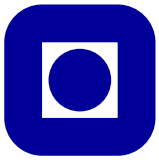

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2015

Øving 10

Frist: 2015-03-27

Mål for denne øvinga:

- Implementere beholdere (containers)
- "Template" funksjoner
- Standard template library (STL); iteratorer og beholdere (containers)

Generelle krav:

- bruk de eksakte navn og spesifikasjoner som er gitt i oppgava.
- det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.

Anbefalt lesestoff:

- Kapittel 16 & 19, Absolute C++ (Walter Savitch)
- It's Learning notater

For en god oversikt over STL, kan du se `http://www.cplusplus.com/reference/stl`

1 «Template» funksjoner (10%)

`Template`-funksjoner lar oss skrive generelle funksjoner som fungerer for forskjellige datatyper, uten at vi skal måtte lage egne separate implementasjoner for hver enkelt datatype. I denne oppgava skal vi skrive noen slike.

a) Skriv template-funksjonen `shuffle` som stokker elementene i en tabell (array), slik at rekkefølgen på elementene i tabellen blir tilfeldig. La tabellen som skal stokkes være den første parameteren og størrelsen på tabellen den andre. Du skal være i stand til å kompilere og kjøre den følgende koden som bruker `template`-funksjonen:

```
int a[] = {1, 2, 3, 4, 5, 6, 7};
shuffle(a, 7); // Resultat, rekkefølgen i a er endret.

double b[] = {1.2, 2.2, 3.2, 4.2};
shuffle(b, 4);

string c[] = {"one", "two", "three", "four"};
shuffle(c, 4); // Resultat, rekkefølgen i c er endret.
```

b) Skriv `template`-funksjonen `maximum` som tar to verdier av samme type som argument og returnerer den største verdien av de to.

Eksempel:

```
int a = 1;
int b = 2;
int c = maximum(a, b);
// c er naa 2.

double d = 2.4;
double e = 3.2;
double f = maximum(d,e);
// f er naa 3.2
```

c) Hvilke begrensinger gjelder for funksjonen du lage i b) (relatert til måten du implementerte den på)?

Funksjonen vil fungere for alle grunnleggende datatyper (int, char, double), men hvis du bruker argumenter av en brukerdefinert type, som en Person eller Circle klasse, vil programmet dit muligens ikke kompilere. Vær sikker på at du forstår hvorfor det er slik, og hva du må gjøre for å bruke denne funksjonen på dine egne klasser.

2 SimpleSet (10%)

I denne oppgavne skal du lage et sett. Et sett er en datastruktur som tillater følgende operasjoner:

- Insert - Sett inn et element in settet
- Remove - Fjern et element fra settet
- Exists - Sjekk om et element finnes i settet

Alle elementer i et sett er unike. Det vil si at om man prøver å legge inn et element som allerede eksisterer skal man ikke få lov til det. I dette tilfellet ønsker vi å lage et sett som inneholder heltall (`int`).

a) Implementer classen `SimpleSet`.

Denne klassedeklarasjonen kan være til hjelp når du skal implementere settet:

```
class SimpleSet{
    public:
        /** Construct empty set **/
        SimpleSet();
        /** Insert i into set, return true if the element was inserted, else false **/
        bool insert(int i);
        /** Returns true if i is in the set **/
        bool exists(int i);
        /** Removes i from the set, return true if an element was removed **/
        bool remove(int i);
    private:
        /** Dynamic array containing set elements **/
        int *data;
        /** Current number of elements **/
        int currentSize;
        /** Max capasity of data **/
        int maxSize;

        /** Returns the index where i may be found, else an invalid index. **/
        int find(int i);
        /** Resizes data, superflous elements are dropped. **/
        void resize(int n);
};
```

3 SimpleSetTemplate (10%)

I denne oppgava skal koden fra `SimpleSet` utvides til å benytte templates.

a) Utvid koden fra `SimpleSet` til å bruke `templates`.

Hint: `Templates` classen må kun bestå av en (header) fil. Dette er fordi kompilatoren må kjenne til hele innholdet av klassen (både deklarasjon og implementasjon) for å generere riktig type av klassen for hver gang den benyttes.

b) Kommenter i koden din dersom det er noen spesielle hensyn / krav til hvilke datatyper som kan bruke settet ditt.

4 Iteratorer (10%)

a) Lag en vektor for strenger (std::vector<string>) og legg inn fem-seks strenger i vektoren ved å bruke `push_back()`-funksjonen. Skriv ut innholdet i vektoren med en for-løkke som bruker `iteratorer` (og IKKE indeksoperatoren [])

Eksempel:

```
Lorem
Ipsum
Dolor
Sit
Amet
Consectetur
```

b) Bruk en `reverserende iterator` (reverse iterator) for å skrive ut innholdet i vektoren i motsatt rekkefølge.

