av utgångar. Moment 22-problemet kvarstår dock med instansiering av Environments; för att skapa en miljö med utgångar måste man först ha skapat en annan miljö. Det kan dock lösas genom att först skapa miljöerna och sedan lägga till utgångar genom **add\_exit** eller dylikt.

Låter finfint :) DONE

Funkar det med private på instances i Object och Actor?

Stoppa in player i environmenten han är i DONE

**Storyn** kan vara som starwars, man är anakin osv

Så målet med spelet är att bara slakta alla som finns, som anakin gör i jedi-templet i Episode III

Ett förslag är att när det händer saker i ett rum så är det någon "**global**" metod som hanterar vad som händer, så man kan dissa det om det händer saker i ett rum som man inte är i

Diskutera format på sparfiler samt kartor (3.2 och 3.3)

En fight kan instansiera en speciell miljö utan exits, så registrerar man samtidigt ett nytt kommando till terminalen som kan vara typ "**fight**" (så man kan välja vapen osv)

Låta Inside ta emot Environments-pekarer som tidigare och automatiskt skapa standard-exits till dessa.

Se över Environment::**description()**. Skriva till stdout eller returnera sträng?

Fixa låsmekanismen på Exits.

Skapa **PairExit** som returnerar olika **get\_outfall** beroende på vilket av de två rummen man står i, o.s.v.

Se över **Segmentation Fault** när man dör.

Se till så man måste låsa upp en dörr för att komma till **evil lair** (för att behöva använda nyckel för att klara spelet, light saber får INTE ligga i det låsta rummet :D)

**stringtoint()** måste hantera fel

Rules to conform to Objects

-- Objects should always be owned by exactly one of the following.

An Environment  
A Container  
An Actor

In that way it is easier to make sure no objects are causing memory leaks or unexpected null references. We are thereby implementing the ownership semantics from `auto_ptr`; there is always exactly one object owning another object.

-- If an object is to be deleted, only the owning object is allowed to do that and is also responsible for removing the pointer to that object.

---

```
actor : vampire,vampire_factory,troll,Human(wizard,player)
environment : Inside(Room,evil_lair),Outside
object : container(bag),food,key
```

```
game
game_commands
exit -> are used in environment to allow actors travelling
terminal
weapon : light_saber,wand,default_weapon
bad_format
```

---

```
Actor : Wolf(CrazyWolf),Human(OldMan,Player)
Object : Firewood,Backpack,Container,Key,Weapon(Sword)
Place : OutdoorPlace(House,Woods,Portal)
```

```
Map
Direction/DirectionSet
Controller
Loader
```

## Doxxygen

Generate documentation from source code

Doxxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.

Doxxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in L<sup>A</sup>T<sub>E</sub>X) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.
2. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency

