TDDE71 – Programmering och datastrukturer

Projektintroduktion och standardbiblioteket

Klas Arvidsson, Eric Ekström

Ursprungliga slides av Oskar Holmström

Institutionen för datavetenskap

LINKÖPINGS UNIVERSITET

Allmän info

…

- Onsdag kl 13-15
  - Möjlighet redovisa labbar
  - Prata projektidéer och krav med assistent
  - Skaffa de andra 5 medlemmarna i din grupp
  - Meka med SFML-exempel

Agenda

1. Projektintroduktion
2. Makefile

LINKÖPINGS UNIVERSITET

Vad ska ni göra i projektet?

Projektintroduktion

- Ni får göra en applikation eller ett spel
- Vi rekommenderar att ni gör ett spel
  - Fyller upp kraven för projektet (Om på rätt nivå)
  - Är otroligt kul att göra och visa upp
- Vad är rätt nivå på projekt?
  - Sikta på 2D plattformsspel à la ”Super Mario”
  - Prata med oss så hittar vi en godkänd nivå.

Worms – space ops

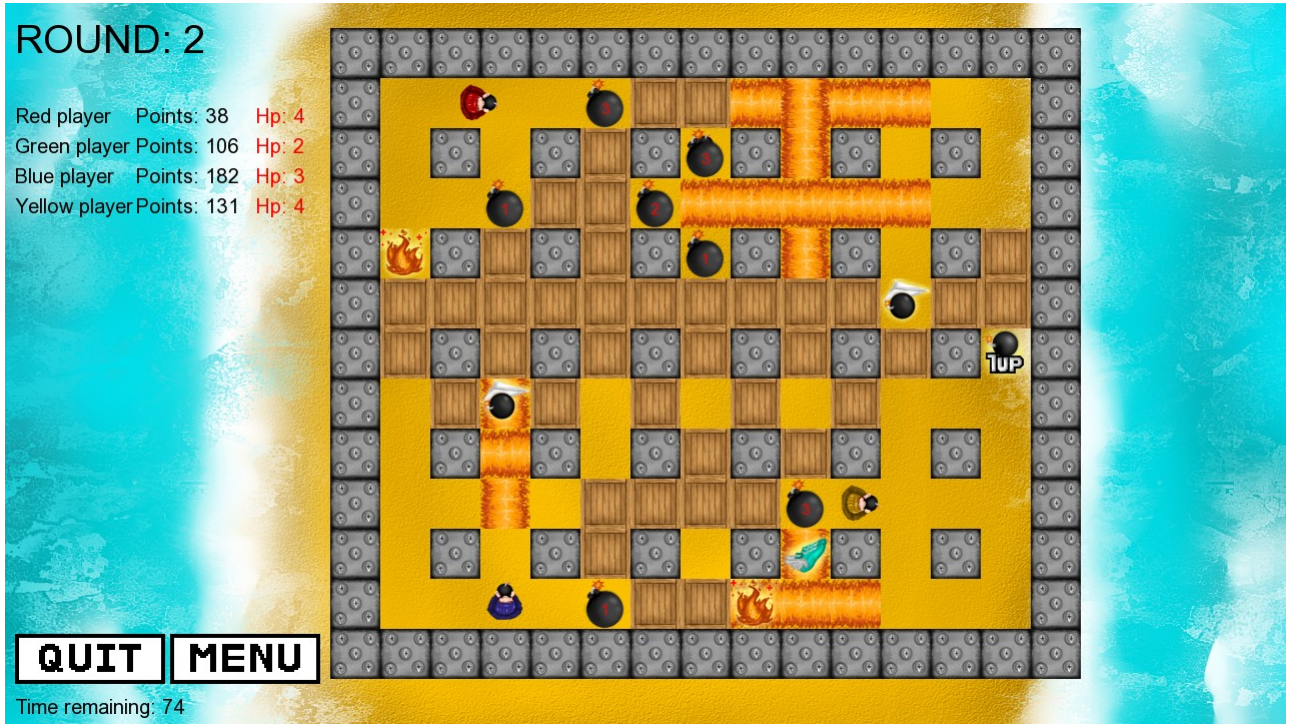
Exempel från tidigare år

- Worms - Space Ops är ett utom(j)ordentligt turordningsbaserat strategispel som tar dig till helt nya dimensioner! Använd dina vapen, krök rumtiden och besegra dina fiender. Sikta mot stjärnorna!

Playing with fire

Exempel från tidigare år

- Efter att vi fått ett anonymt tips om att ingen annan gjort en något som ens påminner om detta spel tänkte vi att vi skulle ta oss an denna stora utmaning. Spelet går ut på att släppa bomber och sedan undvika att bli träffad av dem då de exploderar. Spelaren får poäng genom att skada de andra spelarna, genom att ta powerups, och genom överleva till slutet av en omgång. Det finns powerups som gör att spelaren kan gå snabbare, släppa fler och större bomber samt putta bomber. Eftersom ens kompisar inte alltid är tillgängliga har vi skapat tre olika AI spelare. Dessa använder använder sig av avancerade topphemliga AI algoritmer som kan vara skadliga. Därför har vi beslutat att inte ge stöd för att spela online.