Eksempel:

```
Consectetur
Amet
Sit
Dolor
Ipsum
Lorem
```

c) Skriv funksjonen `replace`. Funksjonen tar en `vector<string> referanse` og to strings (`old` og `replacement`)som argumenter. Funksjonen skal erstatte alle elementer i vektoren som er lik `old` med det man fikk inn som `replacement`. For å få til dette skal du bruke iteratorer og `erase()`- og `insert()`-funksjonene. Dersom det ikke finnes noen elementer lik `old` i vektoren, skal funksjonen selvfølgelig ikke endre noe som helst.

Bruk funksjonen på vektoren din og skriv den ut igjen.

Eksempel: Vektoren har i grunnlaget:

```
Lorem
Ipsum
Dolor
Lorem
```

Etter å ha kjørt:

```
replace(vektor, "Lorem", "Latin");
```

Ser vektoren slik ut:

```
Latin
Ipsum
Dolor
Latin
```

5 Lister (Lists) (10%)

a) Lag klassen `Person` med medlemsvariabler for fornavn og etternavn. Inkluder alle konstruktører, medlemsfunksjoner og overlagrede operatorer du mener er nyttige, inkludert en måte å skrive ut Person-objekter til skjermen.

b) Lag en `std::list<Person>` variabel og skriv en funksjon for å sette inn `Person`-objekter i sortert rekkefølge (basert på den alfabetiske rekkefølgen til navnene).

```
void insertOrdered(list<Person> &l, Person p);
```

Hint: strenger kan sammenlignes med operatorene $<$ og $>$. (Alfabetisk, slik at "ABCD" $<$ "BCDEF")

c) Lag en løkke i `main()` som skriver ut alle objektene i listen til skjermen.

6 «Sets» (10%)

a) Lag et `Set` som inneholder alle heltall fra 0 til 100

b) Fjern alle tall som er delbare med 2 (med unntak av tallet 2 selv) fra settet

c) For hvert tall mellom 2 og 50 gjør det samme som i forrige oppgave. Dersom 3 blir brukt som et eksempel skal 6, 9, 12 etc fjernes fra settet, men 3 beholdes.

d) Skriv ut alle verdiene i settet ved hjelp av en iterator

7 Lenkede lister (20%)

I denne øvinga skal du implementere en lenket liste (linked list). En lenket liste er en liste hvor hvert element, i tillegg til å inneholde data, peker til det neste elementet i listen. Bruk den gitte filen `"LList.h"`, eller kopier og lim den følgende koden, og implementer en lenket liste hvor hver *node* inneholder en *string* som data.

```
class ListNode {
private:
    std::string value;
    ListNode *next;
public:
    ListNode(const std::string& value);
    const std::string& getValue() const;
    ListNode* getNext() const;

    friend class LinkedList;
};

class LinkedList {
private:
    ListNode *head;
    ListNode *last;
public:
    LinkedList();
    ~LinkedList();

    bool isEmpty() const;
    void insertAtFront(const std::string & elem);
    void insertAtBack(const std::string & elem);
    bool removeFromFront(std::string & output);
    bool removeFromBack(std::string & output);

    friend std::ostream & operator<<(std::ostream & stream, const LinkedList & list );
};
```

Implementer følgende funksjoner og operator:

- `ListNode::ListNode(const string &value)`

  - initialiserer noden.
- `string ListNode::getValue() const`

  - returnerer verdien til noden.
- `ListNode *ListNode::getNext() const`

  - returnerer pekeren til den neste noden.
- `LinkedList::LinkedList()`

  - konstruktør som lager en ny liste med head og last satt til `NULL`.
- `LinkedList::~LinkedList()`

  - destruktør som sletter alle elementene i listen for å frigjøre minne.
- `bool LinkedList::isEmpty() const`

  - returnerer true hvis listen er tom.
- `void LinkedList::insertAtFront(const string &value)`

  - som legger til et element fremst i listen.
- `void LinkedList::insertAtBack(const string &value)`

  - som legger til et element bakerst i listen.
- `bool LinkedList::removeFromFront(string &value)`

  - som fjerner et element fremst i listen.
  Verdien til elementet som fjernes skal lagres i string-referansen som gies som argument.
- `bool LinkedList::removeFromBack(string &value)`

  - som fjerner et element bakerst i listen.
  Verdien til elementet som fjernes skal lagres i string-referansen som gies som argument.
- `ostream& operator<<(ostream &stream, const &LinkedList)`

  - som skriver ut listen i et lesbart format

Den lenkede listen du nettopp lagde kan brukes til å implementere andre datastrukturer på en lett måte. Hvordan ville du brukt LinkedList klassen for å implementere en *stack* eller *queue*? (Du trenger ikke å implementere dem.)

8 Søke i den lenkede listen (10%)

Legg til en medlemsfunksjon for å søke i den lenkede listen etter en gitt *string*-verdi. Funksjonen skal returnere en peker til den første noden som inneholder verdien, eller NULL hvis verdien ikke finnes i listen.

```
ListNode *LinkedList::search(const string &value);
```

9 Slette noder fra den lenkede listen (10%)

Legg til en medlemsfunksjon som fjerner og sletter (`delete`) alle elementene i en lenket liste som har lik verdi som en gitt string (argumentet til funksjonen).

```
void LinkedList::remove(const string &value);
```