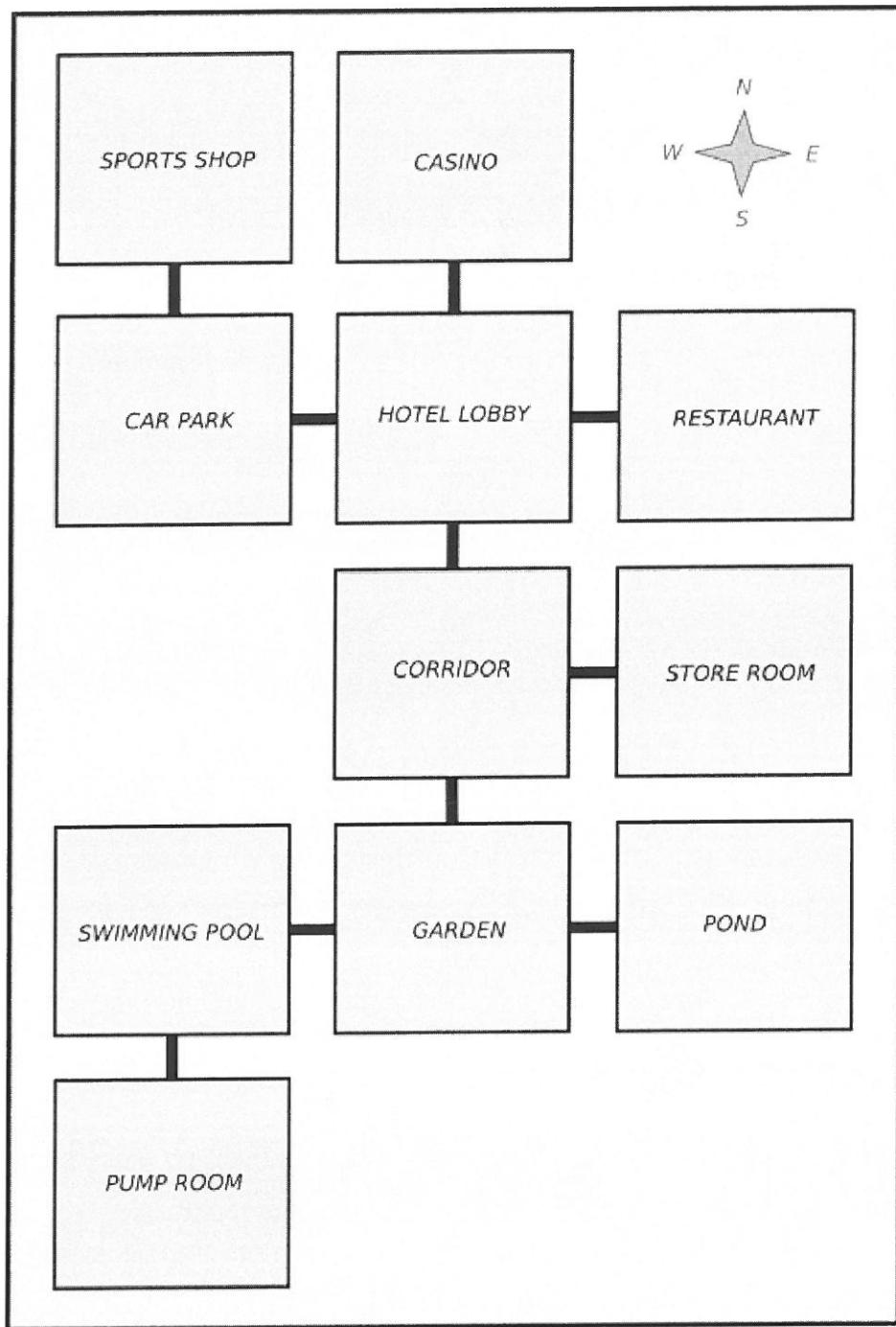
graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are available.

- Doxygen has built-in support to generate inheritance diagrams for C++ classes.

