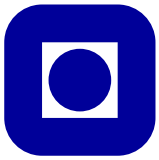

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2014

Øving 10

Frist: TBA

Mål for denne øvingen:

- Bruke det vi har lært i faget.
- Unntakshåndterings: «`try-catch`» blokker, kaste unntak (`throw` exceptions), «exception»-klasser

Generelle krav:

- bruk de eksakte navn og spesifikasjoner som er gitt i oppgaven.
- det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.
- skriv den nødvendige koden for å demonstrere implementasjonene dine

Anbefalt lesestoff:

- Kapittel 17 & 18, Absolute C++ (Walter Savitch)
- It's Learning notater

1 SafeArray (20%)

Tidligere i øvingsopplegget har vi lært at tabeller (arrays) i C++ har enkelte begrensninger i forhold til det vi kanskje er vant med fra andre programmeringsspråk, i og med at vi må passe på at vi ikke kan endre størrelsen på dem enkelt, og heller ikke enkelt slå dem sammen. I denne oppgaven skal vi derfor lage oss en klasse som oppfører seg litt mer som Python-lister, og se at vi kan få akkurat den funksjonaliteten vi vil, ved å bygge sammen konsepter vi har lært tidligere i øvingsopplegget.

Først noen generelle krav til klassen SafeArray:

1. SafeArray skal være en template-klasse slik at den kan holde alle datatyper.
2. Alle medlemsvariabler skal være private.
3. Det skal være mulig å lagre elementer i klassen.
4. Dataene skal lagres i en dynamisk tabell internt i klassen.
5. Det skal være mulig å finne ut hvor mange elementer det er plass til
6. Det skal være mulig å endre hvor mange elementer der er plass til
7. Dersom man reduserer størrelsen skal de siste elementene som det ikke er plass til, gå tapt.
8. Dersom man øker størrelsen, skal alle nye elementer ha verdi 0 (eller tilsvarende), mens alle gamle skal bevares.
9. Det skal være mulig å bruke []-operatoren til å plukke ut elementer
10. SafeArray skal ha konstruktører, slik at man kan opprette ei liste med N elementer, som alle er initialisert til 0.
11. SafeArray skal ha en destruktør som rydder opp i allokert minne.
12. SafeArray skal ha kopikonstruktør og operator= slik at vi trygt kan kopiere SafeArrays.
13. SafeArray skal kaste exceptions hvis man forsøker å utføre ulovlige handlinger (Dette vil bli spesifisert nøyere lengre ned).

Du står rimelig fritt til å velge hva du kaller medlemsvariabler, og medlemsfunksjoner, men **det forventes at samtlige krav testes og demonstreres i `main()`**. Under får du allikevel en kort oppskrift på rekkefølgen det kan være nyttig å angripe problemet i:

a) Opprett template-klassen `SafeArray` (Krav 1)

b) Legg til de medlemsvariablene du trenger for å oppfylle krav 2, 3 og 4

c) Skriv konstruktørene og destruktørene som trengs for å oppfylle krav 10, 11 og 12

d) Skriv de(n) medlemsfunksjonen(e) som trengs for å oppfylle krav 5

e) Skriv de(n) medlemsfunksjonen(e) som trengs for å oppfylle krav 9

f) Skriv testkode i `main()` som oppretter et SafeArray, fyller det med verdier, og skriver disse ut
Forsikre deg også om at du kan endre verdiene etter at de er lagt inn.

g) Skriv de(n) medlemsfunksjonen(e) som trengs for å oppfylle krav 6, 7 og 8 Utvid testene dine, slik at du forsikrer deg om at kravene er oppfyllt.
Hint: Du må lage nye tabeller, og kopiere innholdet

h) Utfør følgende test i main()

- Opprett et nytt SafeArray med plass til 10 elementer
- La disse elementene være tallene 0-9
- Gang hvert tall med posisjonen sin, slik at tabellen nå inneholder N*N på posisjon N.