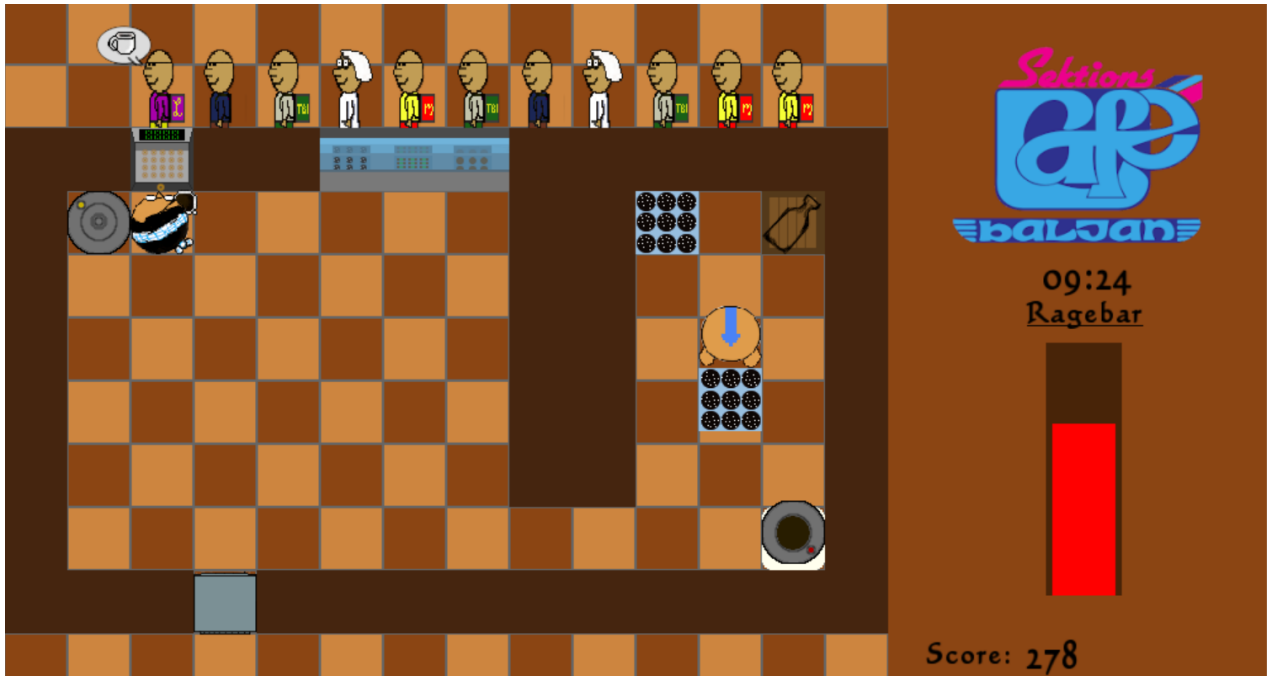


Baljan the Game

Exempel från tidigare år

- Att fyll på klägg, passa upp kunder och koka kaffe kanske låter enkelt. Men baljans kö är hetsig, klarar du hela dagen?
  Vi garanterar äkta Baljan-känsla!