## Text game example



```
#include <iostream>
#include <string>
#include <vector> // For the command handling function.
#include <cctype> // Will be used to eliminate case sensitivity problems.

using namespace std;

enum en_DIRS {NORTH, EAST, SOUTH, WEST};
enum en_ROOMS {SPORTSHOP, CASINO, CARPARK, LOBBY, RESTAURANT,
               CORRIDOR, STOREROOM, POOL, GARDEN, POND, PUMPROOM};
enum en_VERBS {GET, DROP, USE, OPEN, CLOSE, EXAMINE, INVENTORY, LOOK};

const int NONE = -1;
const int DIRS = 4;
const int ROOMS = 11;
const int VERBS = 8;

struct word
{
    string word;
    int code;
};

struct room
{
    string description;
    int exits_to_room[DIRS];
};

// -----
-----

void set_rooms(room *rms)
{
    rms[SPORTSHOP].description.assign("sports shop");
    rms[SPORTSHOP].exits_to_room[NORTH] = NONE;
    rms[SPORTSHOP].exits_to_room[EAST] = NONE;
    rms[SPORTSHOP].exits_to_room[SOUTH] = CARPARK;
    rms[SPORTSHOP].exits_to_room[WEST] = NONE;

    rms[CASINO].description.assign("bustling casino");
    rms[CASINO].exits_to_room[NORTH] = NONE;
    rms[CASINO].exits_to_room[EAST] = NONE;
    rms[CASINO].exits_to_room[SOUTH] = LOBBY;
    rms[CASINO].exits_to_room[WEST] = NONE;

    rms[CARPARK].description.assign("car park");
    rms[CARPARK].exits_to_room[NORTH] = SPORTSHOP;
    rms[CARPARK].exits_to_room[EAST] = LOBBY;
    rms[CARPARK].exits_to_room[SOUTH] = NONE;
    rms[CARPARK].exits_to_room[WEST] = NONE;

    rms[LOBBY].description.assign("hotel lobby");
    rms[LOBBY].exits_to_room[NORTH] = CASINO;
    rms[LOBBY].exits_to_room[EAST] = RESTAURANT;
    rms[LOBBY].exits_to_room[SOUTH] = CORRIDOR;
    rms[LOBBY].exits_to_room[WEST] = CARPARK;
```

```
rms[RESTAURANT].description.assign("restaurant");
rms[RESTAURANT].exits_to_room[NORTH] = NONE;
rms[RESTAURANT].exits_to_room[EAST] = NONE;
rms[RESTAURANT].exits_to_room[SOUTH] = NONE;
rms[RESTAURANT].exits_to_room[WEST] = LOBBY;

rms[CORRIDOR].description.assign("corridor");
rms[CORRIDOR].exits_to_room[NORTH] = LOBBY;
rms[CORRIDOR].exits_to_room[EAST] = STOREROOM;
rms[CORRIDOR].exits_to_room[SOUTH] = GARDEN;
rms[CORRIDOR].exits_to_room[WEST] = NONE;

rms[STOREROOM].description.assign("store room");
rms[STOREROOM].exits_to_room[NORTH] = NONE;
rms[STOREROOM].exits_to_room[EAST] = NONE;
rms[STOREROOM].exits_to_room[SOUTH] = NONE;
rms[STOREROOM].exits_to_room[WEST] = CORRIDOR;

rms[POOL].description.assign("swimming pool area");
rms[POOL].exits_to_room[NORTH] = NONE;
rms[POOL].exits_to_room[EAST] = GARDEN;
rms[POOL].exits_to_room[SOUTH] = PUMPROOM;
rms[POOL].exits_to_room[WEST] = NONE;

rms[GARDEN].description.assign("tranquil garden");
rms[GARDEN].exits_to_room[NORTH] = CORRIDOR;
rms[GARDEN].exits_to_room[EAST] = POND;
rms[GARDEN].exits_to_room[SOUTH] = NONE;
rms[GARDEN].exits_to_room[WEST] = POOL;

rms[POND].description.assign("patio with a fish pond");
rms[POND].exits_to_room[NORTH] = NONE;
rms[POND].exits_to_room[EAST] = NONE;
rms[POND].exits_to_room[SOUTH] = NONE;
rms[POND].exits_to_room[WEST] = GARDEN;

rms[PUMPROOM].description.assign("damp pump room");
rms[PUMPROOM].exits_to_room[NORTH] = POOL;
rms[PUMPROOM].exits_to_room[EAST] = NONE;