- Skriv ut tabellen
- Lag et annet SafeArray som kopi av det første, ved hjelp av =.
- Reduser størrelsen til halvparten
- Legg til 5 til alle tallene som nå er i tabellen
- Øk størrelsen til 10 igjen
- Skriv ut begge tabellene, og bekreft at svaret stemmer med det du forventer.

2 FancySafeArray (20%)

I forrige oppgave la vi grunnarbeidet for å lage en litt mer praktisk tabell-implementasjon enn de enkle dynamiske tabellene C++ har i grunnlaget. Vi har allerede en tabell som kan lagre fritt valgte datatyper, og endre størrelse ettersom vi trenger det, men hva om vi har lyst til å gjøre livet vårt litt enklere?

Vi skal nå lage en subklasse `FancySafeArray` av SafeArray, som løser de forrige kravene, og noen nye:

1. FancySafeArray skal kunne inneholde de aller fleste datatyper (mildere krav enn for SafeArray)
2. Det skal være mulig å legge sammen to FancySafeArray med +, resultatet av å legge sammen to FancySafeArray (A + B), skal være et nytt FancySafeArray der alle elementene i A ligger først, og så alle elementene i B: (1, 2) + (3, 4) = (1, 2, 3, 4).
3. Det skal ikke være mulig for +-operatoren å endre på noen av de klassene den tar inn.
4. Det skal være mulig å skrive ut et FancySafeArray med <<-operatoren.
5. FancySafeArray skal kaste exceptions hvis man forsøker å utføre ulovlige handlinger (Dette vil bli spesifisert nøyere lengre ned).
6. FancySafeArray skal IKKE vaere friend med noe som helst

For testingen sin del, kan det være nyttig å gjenta testene som ble gjort for `SafeArray` for `FancySafeArray`.

a) Opprett klassen `FancySafeArray` som arver fra `SafeArray`

b) Skriv den koden som trengs for å oppfylle krav 4
Du står fritt til hvordan du vil formatere utskriften, men det kan være fordelaktig å skille elementer med et eller annet tegn (f.eks. komma eller mellomrom), og kanskje å ha en eller annen form for skilletegn rundt hele utskriften. (F.eks. parentes, slik at det blir lettere å skille utskrift av flere lister fra hverandre)

NB: Hvilken begrensning legger dette på hva slags elementer vi kan legge inn i tabellen vår? Forklar hvorfor krav 1 sier «de aller fleste» og ikke «alle datatyper» som i Oppgave 1.

c) Skriv de(n) medlemsfunksjon(ene) som trengs for å oppfylle krav 2 og 3
Sørg også for at dette testes i main().

NB: For å oppfylle krav 3, må du kanskje endre noen funksjonshoder i SafeArray også. *Hint: Du vil ha bruk for å kunne bruke [] både på const-SafeArrays, og ikke-const-utgaven*

3 Unntak/Exceptions (10%)

Vi har nå løst alle kravene i de foregående oppgavene, med unntak av det siste kravet om exceptions.

a) Et sted vi kan få problemer, er dersom vi forsøker å få tilgang til et element som ikke eksisterer, vi har definert at vi skal kunne aksessere elementer med [], som potensielt kan gi oss problemer hvis vi prøver å be om element 150 i et (Fancy)SafeArray med 10 elementer.
Endre implementasjonen av []-operatoren slik at den kaster en `std::out_of_range`-exception hvis man prøver å aksessere et element som ikke er i tabellen.
Utvid også koden i main() til å overpøve at dette faktisk skjer. (Du må med andre ord legge til try-catch).

b) Et annet potensielt problem er at man prøver å endre størrelsen på tabellen til en negativ størrelse, det kan vi ikke få til å gi mening, så det vil vi helst unngå.
Endre implementasjonen av funksjonen(e) som lar deg endre størrelsen på tabellen, slik at disse nå kaster en `std::out_of_range`-exception.
Finnes det en annen måte å løse dette på?

