

TDDE71 – Programmering och datastrukturer

Projektintroduktion och standardbiblioteket

Klas Arvidsson, Eric Ekström

Ursprungliga slides av Oskar Holmström

Institutionen för datavetenskap

Allmän info

Två mentifrågor

- Menti.com
- 5461 9490

Era svar på mentifråga 1

Kurspaus 23/10 - 3/11. Din deadline 28/10 förskjuts till efter paus. Vilket datum?

30

8/11

17 responses

57%

9/11

6 responses

20%

10/11

7 responses

23%

Rätt svar: 9/11 (23/8-28/8 är 6 dagar av pausen, 4/11-9/11 är 6 dagar efter pausen)

Ditt bästa tips för att förbereda duggan?

Era svar på menti-frågan

Exempeldugga	Be chatgpt ge fler exempelduggor	Fixa ett fett fuskblad Gör exempelduggan Du får ha VADSOMHELEST på fuskbladet	Skriv ner dugga lösningsförslag på fuskpapper
Gör exempelduggan (tidigare dugga)	Kopiera ut exempel duggan	FORMELBLAD	Jättebra formelblad
Göra testdugga	Samma	Ha hela duggakoden på papper	Copy pasta gamla duggan
Gör den gamla duggan	Stressa inte på duggan		
Fråga max	Lycka till!	Programmera mycket	Programmera
Chatgpt	Hitta på egna duggor och lös		
Skriv ner vanliga fel du brukar göra och hur du brukar lösa det.	Förbered dig		

Allmän info

Datastrukturer och algoritmer

- Glöm inte att datastrukturer och algoritmer ingår i kursen!
- 2p bonus för halvvägs klar 22/10 (121p)
- 2p bonus för helt klar 20/12 (241p)
- <https://www.ida.liu.se/~TDDE71/current/info/opensa.sv.shtml>

Allmän info

Kurspaus

- Kursen tar paus från och med 23/10 till och med 3/11
- Deadline för labkomplettering när återkoppling från oss erhålls under pausen infaller 11/11.
- Deadline som nyttjar +N dagar av pausen förskjuts till +N dagar efter pausens slut. Exempel: deadline 28/10 (+6) hamnar 9/11.
 - Detta tjänar även som exempel på en beräkning där många programmerare skulle gjort minst ett så kallat "off-by-one" fel.
- Epost till assistent kanske inte besvaras förrän efter pausens slut.
- Epost med kursledar-/examinatorsfråga besvaras som vanligt.

Allmän info

Redovisning av labbar

- Förvänta information från oss första veckan i HT2. Det är läge att då ha kvarvarande laborationer redo att redovisa.
- I avsaknad från erbjudanden från oss har du själv ansvar att boka in tid för redovisning. Kontakta assistenten för din webreg-grupp.
- Hård deadline att redovisa den 20/12
- Endast en komplettering per uppgift 20/12 – 10/1

Allmän info

Tips till nästa dugga

- Vad ska du göra när du kör fast?
 - Kontrollera allmänna meddelanden i klienten.
 - Skicka en fråga i klienten.
 - Titta på ditt (väl förberedda?) fuskblad.
 - Visa vad du kan:
Lös så mycket annat som möjligt i första hand.
 - Hitta på ett alternativt sätt att lösa det.

Allmän info

Tips till nästa dugga

- Hur ska du förbereda nästa dugga?
 - Öva själv utan hjälpmedel.
 - Skriv saker du behöver kolla upp på blivande fuskbladet.
 - Jämför din lösning med en kompis lösning, diskutera för och nackdelar.
 - Ta med din lösning till examinator och be om tips. Lösningen behöver inte vara komplett.
Epost funkar om du är introvert. Om du är extravert också faktiskt.
 - Hitta din egen strategi för att fokusera trots stress.
 - Utvärdera din tidigare strategi.

Allmän info

Duggaresultat

- Du har fått koden du lämnade in och de meddelanden som skickats i klienten.
- Det är viktigt att du går igenom din lösning och återkopplingen.
- Tycker du att du missat små saker? En liten justering och så fungerar allt? Kontakta examinator!
- Tänk på att vi under duggan bedömer "live" vilket innebär att vi denna gång gjort 168 bedömningar på ca 60 minuter. Att du går igenom din lösning och återkopplingen från oss är *minst* lika viktigt som det är vid en vanlig salstenta!