rms[PUMPROOM].exits_to_room[SOUTH] = NONE;
rms[PUMPROOM].exits_to_room[WEST] = NONE;

}

// -----
-----

void set_directions(word *dir)
{
    dir[NORTH].code = NORTH;
    dir[NORTH].word = "NORTH";
    dir[EAST].code = EAST;
    dir[EAST].word = "EAST";
    dir[SOUTH].code = SOUTH;
    dir[SOUTH].word = "SOUTH";
    dir[WEST].code = WEST;
    dir[WEST].word = "WEST";
```

```
}

// -----
-----

void set_verbs(word *vbs)
{
    // enum en_VERBS {GET, DROP, USE, OPEN, CLOSE, EXAMINE, INVENTORY, LOOK};
    vbs[GET].code = GET;
    vbs[GET].word = "GET";
    vbs[DROP].code = DROP;
    vbs[DROP].word = "DROP";
    vbs[USE].code = USE;
    vbs[USE].word = "USE";
    vbs[OPEN].code = OPEN;
    vbs[OPEN].word = "OPEN";
    vbs[CLOSE].code = CLOSE;
    vbs[CLOSE].word = "CLOSE";
    vbs[EXAMINE].code = EXAMINE;
    vbs[EXAMINE].word = "EXAMINE";
    vbs[INVENTORY].code = INVENTORY;
    vbs[INVENTORY].word = "INVENTORY";
    vbs[LOOK].code = LOOK;
    vbs[LOOK].word = "LOOK";
}

// -----
-----

void section_command(string Cmd, string &wd1, string &wd2)
{
    string sub_str;
    vector<string> words;
    char search = ' ';
    size_t i, j;

    // Split Command into vector
    for(i = 0; i < Cmd.size(); i++)
    {
        if(Cmd.at(i) != search)
        {
            sub_str.insert(sub_str.end(), Cmd.at(i));
        }
        if(i == Cmd.size() - 1)
        {
            words.push_back(sub_str);
            sub_str.clear();
        }
        if(Cmd.at(i) == search)
        {
            words.push_back(sub_str);
            sub_str.clear();
        }
    }
    // Clear out any blanks
    // I work backwards through the vectors here as a cheat not to invalidate
the iterator
    for(i = words.size() - 1; i > 0; i--)
    {
```



```
if(words.at(i) == "")  
{  
    words.erase(words.begin() + i);  
}  
}  
// Make words upper case  
// Right here is where the functions from cctype are used  
for(i = 0; i < words.size(); i++)  
{  
    for(j = 0; j < words.at(i).size(); j++)  
    {  
        if(islower(words.at(i).at(j)))  
        {  
            words.at(i).at(j) = toupper(words.at(i).at(j));  
        }  
    }  
}  
// Very simple. For the moment I only want the first two words at most (verb  
/ noun).  
if(words.size() == 0)  
{  
    cout << "No command given" << endl;  
}  
if(words.size() == 1)  
{  
    wd1 = words.at(0);  
}  
if(words.size() == 2)  
{  
    wd1 = words.at(0);  
    wd2 = words.at(1);  
}  
if(words.size() > 2)  
{  
    cout << "Command too long. Only type one or two words (direction or v  
e and noun)" << endl;  
}  
}  
  
-----  
void look_around(int loc, room *rms, word *dir)  
{  
    int i;  
    cout << "I am in a " << rms[loc].description << "." << endl;  
    // LOOK should also allow the player to see what exits exist from the cur  
    rent room.  
    for(i = 0; i < DIRS; i++)  
    {  
        if(rms[loc].exits_to_room[i] != NONE)  
        {  
            cout << "There is an exit "  
            << dir[i].word << " to a "  
            << rms[rms[loc].exits_to_room[i]].description << "." << endl  
        }  
    }  
}
```

