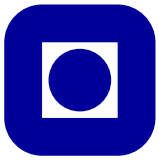

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2013

Øving 9

Frist: 2013-04-19

Mål for denne øvingen:

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

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.

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 (20 poeng)

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 oppgaven 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, rekkef\T1\o lgen 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, rekkef\T1\o lgen 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 Iteratorer (20 poeng)

a) Lag en vektor for strenger (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* som 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
```

3 Lister (Lists) (20 poeng)

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 *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.

4 «Sets» og «maps» (20 poeng)

En header-fil *Phonebook.h* er gitt med deklarasjon og delvis implementasjon av de to klassene *PhonebookEntry* og *Phonebook*. Du skal implementere de gjenstående funksjonene og overlagre operatorer.

PhonebookEntry inneholder et navn (som en *string*) og telefonnumre organisert i et "map" (map<string, string>) hvor den første strengen, eller nøkkelen (key), er en markelapp (label) som beskriver telefonnummeret ("home", "work", "mobile" osv.) og den andre strengen er telefonnummeret.

Phonebook inneholder et sett av PhonebookEntry (set<PhonebookEntry>)

a) For å lage et *set* må < operatoren være definert for elementene i settet. Overlagr derfor < operatoren for *PhonebookEntry*, slik at name-variabelen til objektene sammenlignes.

b) Implementer en funksjon, *add(const PhoneBookEntry copyFrom)* i *PhonebookEntry*.

Denne funksjonen tar inn et annet PhoneBookEntry *copyFrom*, og legger til alle telefonnumrene som ligger i *copyFrom* i objektet funksjonen kalles på. Hvis en label allerede eksisterer i mapet, skal oppføringen oppdateres.

```
PhoneBookEntry olaNormann har:
Home: 11 23 45 67
Work:  11 06 54 32

PhoneBookEntry olaMobil har:
Work: 11 33 44 55
Mobile: 11 22 33 44

olaNormann.add(olaMobil);

// Phonebookentry olaNormann har nå:
Home: 11 23 45 67
Work:  11 33 44 55
Mobile: 11 22 33 44
```

c) Implementer funksjonen *add* i *Phonebook*. Denne funksjonen tar inn et PhoneBookEntry *addMe* som skal legges til i PhoneBooken.

- Funksjonen skal sjekke om det eksisterer en oppføring med samme navn (*name*) som *addMe*.
- Dersom det allerede finnes en oppføring med dette navnet, skal verdiene i *addMe* brukes til å oppdatere den eksisterende oppføringen (ved å bruke funksjonen fra oppgave b)
- Ellers skal den nye oppføringen bare legges til i telefonboken.

Merknad om å endre elementer i et sett: *Det er uenighet om det skal være tillatt å endre på elementer i et sett. Noen kompilatorer, som g++, lar deg ikke gjøre dette fordi siden nøklene (keys) i et set skal være sorterte og unike skal de ikke kunne forandres på, siden dette kan føre til usorterte sett og duplikater. Andre kompilatorer, som den som brukes i Visual C++, lar deg derimot endre på elementene i et set, og lar det være opp til programmereren å sørge for at settet fortsatt er sortert med unike verdier.*

- *For å sørge for at koden din ikke lager et ugyldig sett, og fordi koden din bør være så allsidig som mulig, bør du endre på et element i et sett ved å bruke denne fremgangsmåten:*
- *Kopier elementet du ønsker å endre til en ny variabel*
- *Utfør endringene på denne variabelen*

- *Slett den gamle oppføringen fra settet*
- *Legg til den nye variabelen i settet*

d) Overlagr «-operatoren for både *PhonebookEntry* og *Phonebook*.
Hint: bruk operatoren du overlagrer for *PhonebookEntry* i implementasjonen av operatoren til *Phonebook*.

5 Søking (20 poeng)

a) Implementer funksjonen *find* for *Phonebook*:

- Funksjonen skal søke i telefonboken.
- *find*-funksjonen din bør være fleksibel og skal støtte å søke etter *substrings* (søke etter en del av navnet).
- Den første parameteren til funksjonen er navnet (eller en del av navnet) som det søkes etter.
- Den andre parameteren er *label*'en (eller en del av den) som det søkes etter.
- Den tredje parameteren er et *Phonebook*-objekt som brukes til å returnere navnene og numrene som søket finner.
- En tom streng ("") kan anses som å være en *substreng* av enhver streng. (Med andre ord, hvis argumentet som representerer navnet det søkes etter er tomt, skal alle navnene i telefonboken returneres, og hvis argumentet som representerer *label*'en er tomt, skal alle numrene i *PhonebookEntry*'en returneres.)

Hint: Implementer *find*-funksjonen til *PhonebookEntry* i tillegg, og bruk denne i *find*-funksjonen til *Phonebook*. Hint: Du må søke etter substrings ved å bruke *find(string&)*-funksjonen til *string*-klassen *find()*-funksjonen søker etter en *substring* og skiller mellom store og små bokstaver. (Hvis du ikke vil skille mellom stor og små bokstaver må du implementere din egen søkealgoritme.) Husk fra tidligere øvinger at *find()*-funksjonen returnerer *string::npos* hvis argumentet ikke finnes i strengen.

Eksempel (anta at pb er en telefonbok med flere oppføringer):

```
Phonebook result;
if(pb.find("Bob", "mobile", result))
        cout << result;
```

Resultat:

```
Bob the Builder
mobile: 22334455
Bob Hund
mobile1: 99999999
mobile2: 88888888
```

Eksempel:

```
Phonebook result;
if(pb.find("Mario", "", result))
        cout << result;
```

Resultat:

```
Super Mario
home: 12345678
mobile: 98765432
work: 33333333
```