Allmän info

Tema nästa dugga (preliminärt)

- Tillståndsmaskin (state machine)

Agenda

- 1 Projektintroduktion
- 2 C++ standardbibliotek

Projektgrupper

Projektinformation

- Klarat duggan?
 - Du **ska** bilda grupp med andra som klarat duggan.
(Jämnare nivå i gruppen och *vi behöver inte ändra i gruppen senare.*)
- Kommer klara duggan den 8/11?
 - Du **ska** bilda grupp med andra som kommer klara duggan 8/11 (vilket vi utgår från att **alla** gör).
 - Gruppen ansvarar för att hjälpa varandra klara duggan.

Projektgrupper

Projektinformation

- Övning för projektgrupper som ej klarat duggan:
 - Examinator kommer publicera en duggaövningsuppgift
 - Var och en gör sin lösning från scratch (skriv upp saker som behöver slås upp)
 - Gruppen jämför och diskuterar alla lösningar tillsammans
 - Gruppen lämnar in en gemensam lösning till examinator för tips och återkoppling senast 6/11.
 - Högst rekommenderad övning, men frivilligt.

Projektgrupper

Projektinformation

- Strategier för projektgrupper med medlemmar som kommer klara duggan
 - Strategi 1: Skit samma, vi gör så lite som möjligt på projektet tills duggan är avklarad
 - Strategi 2: Vi gör övningen på föregående slide
 - Strategi 3: Vi bygger en bra spelmotorgrund redan före duggan som övning (se bonusuppgift "release 0.1")

Projektgrupper

Projektinformation

- Varje grupp ska bestå av 5-6 personer
 - Anmälan öppen i WebReg till och med 5/11
<https://www.ida.liu.se/webreg-beta/TDDE71-2024-1/PRA1>
 - Ni bildar själva grupp.
 - Vi förbehåller oss rätten att vid behov ändra i grupper:
 - Så att alla har en grupp när projektet drar igång.
 - Så att alla grupper är fyllda.
 - Se till att alla i gruppen har lika ambition och kunskap, det gör det lättare att diskutera och samarbeta.
 - **Skriv ett gruppkontrakt**
- Varje grupp har en assistent för stöd och återkoppling
 - Behöver inte vara samma assistent som i labserien

Projektintroduktion

- Vi går igenom projektet nu så ni är redo sätt igång från dag 1 i nästa läsperiod
- Projektet är det moment i kursen som upplevs som mest lärorikt och roligast
- Ni får möjligheten att använda er av allt ni lärt er i denna kursen
- Och mer!

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?
 - Prata med oss så hittar vi en godkänd nivå.

Krav på projekt, godkänd nivå

Projektintroduktion

- Se kurshemsidan för alla detaljer!
- Objekt som reagerar på användarindata (mus, tangentbord)
- Objekt som reagerar på förfluten tid
- Objekt som är stationära relativt spelkartan (eller motsvarande)
- Objekt som är rörliga relativt spelkartan (eller motsvarande)
- Flera olika beteenden på objekten ovan
- Kollisionshantering (eller motsvarande)
- Klasshierarki som *nyttjas* för att behandla alla objekt enhetligt

Exempel på nivå på projekt

Projektintroduktion

- Se kurshemsidan för alla detaljer!
- Godkänd nivå på spel:
 - 2D arkadspel: Pacman, Breakout, Space Invaders etc.
 - Kan behöva extra funktionalitet för att ge alla gruppmedlemmar tillräckligt mycket kod att producera.
- Ej godkända:
 - Tetris, Snake, Tre i rad, memory, etc.
 - Kan ibland utökas eller kombineras för att komma upp i godkänd nivå.
- Har du ett exempel? Vi hjälps åt resonera om kraven går nå!

Hur mycket får vi "låna"?