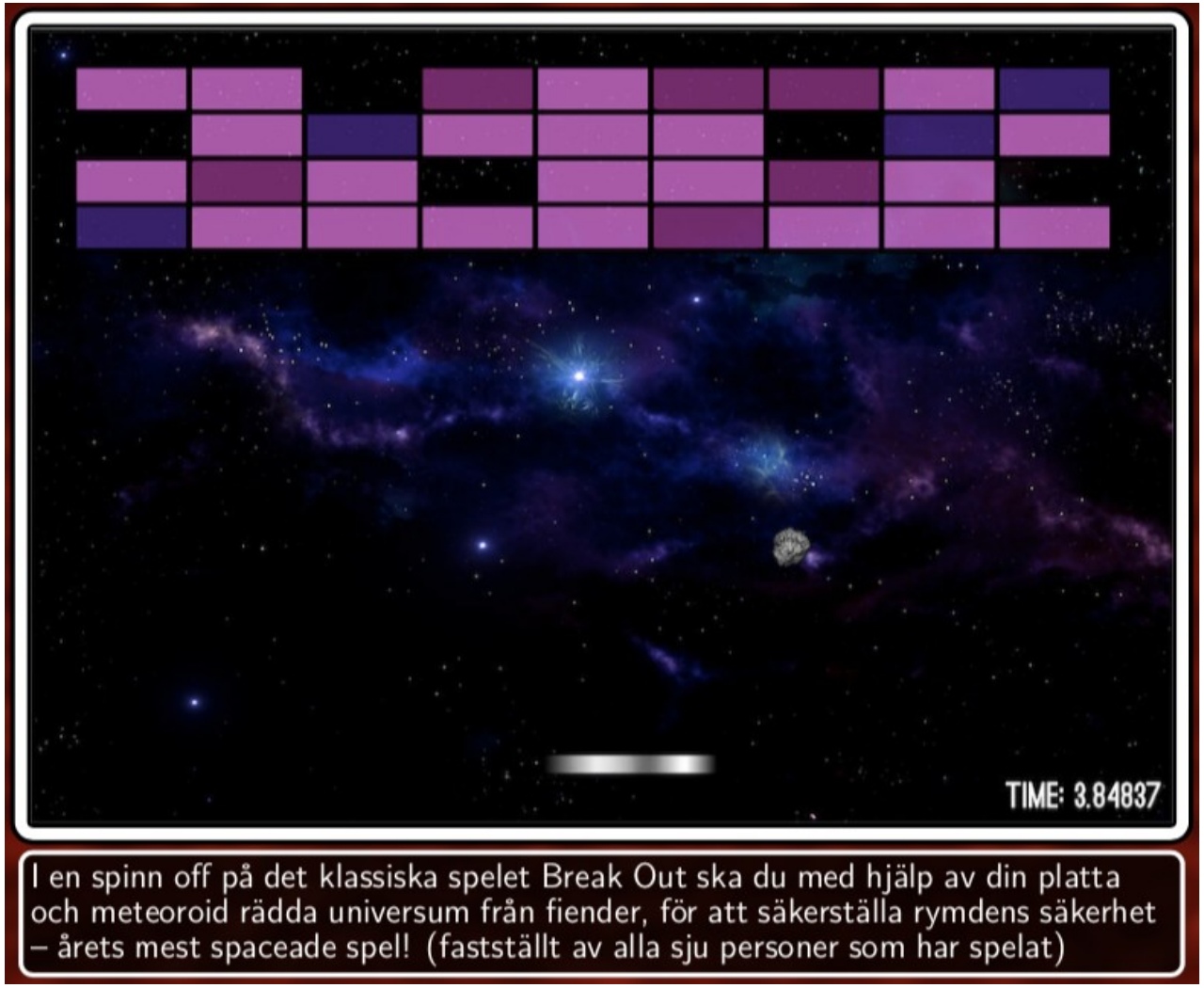
#baljanthegame



Space out

Exempel från tidigare år

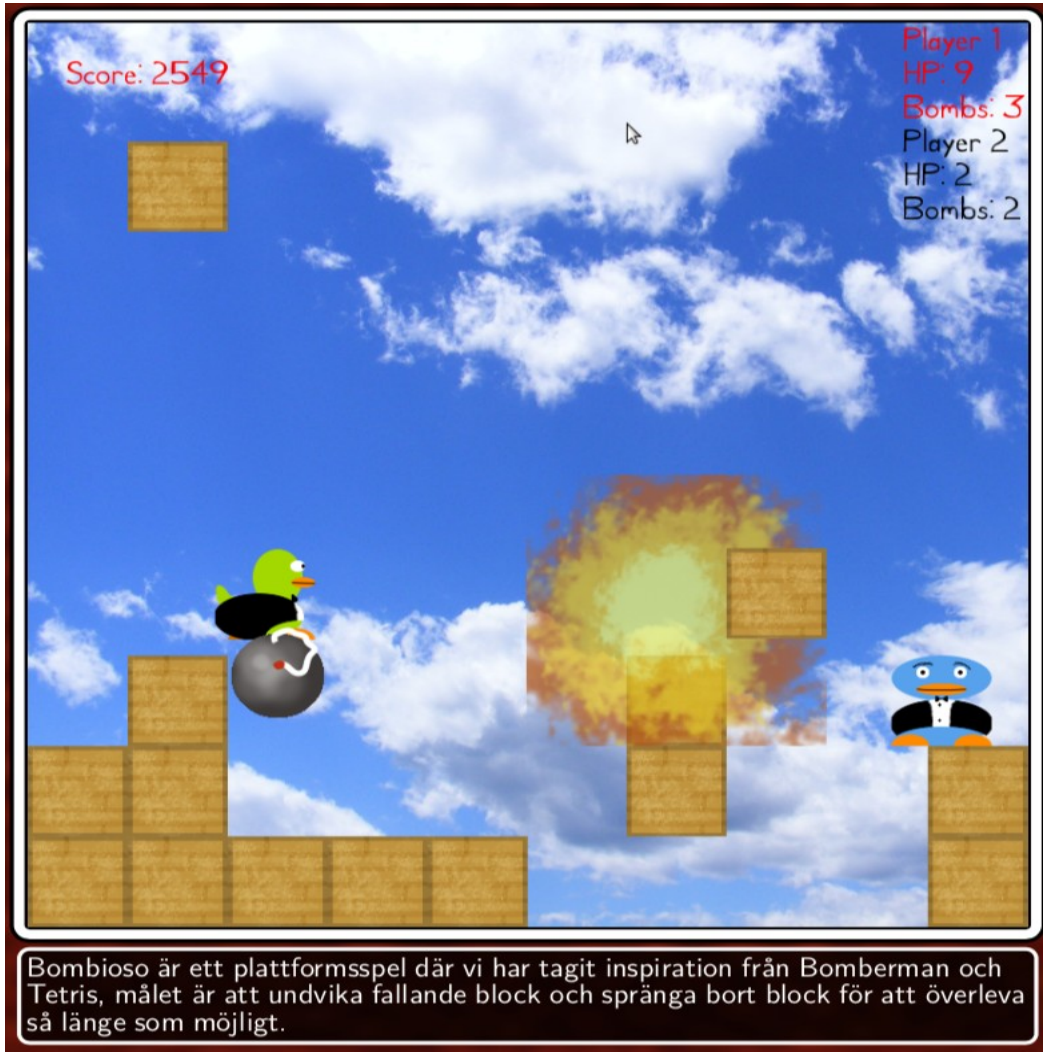
- I en spin-off på det klassiska BreakOut ska du med hjälp av din platta och meteorid rädda universum från fiender och säkerställa rymdens säkerhet – årets mest spacade spel! (fastställt av alla sju personer som spelat)



Bombioso

Exempel från tidigare år

- Bombioso är ett plattformsspel där vi tagit inspiration från Bomberman och Tetris. Målet är att undvika fallande block och spränga bort block för att överleva så länge som möjligt.



Personal space invaders

Exempel från tidigare år

- Simulera din dagliga färd upp för märkesbacken!
  Skjut ned festeriernas skmliga föslag med bestämda ”Nej”.
  Träffa enhörningen för extra liv på din färd mot att, nyktert, plugga hårt och få höga betyg.

Krav på projekt, godkänd nivå

Projektintroduktion

- Du har lagt ca 40h på programmering av valt projekt. Resultatet är commits i git av dig. Commitsen ska bestå av programkod i cc-filer och ska användas i det färdiga programmet. Programkoden ska vara utan koduppprepning och väl formaterad.
- Varje projektmedlem har lagt ca 40h på övriga projektuppgifter (dokument, kvalitetskontroll, möten, administration, git, make, research, etc).
- Ett färdigt projekt brukar bestå av ca 3000 rader välskriven kod och ca två dussin klasser.

Projektgrupper

Projektinformation

- En projektgrupp ska bestå av 6 personer
  - Ev. undantag anvisas av oss
- Tänk igenom din ambition:
  - Vill du göra något speciellt/häftigt? Hitta likasinnade.
  - Har du en spelidé? Hitta andra som är taggade på samma idé
- Registrera er grupp i webreg

Hur mycket får vi ”låna”?

Projektintroduktion

- Projektet ska inkludera ”credits.txt” där ni listar referenser/weburl till allt material ni använt för att skapa ert spel, hur ni använt det, och i vilka filer ni använt materialet. Det kan vara:
  - Tutorials
  - Dokumentetion
  - Referensmanualer
  - Kodexempel
  - Andra publika projekt
  - Bilder
  - Musik
  - Frågor till LLM (”AI”), url och komplett log
- Tänk på att en del material inte får användas fritt utan godkännande från upphovsrättsinnehavare.
- Även om ni använder kod ni inte skriver själva gäller att var och en ska skriva EGEN kod motsvarande ca 40h arbete.
- Spelmotorn SKA vara skriven av projektmedlemmarna själva.

Hur ni kontaktar er assistent

Projektinformation

- Projektgruppen ansvarar för att söka och hålla kontakt med sin assistent
  - För diskussion av lösningsalternativ gruppen väger mellan
  - För bokning av obligatoriska avstämningar och bonusredovisningar
  - Om konflikter eller irritation uppstår inom gruppen
  - Om tekniska problem tillstöter
  - Allt annat…

Hur ni kontaktar er assistent

Projektinformation

To: [er assistent]

CC: LiU-ID@student.liu.se för samtliga projektmedlemmar

Subject: TDDE71-Grupp-XX: [Kort ärendebeskrivning]

Content:

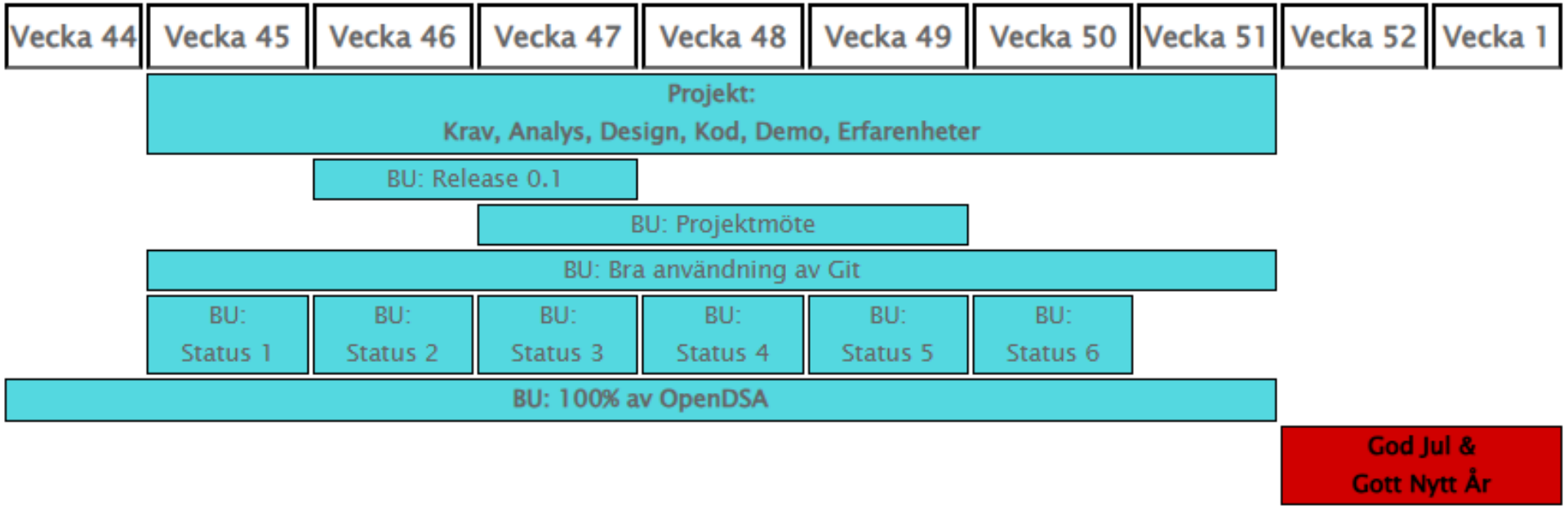
En tydlig beskrivning av ärendet.

Vid problem beskrivs vad som är problemet, vilka steg som utförs för att återskapa problemet, vad ni gjort för att försöka lösa problemet, och vad ni hoppas assistenten ska kunna bidra med.

Tidslinje från kurshemsidan

Projektintroduktion

TentaP + HT2



Kravspecifikation

Projektinformation

- Kravspecifikation är det första som utförs i projektet
- En beskrivning av *vad* spelet innebär (inte *hur* det ska implementeras)
  - De krav er kund ställer på ert spel
  - Vision av vad spelet ska vara när det är klart
- Består av:
  - Beskrivning av spelets handling och regler, text och lo-fi figurer
  - Beskrivning av användargränssnitt, menyer och interaktion med spelet
  - Manuellt numrerad lista med kravpunkter i minst tre nivåer (absolut, bör, kanske)
  - Saker som vi normalt förväntar ingår men som ni *inte* kommer utföra (avgränsningar)

Kravspecifikation

Projektinformation

- Ska beskriva hela spelapplikationen, inte bara spelets mål. Det ingår menyer, navigation, hur tangentbord och mus används, etc…
- Beskrivningen ska vara på en nivå så att även en person som aldrig spelat ett spel ska förstå vad det handlar om.
- ”Concept art”: Figurer är mycket beskrivande. Rita/skissa gärna någon typ av seriestripp. Nivån att fota av skissen är ok, det behöver inte vara snygga figurer ritade på dator, det är idén som är viktig att få fram tydligt med ett minimum av spenderad tid!

Kravspecifikation

Projektinformation

- Se till att få med krav på lite häftigare saker ni *inte* vet hur ni ska lösa. Vi hjälper till med tips på rimlig implementation.
- Se till att varje gruppmedlem har hittat på ett antal egna krav på saker som ska finnas i spelet och hur de ska bete sig.
- Alla krav ska relatera till programvaran ni ska skapa. Krav kursen ställer på ert projektarbete ingår inte.

Kravspecifikation

Projektinformation

- Ett krav ska vara
  - endast ett krav.
  - tydligt beskrivet.
  - mätbart.
  - realistiskt.
- Alla krav tillsammans ska ge en fullständig beskrivning av spelet
- Du ska kort och koncist kunna instruera en testperson hur hen kontrollerar att kravet är uppfyllt.

Projektintroduktion

Kravspecifikation

- Ask LLM: How do I formulate a good requirement list in a software requirement specification?

Objektorienterad analys

Projektinformation

- Tisdag 11:e november kl 10-12
  - Seminare för Objekt-Orienterad Analys (OOA)
- Input:
  - Färdig kravspecifikation
- Output: Spelets konstruktionsritning:
  - Klassdiagram
  - Klassbeskrivningar
  - Användningsfall/scenarion

OOA – erfarenhet från tidigare års grupper

Projektinformation

- Glöm inte att OOA ska täcka hela spelapplikationen, dvs även menyer och annat.
- Grupper med en väl utförd OOA som uppdaterats efterhand är tacksamma för den tid de lagt för att bygga en gemensam förståelse för klasserna.
- Grupper med en slarvig OOA uttrycker att de borde lagt mer tid på klassdiagram för att kunnat undvika många timmars strul till följd av skilda uppfattningar om klassdesignen inom gruppen.

Användning av Git

Projektinformation

- Git ska användas av alla i gruppen och genom hela projektet
  - Vi har inget annat sätt att bedöma om du har varit delaktig i utförandet av projektet.
  - Varje medlem ska ha commits som visar att de implementerat klasser och faktiskt skrivit programkod
- Vid inlämning skickas ett mail att ni är färdiga med projektet och en länk till ert git-projekt
- Allt som är i projektet vid inlämning kommer att bedömas
  - Städa i ert repository så inlämningen blir en ren release utan gamla saker som inte längre används, det underlättar för oss och ger mycket bättre intryck…

Användning av Git

Projektinformation

- Senaste version av spelet ska vid var tid ligga i main-branchen
- Varje projektmedlem ska ha ETT användarnamn (LiuID) i git – se till att varje konto du använder har samma config
- Vid commit ska samtliga medlemmar som gjort väsentliga bidrag till committen taggas i commitmeddelandet
- Varje vecka erhåller gruppen en summering av gruppens git-användning. Om summeringen inte stämmer ska gruppen kommentera detta i statusrapport till sin handledare.

Användning av Git

Projektinformation

- Ert git-repositorys main-branch ska innehålla:
  - Projektdokumentationen (gruppkontrakt, krav, ooa, design, erfarenheter, credits.txt, mötesprotokoll, etc)
  - All programkod som behövs för kompilering (Makefile, h-filer, cc-filer)
  - All media/data som behövs för att köra spelet (datafiler, bilder, musik). Vill ni undvika stora filer i git kan ni låta Makefile ladda ned er media från extern url via curl eller wget (fråga).
- Ert repository ska INTE innehålla (gitignore):
  - Filer som enkelt kan genereras eller laddas ned med automatik (*.o, a.out).
  - Filer som kan råka skapas av olika verktyg (*~, *.gch).