4 Sudoku (40%)

Sudoku er et spill der man skal forsøke å fylle inn tall på et brett, slik at hver rad, kolonne, og 3x3-boks på et 9x9 brett inneholder hvert av tallene 1-9 nøyaktig en gang. I denne oppgaven skal vi lage en Sudoku-løser.

For mer informasjon om Sudoku, og taktikker for å løse spillet, se følgende linker:
`http://www.sudokuwiki.org/sudoku.htm`
`http://en.wikipedia.org/wiki/Sudoku`

For å løse dette, trenger vi:

1. En klasse, som skal inneholde brettet, og funksjonene vi trenger for å løse det.
2. Funksjoner for å lese inn et brett fra fil
3. Funksjoner for å skrive resultatet til fil, eller skjerm.
4. Funksjoner som forsøker å løse brettet
5. Funksjoner som lar brukeren hjelpe løsningen på vei, når den setter seg fast.

a) Tenk over følgende:

- Hvordan vil du representere et Sudokubrett?
- Hvordan vil du representere en tom rute på et Sudokubrett?

b) Opprett en klasse til formålet over, med hensiktsmessige medlemsvariabler, og konstruktører/destruktører

c) Vi ønsker å kunne lese, og skrive sudokubrett fra fil:

- **Skriv en funksjon som skriver et (delvis) løst Sudokubrett til tekstfil**
  Hint: Er det en måte vi kan gjøre dette på slik at vi også kan bruke samme funksjon til å skrive brettet til skjerm?
- **Skriv en funksjon som leser et Sudokubrett fra tekstfil**

Siden mange taktikker går ut på å finne ut hvilke tall som er tilgjengelige i en gitt rute, kan det være hensiktsmessig å holde orden på dette, slik at vi ikke trenger å regne ut dette på nytt fra bunn igjen for hver rute. Til dette kan du bruke 9 `std::set` for kolonnene, 9 for radene, og 9 for boksene, slik at man har et `set` for hver. Da vil tilgjengelige tall i en gitt rute være gitt av snittet mellom settet for kolonna, rada, og boksen ruta er i. (Altså, de tallene som finnes i alle 3 settene).

d) Legg til passende set som beskrevet over i klassen, og sørg for at de initialiseres riktig i konstruktører

e) Avhengig av hvordan du har valgt å oppbevare settene for 3x3-boksene, så trenger du å vite hvilket set som tilhører en gitt rute, i praksis har du her to valg, du kan enten ha laget deg en tabell på 9 elementer av type `std::set` (og valgt en eller annen numerering av rutene), eller du kan ha laget deg en `std::set`[3][3]-tabell. I alle tilfeller trenger vi funksjonalitet for å finne riktig boks, gitt koordinatene til ei rute.
Skriv funksjon(er) som kan brukes for å finne en boks, gitt koordinatene til ei rute

f) Vi er også interessert i at settene skal bli oppdatert riktig når vi leser inn et brett:
Legg til en funksjon som gitt koordinater, og en verdi, setter inn verdien på brettet, og oppdaterer settene tilsvarende

g) Oppdater funksjonen som leser brettet fra fil, slik at denne bruker funksjonen fra forrige oppgave til å sette verdiene inn på brettet

h) Lag en funksjon som finner snittet av 3 sett av den typen du bruker til å holde orden på tilgjengelige tall

i) Lag en funksjon som tar inn koordinater og en verdi, og ved hjelp av settene, finner ut hvorvidt verdien er lov å plassere i ruta

j) Lag en funksjon som spør brukeren etter koordinater og en verdi, og forsøker å plassere denne verdien på brettet, hvis mulig, la brukeren fortsette å angi verdier til brukeren velger å ikke angi flere *Hint: Spør brukeren om han vil angi flere verdier*

k) Alle kan gjøre feil, det er derfor nyttig å la brukeren angre sine feil:

- **Legg til en måte å lagre alle trekk som blir gjort i klassen, og skriv om funksjonen fra f, slik at alle trekk blir notert ned**
- **Legg til funksjonalitet i funksjonen fra j, som lar brukeren angre et bestemt antall trekk**

l) Legg til kode i main slik at du kan spille, test spillet mot et sudokubrett, f.eks