Projektintroduktion

- ALL programkod ni lämnar in ska vara skriven från scratch utan förlaga av gruppmedlem(ar)
 - ELLER
- Referera till den kod ni tittat på för att skriva er kod. Det gäller ALLA förlagor, även kod från "kursen". Används ChatGPT ska komplett log från den bifogas.
- Grafik och text får lånas fritt så länge ni inte hamnar i konflikt med ev. upphovsrätt.

Worms – space ops

Exempel från tidigare år

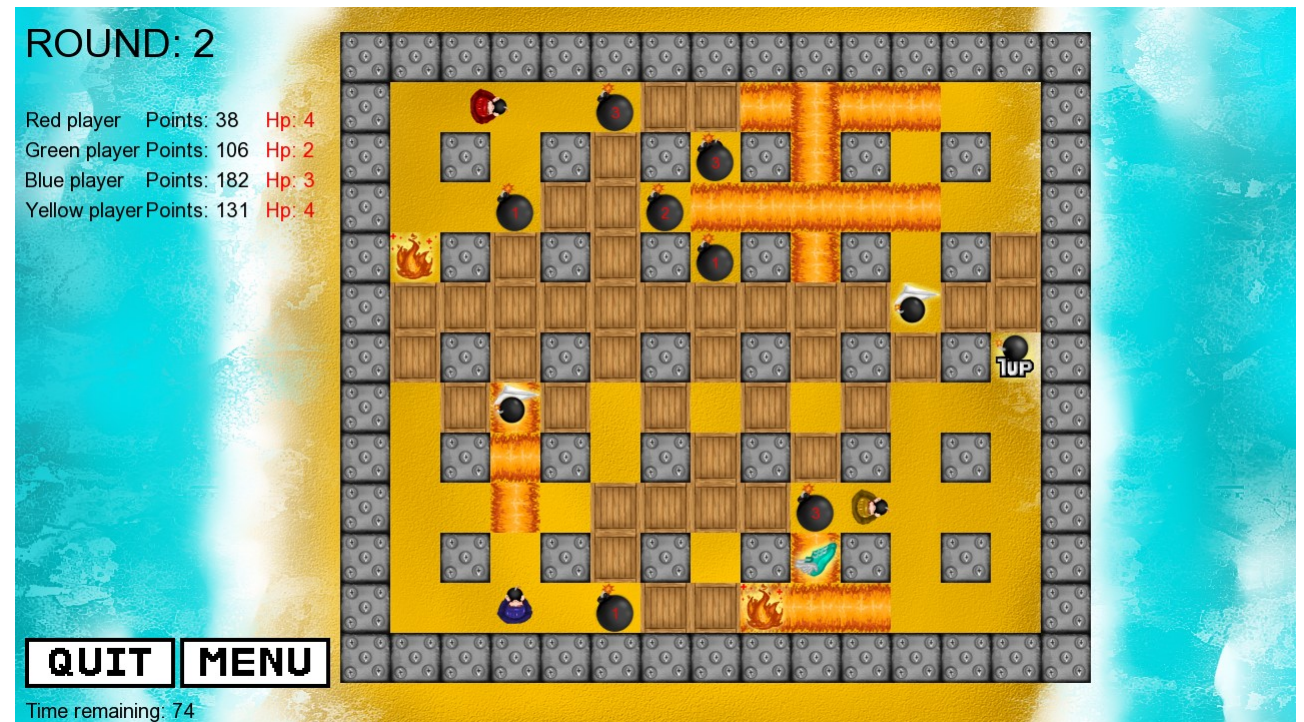
- Worms - Space Ops är ett utom(j)ordentligt turordningsbaserat strategispel som tar dig till helt nya dimensioner! Använd dina vapen, krök rumtiden och besegra dina fiender. Sikta mot stjärnorna!



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 avancerade topphemliga AI algoritmer som kan vara skadliga. Därför har vi beslutat att inte ge stöd för att spela online.