Halvtidsavstämning

Projektinformation

- Gruppen har som ansvar att boka in en halvtids-avstämning med sin assistent
- Ett tillfälle att tidigt fånga upp om projektet är på rätt spår för att bli godkänt
- En sista chans åtgärda strul inom gruppen – alla i gruppen har ett ansvar skapa förutsättningar för att varje medlem ska vilja och kunna arbeta sig till godkänt

Redovisning

Projektinformation

- Den 17/12 finns det ett schemalagt redovisningspass
- Alla tar en dator och så får vi gå runt och testa varandras spel – det bästa på hela terminen!
- Er assistent kommer ställa frågor till var och en angående er del i arbetet och efter det kan ni skicka in ert projekt
- Behöver/vill ni redovisa tidigare så brukar det gå att lösa
  - Hela gruppen behöver planera för detta från start
  - Boka in med er assistent minst två veckor i förväg

Makefile och kodinlämning

Projektinformation

- Assistenten ska kunna hämta ned ert projekt, skriva make och köra igång spelet
  - Lös det från början så ni själva kan dra nytta av det!
- Ni får använder er av make eller CMake i projektet – upp till er!

```
To: Er assistent
CC: LiU-epost för samtliga projektmedlemmar
Subject: TDDE71-XX-YyyZz: Kodinlämning
Content: [Ska som minst innehålla en länk
till ert projekt på Gitlab.]
```

Erfarenhetsrapport

Projektinformtion

- Tillfälle att reflektera vad ni gjort i projektet och vad ni har lärt er
- Omfattning: 8 erfarenheter beskrivna på ca en halv A4 styck ger ca 4 sidor.
- Gemensam och individuell självutvärdering
- Skriv kontinuerligt i samband med statusrapporter!

Vad är en erfarenhet?

Projektinformation

- Vad gjorde ni? Beskriv det utförligt. Gör det konkret.
- Vad blev resultatet? Beskriv det utförligt. Det kan vara både positivt och negativt.
- Vad tar ni med er? Vad skulle ni vilja ändra på? Hur?

- En oerfaren programmerare ska kunna förstå hur hen kan upprepa era goda erfarenheter och undvika de dåliga!

Tips

Projektinformation

- Bygg ert projekt steg för steg, ett krav i taget, MVP https://en.wikipedia.org/wiki/Minimum_viable_product
- Planera tidigt för det ni tror blir svårast att lösa, undersök lösningar och be om råd medan resten av implementationen lätt anpassas för att göra det svåra möjligt
  - Speciellt minneshantering, när objekt ska tas bort med delete, behöver planeras från början.
- Ta tag i problem omgående, kvarstår ett problem efter 3 dagar, lyft det med assistent (även problem med kommunikation/engagemang/närvaro)

Tips

Projektinformation

- Utse ansvarig för olika delar, någon som:
  1. ansvarar för att git-repot är städat och att alla medlemmar vet hur de ska använda git och att var och en bidrar med egna commits
  2. ansvarar för att möten har en agenda, var och en kommer till tals, och beslut dokumenteras där alla hittar dem
  3. har koll på vilka uppgifter var och en ska arbeta på, följer upp hur det går, och hjälper till undanröja hinder för framsteg
  4. kontinuerligt samlar in erfarenheter från var och en och skriver ihop veckas statusrapport
  5. ansvarar för den övergripande arkitekturen, hur spelmotorn ska fungera och ser till att var och en förstår den
  6. ansvarar för att gruppkontrakt följs och att gruppen en har en god stämning, ger beröm för utfört arbete och lyfter stämningen

Ansvar och delegering

Projektinformation

- Att vara ansvarig betyder:
  - Du ser till att arbetet blir klart i tid
  - Du ser till att arbetet har användbar kvalité
  - Du bär hundhuvudet om ovan inte är uppfyllt
- Att vara ansvarig betyder *inte*:
  - Att det är du som utför arbetet
- Att delegera:
  - Du sätter någon annan att göra jobbet
  - Du gör avstämningar så ofta att du ser att kvalité och deadlines kommer mötas, och så arbetet hinner styras upp om så ej ser ut ske

Dokumentera

Projektinformation

- När problem uppstår, t.ex. att en uppgift inte blivit utförd, är det ofta svårt att kräva bättring om det bara finns muntlig överenskommelse om vad som skulle utföras och när.
- Skriv ned i protokoll:
  - Vilka uppgifter som prioriteras
  - Vem som ska utföra respektive uppgift
  - Vem som kontrollerar kvalitén på utfört arbete
  - När uppgiften ska vara klar
- Se även till att var och vet hur hen ska agera om uppgiften är för svår eller tar längre tid än väntat

Detta är historien om fyra människor som hette Alla, Någon, Vem som Helst och Ingen.

Det fanns nämligen en viktig uppgift som måste utföras och Alla var säker på att Någon skulle göra det. Vem som Helst skulle ju kunna ha gjort det, men Ingen gjorde det.

Någon blev arg, eftersom det var Allas jobb.

Alla trodde att Vem som Helst skulle kunna göra det, men Ingen insåg att Alla inte skulle göra det.

Det hela slutade med att Alla anklagade Någon när Ingen gjorde vad Vem som Helst skulle kunna ha gjort.

Bonusuppgifter

Projektinformation

- Hur mycket poäng går det att samla för projektet?

Bonusuppgifter

Projektinformation

- Hur mycket poäng går det att samla för projektet? **16 poäng!**

Bonusuppgifter

Projektinformation

- 16 bonuspoäng totalt
- Vissa är individuella och vissa är poäng för hela gruppen
- Detaljer på kurshemsidan!

Projektstomme i form av release 0.1 (4p)

Bonusuppgift

- Poäng till hela gruppen
- Ska genomföras inom projektets första 2 veckor
- Publiceras på Git
- Stomme för alla klasser kompilerar och någon liten del av projektet fungerar. Exempel: en rektangel som representerar spelaren kan flyttas runt och flera blinkande cirklar som representerar fiender kan placeras ut.
- Ska ge hela gruppen en känsla för projektets stomme
- Ska vara god grund att bygga vidare på, tänk ”spelmotor”

Leda projektmöte med assistent (2p)

Bonusuppgift

- Poäng till alla i gruppen som kan tänka sig att hålla i projektmötet (Slump vem som håller i det)
- Mötet ska visa att projektet rör sig framåt
- Agenda ska sättas ihop och skickas till assistenten
- Genomgång av arbetet som har varit, problem som uppstått, vad har ni gjort för att lösa dem och arbetet som kommer
- Om det inte blir godkänt får ni boka in för ett nytt möte senare i projektet

Bra användning av Git (4p)

Bonusuppgift

- Individuella bonuspoäng
- Ska vara aktiv genom HELA projektet
- Ska utföra arbete på branches som sedan merges till master
- Tydlig namngivning av branches (efter feature eller person)
- ALLA commit-meddelande ska vara tydligt
  - Det ska berätta vad du tillfört i koden med din commit.
- Välstädat repo – enbart relevant kod och god ordning på filer

OBS! Detta är utöver git-användningen som krävs för godkänt

Veckolig statusrapport (1p gånger 6)

Bonusuppgift

- Poäng till hela gruppen, en poäng för varje godkänd rapport
- Kortfattad beskrivning av
  - Arbetet den senaste veckan
  - Problem och utmaningar som uppstått
  - Vad har ni lärt er från veckan
  - Plan för kommande veckan
- Användbart för erfarenhetsrapporten!