						4		8
				1	9			2
			4	3			9	5
		6		5				9
	1	5	9				6	
	3				8	5		4
1					6			
		2		4				
4	6	8	5		3			

For å sjekke løsningen din, kan du bruke Web-soduku-løseren som det linkes til øverst i oppgaven, denne gir deg også mulighet til å få hint.

m) Skriv en funksjon som går gjennom hele brettet, og for hver rute, sjekker hvorvidt det bare er ett tall som kan plasseres i ruta, hvis så er tilfelle, skal tallet plasseres der

n) Skriv en funksjon som forsøker å løse brettet, ved å bruke funksjonen fra forrige oppgave så lenge den klarer å plassere tall. Når funksjonen stopper opp skal brukeren spørres etter input

Legger dette noen føring på returverdien til den forrige funksjonen vi skrev?

o) For å løse hele eksempelbrettet vårt, trenger vi litt mer funksjonalitet, vi trenger nemlig å kunne vite hvorvidt et tall kan plasseres et annet sted i samme rad, kollonne eller boks, i motsatt fall er tallet nemlig NØDT til å plasseres i den ruta vi ser på. (Altså: Dersom dette er det eneste stedet i enten raden, eller kolonnen, eller 3x3-boksen, må tallet være her).

- **Skriv de funksjonene du behøver for å finne ut hvor mange steder i en gitt rad et tall kan plasseres**
- **Skriv de funksjonene du behøver for å finne ut hvor mange steder i en gitt kolonne et tall kan plasseres**
- **Skriv de funksjonene du behøver for å finne ut hvor mange steder i en gitt boks et tall kan plasseres**

p) Sett sammen funksjonene fra forrige oppgave til en funksjon som sjekker om et tall bare kan plasseres i en gitt rute, og bruk denne funksjonen på samme måte som den andre løsningsfunksjonen vår

NB: Legg merke til forskjellen her, den forrige funksjonen fant ut hvorvidt det bare var ett tall som passet i en gitt rute, mens denne finner ut om ei rute er den eneste som et gitt tall kan plasseres i

Sjekk også at eksempelbrettet nå kan løses uten brukerhjelp.

5 Sudokuunntak (10%)

Vi er interessert i at Sudokubrettene våre ikke ender opp med å være uløselige, til dette kan vi bruke unntak for å si ifra når noe problematisk skjer

a) Lag en klasse `IllegalSudokuMoveException`, som arver fra `std::exception`
Klassen skal brukes til å kaste unntak dersom et trekk er ugyldig, som følge av at tallet allerede eksisterer i enten samme kolonne, rad, eller boks. Klassen skal kunne inneholde trekkets posisjon og verdi, samt en måte å si hvorvidt det er rad, kolonne, eller boks som skaper problemer for tallet. (Dersom det er flere årsaker, holder det med én av dem).

b) Lag en klasse `NumberAlreadyInPositionException`, som arver fra `std::exception`.
Denne skal brukes til å si ifra når vi forsøker å sette inn et tall i en rute som allerede har en verdi (ikke tom), og bør kunne inneholde koordinatene, og verdien som allerede er der.

c) Legg til kode i fillesningsfunksjonen din, som kaster et `std::runtime_error`-unntak dersom fila ikke lar seg åpne

d) Utvid koden fra 4f, slik at denne sjekker hvorvidt tallet er gyldig å plassere i ruta, i motsatt fall, skal det passende av de to nye unntakene våre kastes

e) Legg til kode som fanger opp de nødvendige unntakene følgende steder:

- I funksjonen som tar inn brukerinput, hvis mulig, gi brukeren en saklig tilbakemelding, og en ny sjanse.
- I main().

f) Test unntakene, ved å redigere brettet til å være ugyldig (sett f.eks. inn en 4-er i øvre venstre hjørne) for innlesing.