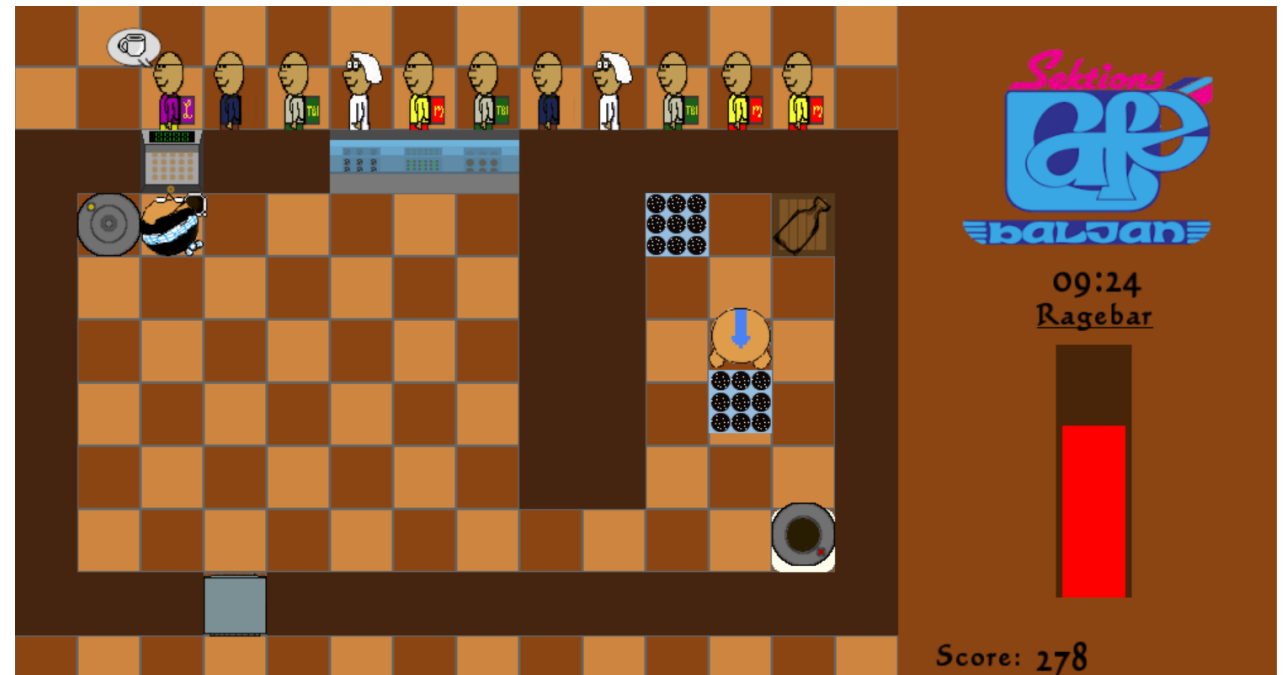


Baljan the Game

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!