SFML – Simple and Fast Multimedia Library

Projektinformation

- Det verktyg vi använder för att skapa grafik
- Varför SFML?
  - Enkelt
  - Vi kan ge support
  - Dokumentation: https://www.sfml-dev.org/tutorials/2.5/
  - Exempel: https://gitlab.liu.se/tddi82-material/sfml-exempel
- Fritt att använda andra verktyg – vi kan däremot inte garantera hjälp med dem
- Nästa föreläsning ger mycket av release 0.1!

Agenda

1. Projektintroduktion
2. Makefile

LINKÖPINGS UNIVERSITET

Space invaders: Många programfiler

Vad är problemet?

```
enemy.cc, enemy.h
asteriod.cc, asteroid.h
ship.cc, ship.h
game_main.cc
test_enemy.cc
test_asteroid.cc
test_ship.cc
test_main.cc
```

Space invaders: Många programfiler

Vad är problemet?

```
enemy.cc, enemy.h
asteriod.cc, asteroid.h
ship.cc, ship.h
game_main.cc
test_enemy.cc
test_asteroid.cc
test_ship.cc
test_main.cc
```

- Vilka kompileringsflaggor ska användas?
- Hur ska varje fil kompileras?
- Hur ska testerna kompileras?
- Hur ska spelet kompileras?
- Måste allt kompileras om när något ändras?

Makefile

Lösningen när det är jobbigt att kompilera!

- `make` är ett verktyg för att hantera kompilering av många filer
- Vi skapar en fil: `Makefile`
  - Innehåller regler för hur filer beror av varandra
  - Innehåller regler för hur allt ska kompileras
  - Kan användas för att automatisera mer än kompilering
- Kommandot `make` kör vår Makefile och de kommandon vi skapat i den för att kompilera filer
- Kommandot `make` genererar automatiskt om varje fil som är äldre än någon av filerna den beror på

Makefile

Grundstruktur för en GNU Makefile

```
VARIABEL=värde
VAR=$(VARIABEL)


fil_att_skapa: filer som behövs
tab--->| kommando 1 hur filen ska skapas
tab--->| kommando 2 hur filen ska skapas


# Kommentar: filen döps “Makefile”
```

Makefile

Exempel

```
enemy.cc, enemy.h, asteriod.cc, asteroid.h
ship.cc, ship.h, game_main.cc
```

Makefile

```
ship.o: ship.cc ship.h
tab--->| g++ -c ship.cc


a.out: ship.o enemy.o asteroids.o game_main.o
tab--->| g++ ship.o enemy.o asteroids.o game_main.o
```

Makefile

Exempel – underhållsvänligare med variabler – här färgkodat!

```
CC=g++
CCFLAGS="-std=c++17 -Wall -Wextra"
LDFLAGS="-lsfml-window -lsfml-graphics -lsfml-system"
OBJS="ship.o enemy.o asteroids.o game_main.o"

game: $(OBJS)
tab--->| $(CC) $(LDFLAGS) $^ -o $@

%.o: %.cc %.h
tab--->| $(CC) $(CCFLAGS) -c $<
```

Makefile

Exempel – aktivera makefile regler

```
make


make game


make ship.o


make clean


```

Kompilering

Flaggor bra att känna till

- Varna vid alla former av fel

```
g++ -std=c++17 -Wall –Wextra –Weffc++ –Woverloaded-virtual –Wold-style-cast
```

  - studenter missar regelbundet tentor och får kompletteringar på sina laborationer för fel de enkelt kunnat upptäcka genom att använda kompileringsvarningar

- Säg som det är lite i taget (varningar blir fel, men bara tre i taget)

```
–Werror –fmax-errors=3
```

- Kompilera bara filen, länka inte ihop programmet (filen main.o genereras)

```
g++ -c main.cc
```

- Döp om den fil som genereras (programet heter nu main i stället för a.out)

```
g++ main.cc –o main
```

- Generera extra information för debugger (och andra verktyg som valgrind)

```
g++ -g main.cc
```

Slut för idag

…

- Resterande slides för självstudier vid behov

Agenda

1. Projektintroduktion
2. C++ standardbibliotek
   - 3.1 Databehållare
   - 3.2 Iteratorer (och auto)
   - 3.3 Algoritmer
   - 3.4 Lambda-funktioner
   - 3.5 Smarta pekare
   - 3.6 Slumptal

LINKÖPINGS UNIVERSITET

Databehållare i STL (Standard Template Library)

Terminologi på kommande bilder

T är en valfri datatyp/klass

t är en variabel eller ett objekt av typ T

K och **k** är som T och t men för söknyckel "key"

V och **v** är som T och t men för sökt värde "value"

N är ett heltal känt *vid kompilering*

n är ett heltal känt under programkörning

it är en *iterator*

Fullständig översikt:

https://en.cppreference.com/w/cpp/container

std::array

Databehållare i STL (Standard Template Library)

```
// Representerar en sekvens lagrade värden
// Statisk – kan ej ändra storlek
// Random access

#include <array>
std::array<T, N> b{};
t = b.at(index); // åtkomst, index börjar på 0
b.at(index) = t; // ändra, index börjar på 0
```

std::vector

Databehållare i STL (Standard Template Library)

```
// Representerar en sekvens lagrade värden
// Effektiv insättning sist
// Random access

#include <vector>
std::vector<T> b(n); // skapa med storlek n
std::vector<T> b{n}; // skapa med storlek 1 och värde n
b.push_back(t); // sätt in sist
t = b.at(index); // åtkomst, index börjar på 0
b.at(index) = t; // ändra, index börjar på 0
```

std::list

Databehållare i STL (Standard Template Library)

```
// Representerar en sekvens lagrade värden
// Effektiv sekventiell åtkomst
// Effektiv insättning på aktuell position

#include <list>
std::list<T> b{};
b.push_back(t); // sätt in sist
b.push_front(t); // sätt in först
```