// -----

```
bool parser(int &loc, string wd1, string wd2, word *dir, word *vbs, room *rms)
{
    int i;
    for(i = 0; i < DIRS; i++)
    {
        if(wd1 == dir[i].word)
        {
            if(rms[loc].exits_to_room[dir[i].code] != NONE)
            {
                loc = rms[loc].exits_to_room[dir[i].code];
                cout << "I am now in a " << rms[loc].description << endl;
                return true;
            }
            else
            {
                cout << "No exit that way." << endl;
                return true;
            }
        }
    }

    // Handle verbs. As a reference, here are the verbs I am using in the game.
    // enum en_VERBS {GET, DROP, USE, OPEN, CLOSE, EXAMINE, INVENTORY, LOOK};
    int VERB_ACTION = NONE;

    for(i = 0; i < VERBS; i++)
    {
        if(wd1 == vbs[i].word)
        {
            VERB_ACTION = vbs[i].code;
            break;
        }
    }

    if(VERB_ACTION == LOOK)
    {
        // This is an example of sub proceduralizing a function from the parser.
        look_around(loc, rms, dir);
        return true;
    }

    if(VERB_ACTION == NONE)
    {
        cout << "No valid command entered." << endl;
        return true;
    }
    return false;
}

// -----
// -----
```

```
int main()
{
```

```
int main()
{
    string command;
    string word_1;
    string word_2;

    room rooms[ROOMS];
    set_rooms(rooms);

    word directions[DIRS];
    set_directions(directions);

    word verbs[VERBS];
    set_verbs(verbs);

    int location = CARPARK; // using the enumerated type identifier, of course.

    while(word_1 != "QUIT")
    {
        command.clear();
        cout << "What shall I do? ";
        getline(cin, command);
        //cout << "Your raw command was " << command << endl;

        word_1.clear();
        word_2.clear();

        // Call the function that handles the command line format.
        section_command(command, word_1, word_2);

        // Call the parser.
        if(word_1 != "QUIT")
        {
            parser(location, word_1, word_2, directions, verbs, rooms);
        }
    }
    return 0;
}
```