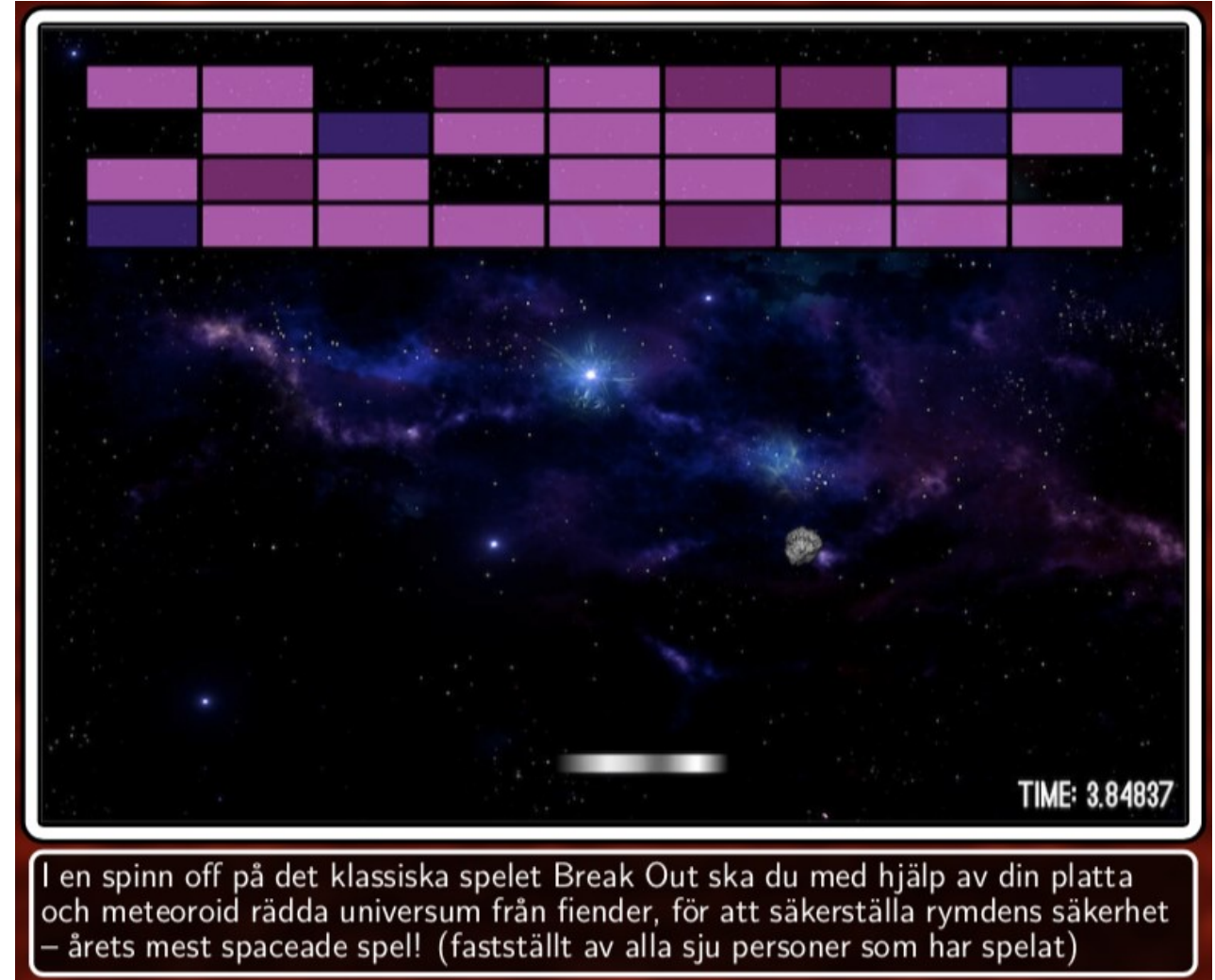
#baljanthegame



Space out

Exempel från tidigare år

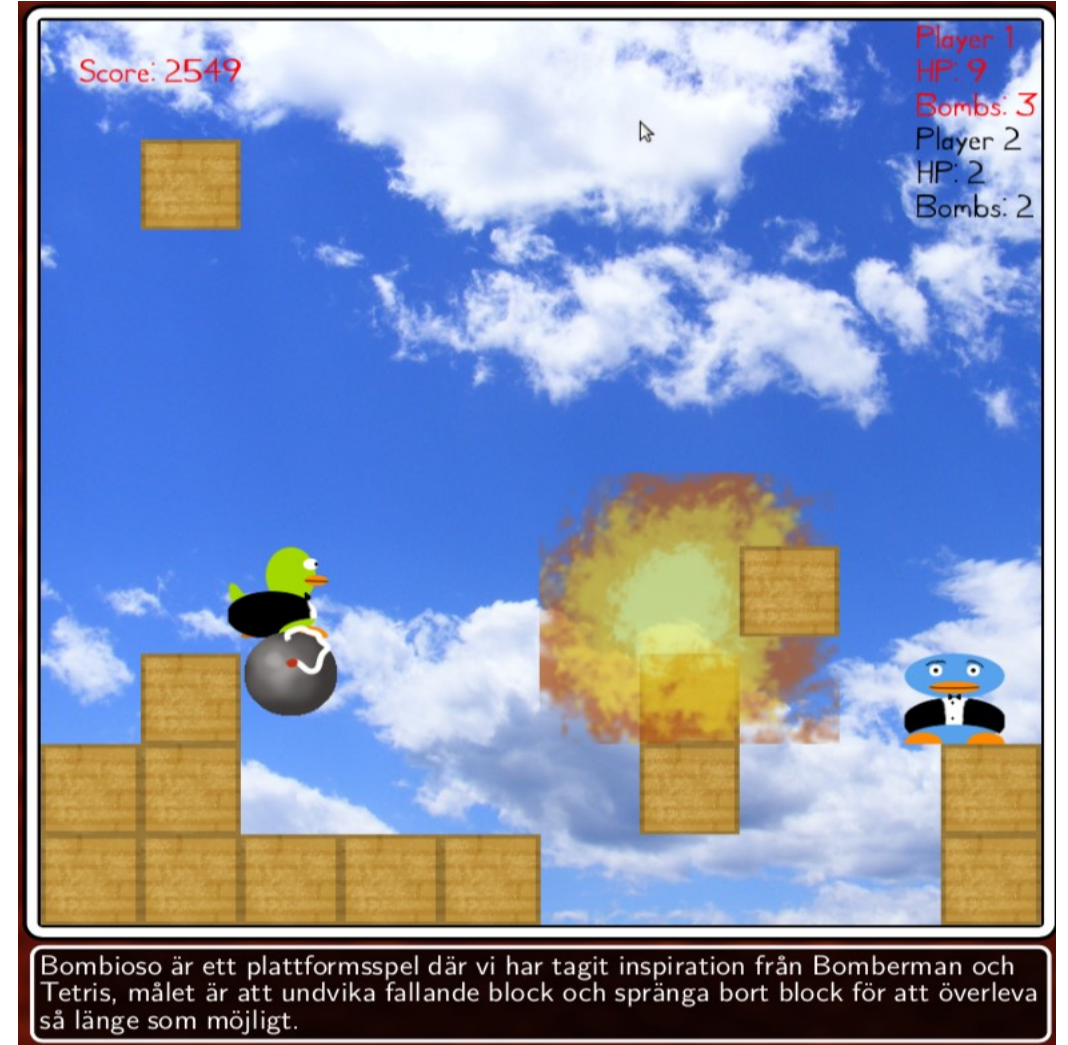
- I en spin-off på det klassiska BreakOut ska du med hjälp av din platta och meteoroid rädda universum från fiender och säkerställa rymdens säkerhet – årets mest spacade spel! (fastställt av alla sju personer som spelat)