std::set

Databehållare i STL (Standard Template Library)

```
// Representerar en uppsättning unika värden
// Effektiv sökning och insätting

#include <set>
std::set<K> b{};
b.insert(k); // sätt in
it = b.find(k); // sökning
b.count(k); // antal matchande värden (1 eller 0)
```

std::map

Databehållare i STL (Standard Template Library)

```
// Representerar en uppsättning unika nycklar med tillhörande värde
// Effektiv sökning och insätting

#include <map>
std::map<K,V> b{};
b[k] = v; // sätt in värde v för nyckel k
b[k] = v; // uppdatera värde för nyckel k (om ett värde redan fanns)
v = b[k]; // hämta aktuellt värde för nyckel k
v = b[k]; // sätt in värdet V{} för nyckel k (om inget värde fanns)
v = b.at(k); // hämta värde för nyckel k (undantag om värde saknas)
```

std::deque

Databehållare i STL (Standard Template Library)

```
// Representerar en sekvens lagrade värden
// Effektiv insättning först och sist

#include <deque>
std::deque<T> b{};
b.push_back(t); // sätt in sist
b.push_front(t); // sätt in sist
t = b.at(index); // åtkomst, index börjar på 0
b.at(index) = t; // ändra, index börjar på 0
```

std::queue

Adapterbehållare i STL (Standard Template Library)

```
// Representerar en FIFO kö (First In First Out)


#include <queue>
std::queue<T> b{};
b.push(t); // sätt in sist
b.pop(t); // ta bort först
t = b.front(); // titta först
t = b.back(); // titta sist
```

std::stack

Adapterbehållare i STL (Standard Template Library)

```
// Representerar en LIFO kö (Last In First Out)


#include <stack>
std::stack<T> b{};
b.push(t); // lägg överst
b.pop(t); // ta bort överst
t = b.top(); // titta överst
b.empty(); // är den tom?
```

Agenda

1. Projektintroduktion
2. C++ standardbibliotek
   - 3.1 Databehållare
   - 3.2 Iteratorer (och auto)
   - 3.3 Algoritmer
   - 3.4 Lambda-funktioner
   - 3.5 Smarta pekare
   - 3.6 Slumptal

LINKÖPINGS UNIVERSITET

Iteratorer

Vad är en iterator?

- En iterator i C++ är en datatyp som representerar en viss position i en datamängd.
- Alla iteratorer används på samma sätt, men varje typ av datastruktur har sin egen sorts iterator. Iteratorn har därmed inbyggd kännedom om hur den, utgående från nuvarande position, kommer till nästa position i just sin datastruktur.

Standard Template Library

Iterator

- En iterator representerar en viss plats i en behållare
- En iterator har operatorer för att
  - gå till nästa plats i sin behållare (++)
  - avgöra om den representerar samma plats som en annan iterator (!=)
  - hämta ut eller ändra värdet på sin plats (*, ->)
- Du kan be alla behållare om en iterator till första värdet (begin)
- Du kan be alla behållare om en iterator till ”efter slutet” (end)

Iteratorloop

Användning av iteratorer i STL

```
// T är valfri datatyp, container är valfri behållare
container<T> b{};
for ( container<T>::iterator it{begin(b)}; it != end(b); ++it )
{
  T& t{ *it };
  t = t + 1;  // ändra på platsen
  cout << t; // läs av platsen
}
```

Range based for

Användning av iteratorer i STL

```
// T är valfri datatyp, container är valfri behållare
container<T> b{};
for ( T& t : b )
{
  // Samma loop som på föregående sida
  t = t + 1;  // ändra på platsen
  cout << t; // läs av platsen
}
```

Automatisk typ från initialvärdet

Men fortfarande statisk typ som avgörs automatiskt vid kompilering

```
auto a; // kompileringfel
auto a{8}; // int
auto a{'5'}; // char

vector<double> v;
auto a{ begin(v) }; // vector<double>::iterator
```

Agenda

1. Projektintroduktion
2. C++ standardbibliotek
   - 3.1 Databehållare
   - 3.2 Iteratorer (och auto)
   - 3.3 Algoritmer
   - 3.4 Lambda-funktioner
   - 3.5 Smarta pekare
   - 3.6 Slumptal

LINKÖPINGS UNIVERSITET

Algoritmer i STL (Standard Template Library)

Terminologi på kommande bilder

begin är en iterator som anger början på ett intervall

end anger slutet på intervallet (ingår ej i intervallet)

destination är som *begin* med för resultat

operation är en funktion som utförs på varje värde

compare är en funktion som används för att jämföra värden

predicate är en funktion för att avgöra om ett värde ska påverkas av algoritmen eller inte

it är en iterator

https://en.cppreference.com/w/cpp/header/algorithm

std::sort

Algoritm i STL (Standard Template Library)

```
// Sorterar alla värden inom angivet intervall
// Det går att ange hur värdena ska jämföras

std::sort(begin, end);
std::sort(begin, end, compare);
```

std::transform

Algoritm i STL (Standard Template Library)

```
// Utför en operation på alla värden inom angivet
// interval och lägger resultatet i en annan
// behållare med start på angiven plats
// Du måste ange operationen som ska utföras

std::transform(begin, end, destination, operation);
```

std::copy

Algoritm i STL (Standard Template Library)

```
// Kopierar värden inom angivet interval till en
// annan behållare med start på angiven plats
// Du kan filtrera vilka värden som ska kopieras

std::copy(begin, end, destination);
std::copy_if(begin, end, destination, predicate);
```

std::shuffle

Algoritm i STL (Standard Template Library)

```
std::random_device rd{}; // långsamma men bra slumptal
std::mt19937 mt{rd()}; // snabba pseudo random tal med bra slumpfrö
std::uniform_int_distribution<int> dist{1,6}; // slumptal jämt fördelade inom intervall
```

```
// Blandar värden inom angivet interval
// utifrån angiven slumpkälla
std::shuffle(begin, end, mt);
```

std::max_element, std::min_element

Algoritm i STL (Standard Template Library)

```
// Hittar position för
// - största värde inom angivet intervall
// - minsta värde inom angivet intervall

it = std::max_element(begin, end);
it = std::min_element(begin, end);
if ( it != end ) // kontroll av resultat
  cout << *it; // hämta största/minsta värde
```

std::partition

Algoritm i STL (Standard Template Library)

```
// Placerar värden som uppfyller
// angivet kriterie i slutet av intervallet

it = std::partition(begin, end, predicate);
// Värden i intervallet [it, end[ förblir giltiga
```

std::remove_if

Algoritm i STL (Standard Template Library)

```
// Placerar värden som uppfyller
// angivet kriterie i slutet av intervallet

it = std::remove_if(begin, end, predicate);
b.erase(it, end); // tar bort ogiltiga värden
```

std::uniqe

Algoritm i STL (Standard Template Library)

```
// Förutsätter ett sorterat interval
// Placerar icke-unika värden i slutet av intervallet

it = std::unique(begin, end);
b.erase(it, end); // tar bort ogiltiga värden
```

std::accumulate

Algoritm i STL (Standard Template Library)

```
// Utför en accumulerad beräkning av angivet interval
// - acc är summa/product/konkatenering
// - start är initialvärde för acc
// - operation är hur accumuleringen ska utföras
#include <numeric>

acc = std::accumulate(begin, end, start, operation);
```