# SAM

≡

actor

oldman

backpack

outdoor-place

container

place

controller

player

crazy\_wolf

- portal

direction/set

sword

firewood

[types.h

- goal

weapon

- house

[win.txt

human

wolf

key

woods

loader

[maps  
saves

[main

[userfile

wp

object

win

1. pick\_up sword 9. go s
2. equip sword 10. go s
3. go s 11. go s
4. chop 12. go s
5. go n 13. go s
6. go w 14. go s
7. talk oldman 15. portal
8. go e



HTC sense 3

API level 10

— MAC address 00:1c:ff:b7:72:e3

Android 2.3.5

memory 768 MB., Gingerbread

— use Google location (Disable under location&security and under Search.)

1. actor

Actor

2. bag

3. container

4. default-weapon

5. environment

6. evil\_lair

25. weapon

7. exit

26. Wizard → Merlin.

8. food

27. ignitere

9. game

28. badform

10. game\_commands

29. ob\_game.

11. human

30. doxygen.conf

12. inside

13. key

> @make

14. light\_saber

> idz\_game

15. mapfile

16. object

17. outisde

18. player

19. room

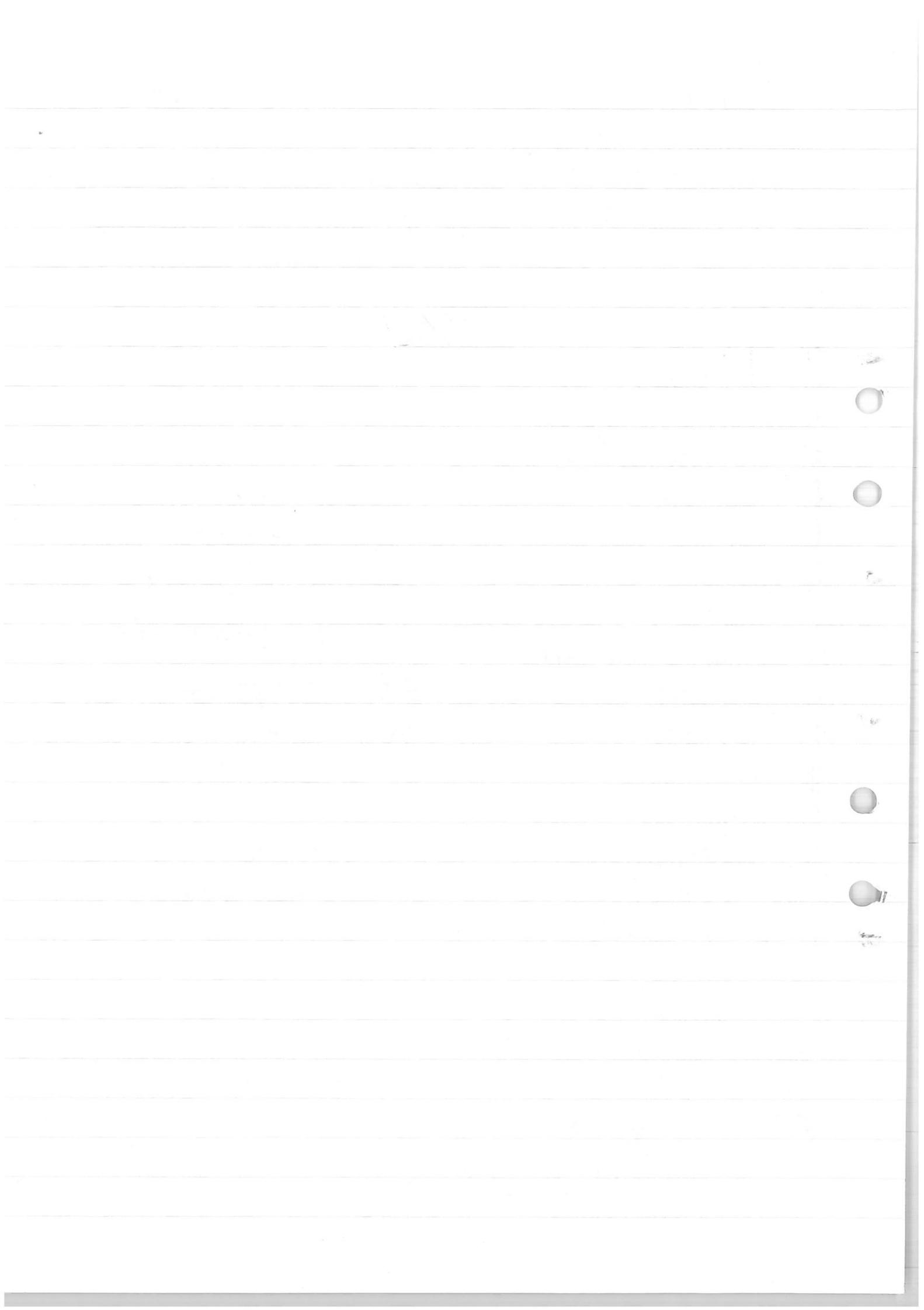
20. terminal

21. troll

22. vampire

23. vampire\_factory

24. wand : stick, have magic properties.



get\_object  
 $\{ \text{player} \rightarrow \text{current\_room} \rightarrow \text{objects}, \text{stringToInt id\_obj} \}$   
 push\_back( get\_object( player  $\rightarrow$  container  $\rightarrow$  get\_object(), id ) )

Objs is vector of shared ptr.

objs  
 words  
 id from parser  
 pick-up + OK  
 container has ① object  
 use + east  $\rightarrow$  invalid read

96	210	96 Bytes	496
29	85	29 Bytes	349
0	0	257 "	357
480	480	3,056 "	721

pick-up (string)

Objs.push\_back( get\_object( , id ) )

const "vec  
 Bag::add( Object &object )  
 eraser()  $\rightarrow$  Environment::pick-up() 198  
~~delete &object;~~  
 clear\_grammar() at the end of run\_game(),  
 Valgrind result

Note: Manual memory management through new and delete is considered BAD PROGRAMMING. Practice in Modern C++ because it easily leads to memory leaks or undefined behaviour, and negatively affects the design of your program in terms of robustness, readability and ease of maintenance.

- Dynamically allocated objects exist until they are freed  
Functions that return pointers to dynamic memory put a burden on their callers - the caller must remember to **DELETE** the memory.

- caller is responsible for deleting this memory

void use\_factory (T args)

    Foo \* p = factory (args)

    " use p but do not delete it

    " p goes out of scope, but the memory

    " to which p points is not freed!

When use\_factory returns, the local variable p is destroyed.

    FOO \* f = new Foo;  
    unique\_ptr<up> (f)

    Bag::add (obj & ob)  
    {  
    }

### - polymorphic dispatch

```
std::vector<shared_ptr<Fruit>> basket;
std::shared_ptr<Apple> p-apple (new Apple);
basket.push_back (p-apple);
```

when you call push\_back, the objects are stored as std::shared\_ptr<Fruit>  
not std::shared\_ptr<Apple>



grep const \*.cpp

grep const \*.h

cat \*.cpp | wc

valgrind --tool=memcheck --leak-check=yes .\equation

- ⑨ game constructor take away loop main from it
- ⑩ valgrind

Leak summary:

definitely lost: 24 bytes in 1 blocks

indirectly lost: 0 " " 0 "

possibly lost: 257 " " 9 "

still reachable: 1,904 " " 51 "



# Logic adventure game (text base game)

- Inventory management screen

- Text adventure, also known as Interactive Fiction, convey the game's story through passages of text, revealed to the player in response to typed instructions. Text-adventure use a simple verb-noun parser to interpret these instructions, allowing the player to interact with objects at a basic level, for example by typing "get key" or "open door".

## Parser

- What is the parser?

- It is a block of code (function) that takes a command entered by the player, interprets it, and then execute the command, it can be very lengthy. Effort can be made to reduce its core size by proceduralizing it somewhat (other, smaller sub functions).
- The parser must recognize "words".

github / ecksun.