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

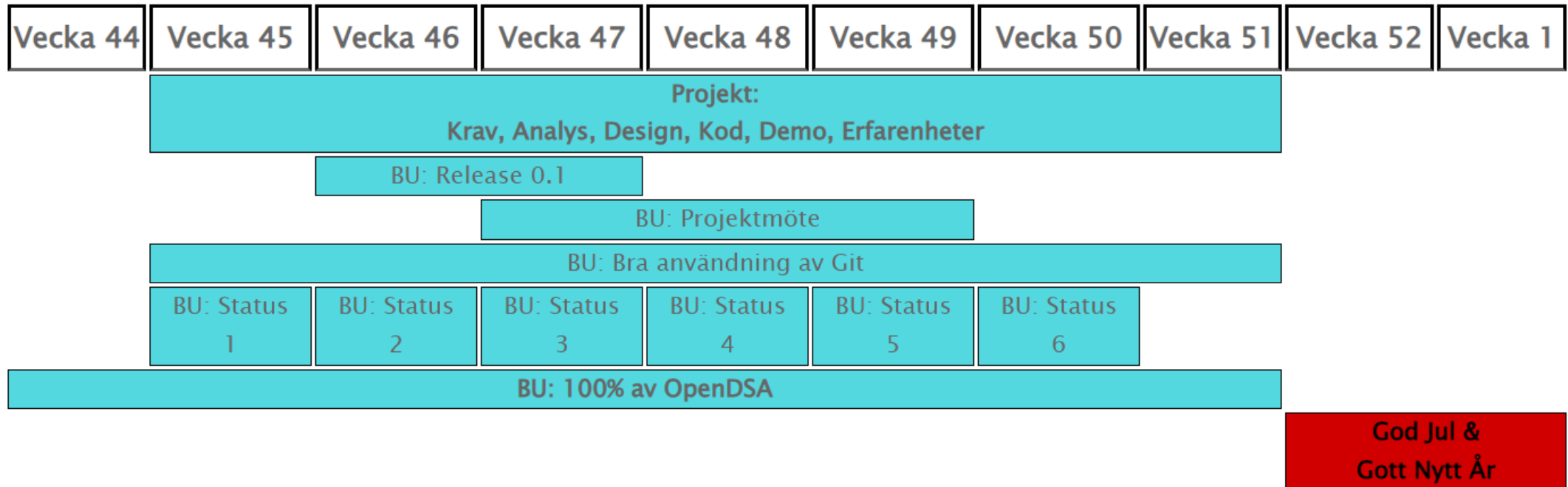
- Simulera din dagliga färd upp för märkesbacken!
Skjut ned festeriernas skamliga förslag med bestämda "Nej".
Träffa enhörningen för extra liv på din färd mot att, nyktert, plugga hårt och få höga betyg.