Agenda

1. Projektintroduktion
2. C++ standardbibliotek
   - 3.1 Databehållare
   - 3.2 Iteratorer (och auto)
   - 3.3 Algoritmer
   - 3.4 Lambda-funktioner
   - 3.5 Smarta pekare
   - 3.6 Slumptal

LINKÖPINGS UNIVERSITET

Standard Template Library

Lamba

- Du kan anpassa många algoritmer genom att beskriva den typ av jämförelse, operation, eller kontroll som ska ske på varje element
- Du skriver en lambda-funktion direkt som argument:

  ```
  [capture](parameters)->returntype { function body }
  ```

  (Du kan även använda vanliga funktioner som ”lambda”-argument genom att utelämna anropsparenteserna.)
- Parametrar och returtyp avgörs av aktuell algoritm, läs dokumentationen
- Capture låter dig specificera lokala variabler eller objekt du behöver nå inuti lambda-funktionen

Lambda-funktioner

Anonyma funktioner som implementeras on-demand

```
// syntax för lambda
[ capture_list ]( parameter_list )
 -> return_type
{
  // function body
}
```

Lambda-funktioner

Anonyma funktioner som implementeras on-demand

```
vector<int> v{4, 5, 3, 8};
int n{2};
transform(begin(v), end(v), begin(v),
          [n](int a)->int { return a*n; }
  );
// 8, 10, 6, 16
```

Agenda

STL Exempel (behållare, iteratorer, algoritmer, lambda)

LINKÖPINGS UNIVERSITET

Exempel

Ett program som använder STL

Fundera på och testa:

- Vad gör varje del?
- Vad åstadkommer programmet som helhet?
- Kan du göra samma sak enklare med färre behållare?

(Exemplet stävar efter att visa flera exempel på användning av STL. Det strävar inte efter att lösa problemet på ett optimalt sätt.)

Exempel

Ett program som använder STL

```
bool is_not_int(string const& s)
{
  return ! std::all_of(begin(s), end(s), [](char c)
                       {
                         return ('0' <= c && c <= '9');
                       });
}
```

Exempel

Ett program som använder STL

```
int main(int, char* argv[])
{
  const std::map<string, int> m{
    // m stores pair<string, int>
    {"ett", 1},
    {"två", 2},
    {"tre", 3},
    {"fyra", 4},
    {"fem", 5}
  };
```

Exempel

Ett program som använder STL

```
ifstream ifs{argv[1]};
string line;

while ( getline(ifs, line) )
{
```

Exempel

Ett program som använder STL

```
istringstream iss{line};
vector<string> l;

copy( istream_iterator<string>{iss},
      istream_iterator<string>{},
      back_inserter(l) );
```

Exempel

Ett program som använder STL

```
transform(begin(l), end(l),
          begin(l), [&m](std::string const& s)
          {
            auto it = m.find(s);
            if ( it != end(m) )
              // it "points to" a pair<string, int>
              return to_string(it->second);
            else
              return s;
          });
```

Exempel

Ett program som använder STL

```
vector<string>::iterator it = remove_if(begin(l), end(l), is_not_int);
l.erase(it, end(l));

vector<int> v(l.size());
transform(begin(l), end(l), begin(v), [](std::string const& s)
          {
            return stoi(s);
          });
```

Exempel

Ett program som använder STL

```
    int sum = accumulate(begin(v), end(v), 0, [](int sum, int i)
                         {
                           return sum + i;
                         });

    cout << sum << endl;
  }
  return 0;
}
```

Agenda

1. Projektintroduktion
2. C++ standardbibliotek
   - 3.1 Databehållare
   - 3.2 Iteratorer (och auto)
   - 3.3 Algoritmer
   - 3.4 Lambda-funktioner
   - 3.5 Smarta pekare
   - 3.6 Slumptal

LINKÖPINGS UNIVERSITET

Smarta pekare

Tidigare råpekare, nu smarta: std::unique_ptr

- Råpekare är vad vi arbetat med tidigare
  - ```
    int * x{};
    Node* n{};
    ```
  - Medför att vi behöver hålla koll på vem som har adressen för pekaren och hur det allokerade minnet ska destrueras
  - Problem när två eller flera pekare har samma adress:
    - Vilken ska destrueras?
    - Ska förändring av utpekad data påverka alla?
    ```

Smarta pekare

Tidigare råpekare, nu smarta: std::unique_ptr

- Smarta pekare (unique_ptr) löser det åt oss
  - Introducerar ägarskap – smartpekar-objektet äger pekaren vi ger det
  - Kopiering förbjuden – ger kompileringsfel!
  - Flytt från en smartpekare till en annan görs med `std::move`
  - När smartpekar-objektet går ur scope avallokeras pekaren
  - Istället för `Elem* e{ new Elem{7} };`
    och istället för `unique_ptr<Elem> e { new Elem{7} };`
    skriver vi `unique_ptr<Elem> e {make_unique<Elem>(7)};`
- https://en.cppreference.com/w/cpp/memory/unique_ptr

Agenda

1. Projektintroduktion
2. C++ standardbibliotek
   - 3.1 Databehållare
   - 3.2 Iteratorer (och auto)
   - 3.3 Algoritmer
   - 3.4 Lambda-funktioner
   - 3.5 Smarta pekare
   - 3.6 Slumptal

LINKÖPINGS UNIVERSITET

Slumptal

std::random_device, std::mt19937

För enstaka tal:

```
std::random_device rd; // create only one device!
std::uniform_int_distribution<int> die(1, 6);
std::out << ”Die rolled: ” << die(rd) << std::endl;
```

Effektivare för mång tal:

```
std::random_device rd; // create only one device!
int seed{ rd() }; // constant seed give same sequence
std::mt19937 mt{ seed }; // create only one mt19937!
std::out << ”Random integer: ” << mt() << std::endl;
```

Vill ni lära er mer om C++?

- Gå kursen TDDD38 – Avancerad programmering i C++

www.liu.se

LINKÖPINGS UNIVERSITET