Actor: Merlin, Dragon, Wizard, Morgan, Human

Human

Evans

Actor: Human(Merlin, wizardMorgan, player), Dragon, Arthur

environ: Inside(room, villain), outside(woods)

Camelot castle

object: container(bag), weapon/sword, lightsaber

vector<object\*> \* objects;

vector<Actor\*> \* actors;

map<string, environment> neighbors;

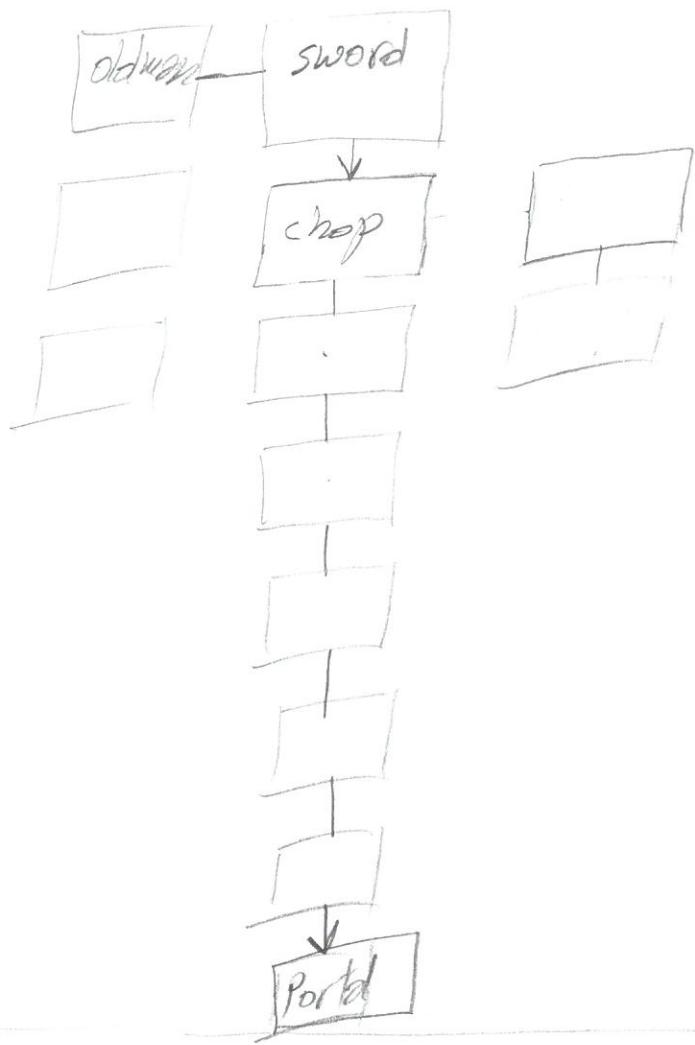
map<string, Exit\*> exits;

} get\_objects()

} get\_actors()

Dungeon-World (sageit) GitHub





> make  
> game



struct less than

```
{ bool operator() (Tx, Ty)  
{ return x < y; } }
```

y;

| Functor :  
| (Functionobject)  
| or object some numbers  
| syntaxes has no fct.  
|  
|

savefile

```
if (! save_name.empty()) {  
    save ("saves/" + save_name);  
    std::cout << "Game saved." << std::endl;  
} else {  
    std::cout << "Failed to save game,  
    please use \" save name \". "  
    << std::endl;
```

```
save (std::string filename)  
{ ofstream file (filename);  
    map.save (file); }
```

(3.1)

Some

Types 20

to avoid circular dependency

Direction Map Loader controller object actor place

Actor: wolf (crazywolf), Human (oldman, player)

object: Firewood, Backpack, container, key, weapon (sword)

place: outdoorPlace (House, woods, portal)

Map

{maps/mop.txt}

Direction / Directionset

controller

loader.

Exit Terminal

ECK

game  
game\_commands

object

actor

Environment

Container  
Food  
key  
bog

Food

weapon

key

vampire  
Human  
vampire\_troll  
vampire\_factory

Inside

Room

evil Lair

outside

Actor: vampire, vampirefactory, troll, Human (wizard, player)

Environment: Inside (room, evillair), outside

Object: container (box), food, key

game:

game\_commands

exit

Terminal

weapons: lightsaber, sword, defontiongun

Actor  
Dragon  
Human  
Wizard  
Monster

Actor \* opp = get\_actor (player → current\_room → actors,  
std::string s  
[ ] (std::vector<Actor> &actors, int id)  
actor { int n = -4711;  
std::istringstream stream(s);  
if (!stream >> n)  
throw -4711;  
return n; } );