Tidslinje från kurshemsidan

Projektintroduktion

TentaP + HT2



Projektinformation

Fullständig information finns på kurshemsidan:

<https://www.ida.liu.se/~TDDE71/current/projekt/index.sv.shtml>

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...
- All kontakt ska följa ett anvisat mail-format och
 - Ni ska tydligt beskriva vad som är problemet och vad ni prövat
 - Vad ni hoppas assistenten ska kunna bidra med

Kravspecifikation

Projektinformation

- Kravspecifikation är det första som utförs i projektet
- En beskrivning av *vad* spelet innebär (inte *hur* det ska implementeras)
 - ”Vad ska spelet vara när det är klart?”
 - Begränsa er *inte* till vad ni tror ni kan implementera.
- Består bland annat av:
 - Beskrivning av spelet, text och bild
 - Användargränssnitt – interaktion med spelet
 - **Krav (Absolut, bör, kanske)**
- Ett ”kontrakt” mellan gruppen och kursledningen

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

- Ett krav ska vara
 - endast ett krav.
 - tydligt beskrivet.
 - mätbart.
 - realistiskt.
- Alla krav tillsammans ska ge en fullständig beskrivning av spelet

Projektintroduktion

Kravspecifikation

- Exempel på dåligt krav (för en tärning):
 - ”En tärning ska ha 6 stycken sidor och varje sida har ett antal pluppar”
- Exempel på ett bättre krav (för en tärning):
 - ”En tärning ska ha 6 stycken sidor.”
 - ”Varje sida på tärningen ska ha 1-6 markeringar”
 - ”En markering är i form av en fylld cirkel”
 - ”Markeringarna ska vara symmetriskt placerade”

Objektorienterad analys

Projektinformation

- Vi har gjort OOA på ett seminarie – Space Invaders
 - Nästa seminarie får ni möjlighet att göra en OOA för just ert projekt – med assistenter till hands
 - Viktigt att ha en god uppfattning om målet redo (kravspec)
- Arbetsprodukter:
 - 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...

Halvtidsavstämning

Projektinformation

- Gruppen har som ansvar att boka in en halvtidsavstämning 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 18/12 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 lite frågor och efter det kan ni skicka in ert projekt
- Behöver/vill man redovisa tidigare så brukar det gå att lösa
 - Kontakta er assistent i god tid innan

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 använda 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

Projektinformation

- 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
- 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

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 kompilarar 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 C++ standardbibliotek

Standardbiblioteket

Livekodning

- Vi använder standardbiblioteket för att bygga en poänglista
 - Containern `std::map`
 - `std::pair`
 - Containern `std::vector`
 - Iteratorer
 - Algoritmen `std::sort`
 - Jämförelsefunktion till `std::sort`
 - Borttagning ur container

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

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 insättning
```

```
#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 insättning

#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

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. Iteratoren 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

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 slumpstal  
std::mt19937 mt{rd()}; // snabba pseudo random tal med bra slumpfrö  
std::uniform_int_distribution<int> dist{1,6}; // slumpstal jämt fördelade inom intervall  
  
// Blandar värden inom angivet interval  
// utifrån angiven slumpkälla  
std::shuffle(begin, end, mt);
```

std::max_element, std::min_element

Algoritmer 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::unique

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 ackumulerad beräkning av angivet interval  
// - acc är summa/product/konkatenering  
// - start är initialvärde för acc  
// - operation är hur ackumuleringen 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

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)

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

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

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 många 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