TOD070 VIDEREGÅENDE PROGRAMMERING
Øving 4
Høsten 2010

HØGSKOLEN I BERGEN AVDELING FOR INGENIØRUTDANNING

Øving 4 Uke 39-40 innleveringsfrist mandag 11.10 kl.15.00 Sist oppdatert 28.09

Siste øving i C++ -delen av faget utlevert onsdag 22.09

Formål

- **Polymorfisme og virtuelle metoder**
- **Unntak**
- **STL**
- **Fil**

Oppgave 1

Skriv en klasse Dictionary som definerer en enkel ordliste. En ordliste består av et antall innganger. Hver inngang består av to deler: et referanseord og en forklarende tekst. Bruk en ekstra hjelpeklasse Entry til å definere en slik inngang.
Klassen Dictionary skal ha medlemsmetode for å legge inn en ny inngang. Tabellen trenger ikke være sortert. Det skal også være en indeksoperator[] som har en tekst som parameter. Denne teksten skal være et referanseord. Operatoren skal slå opp på referansen i ordlisten og gi som resultat en peker til tilsvarende forklarende tekst eller 0, (NULL-referansen) hvis referanseordet ikke fins.

Oppgave 2

a)
Det skal lages noen klasser (utvid gjerne med flere). **`Vehicle, Motor_vehicle, Private_car, Person, Owner`**
Basisklasen skal være abstrakt. En eier er altså en person som kan være eier av flere biler. Lag et program som tester ut klassene. Klassen Owner: Metoden add må legge inn ny kjøpt bil i tabellen og metoden erase må fjerne bil fra tabellen hvis den fins.

```
class Vehicle {
public:
  virtual void write_info()const = 0;;
};

class Motor_vehicle : public Vehicle {
public:
 Motor_vehicle(const string& no) : reg_num(no) {}
 const string& number() { return reg_num; }
 void write_info();
protected:
  string reg_num;
};
class Private_car : public Motor_vehicle {
public:
  Private_car(const string& no, int n)
  : Motor_vehicle(no), num_seats(n) {}
   void write_info()const;
protected:
  int num_seats;
};
```

```
class Person {
  public:
   Person() {};
   Person(const string& n) : name(n) { };
   const string getName() const;
   void setName(const string& n);
   void write_info() const;
  private:
    string name;
 };

class Owner : public Person {
public:
  Owner(const string& n) : Person(n) {};
  void add(Motor_vehicle& vehicle); // "add"
  void erase(Motor_vehicle& vehicle);//"erase"
  void write_info() const;
private:
    Motor_vehicle* carCollection; // dynamisk tabell, alt: bruke vector, da skriver
                                  //du kun vector<Motor_vehicle *> carCollection;
    int capasity;
    int used;
}
```

b)

Skriv en virtuell funksjon **`max`** som returnerer det største av to Vehicles (kjøretøy). La fp1, fp2 og fp3 være peker til Vehicle. Det skal være mulig å utføre fp3 = fp1.max(*fp2).
Lag et program som tester det ut, for eksempel for `Motor_vehicle`**.** Den abstrakte basisklassen har den ekte virtuelle metoden **:**
`virtual Vehicle* max(Vehicle* Pveh) = 0;` Vi må bruke dynamisk casting for å få det til.
Ellers er det å bemerke at vi bør unngå å bruke dynamisk casting, se lærebok. Vi kan bruke pekere eller referanser. Se lærebok side 706 og punktene 1 og 2 samme side.
Første delen av koden for **`max`** i klassen **`Motor_vehicle`** er vist under:
Metoden tar inn en referanse til en Vehicle og returnerer en peker til en Motor_vehicle:

```
Motor_vehicle* Motor_vehicle::max(Vehicle& veh){
Motor_vehicle* mveh = dynamic_cast<Motor_vehicle*>(veh);
```

Fyll ut resten i metoden og lag et testprogram.
…

c)
Definer en klasse **`Point`** som definerer et punkt (x, y) i et todimensjonalt koordinatsystem.
Definer en klasse **`Figure`** som inneholder et startpunkt. La klassen **`Figure`** ha en ekte virtuell metode **`area().`** Figurene skal være basisklasse for tre avledet klasser, **`Circle`**, **`Triangle`** og **`Rectangle`**.
For at det skal være enkelt å beregne areal gjør vi følgende forenklinger:
1) Sidene i et rektangel er parallell med x -og y-aksen.
2) Trekantene er rettvinklete der en side er parallell med x-aksen og en med y-aksen.
De tre avledete klassene skal ha passende medlemsdata og sin egen versjon av metoden area() som skal beregne arealet.
Lag en tabell av pekere til en vilkårlig figur og et testprogram som skriver ut areal til ulike figurer.

Oppgave 3

Generelt om unntak:
Under utføring av program kan feilsituasjoner oppstå.
For eksempel:

- indeksering utenfor tabell
- feile inndata til en funksjon

Forhåndsdefinerte unntak:
Vi har en del forhåndsdefinerte unntak som er ordnet i arvehierarki .
Se figuren under

Følgende unntak er brukt for Standard Library og kan bli brukt av programmerere.

Standard Exceptions, derived from logic_error, thrown by Standard Library		
Name	Thrown by	Header
domain_error	Used to report that a function's argument is outside the set of values the function can accept.	<stdexcept>
invalid_argument	bitset constructor	<stdexcept>
length_error	standard string	<stdexcept>
out_of_range	Indexed STL container at() member. Used to report that an argument values is not in an expected range as when a bad index is used in a string, or an array like collection	**<stdexcept>**
Standard Exceptions, derived from runtime_error. These are used to report errors beyond the scope of a program and are not easily avoidable.		
range_error	used to report internal computation range errors	<stdexcept>
overflow_error	used to report an arithmetic overflow	<stdexcept>
underflow_error	used to report an arithmetic overflow	<stdexcept>

a) Gitt følgende rekursive kode for å beregne fakultet:

```
int nfac(int n) // calculate n!{
    if(n <= 0)
        return 1;
   else
        return n*nfac(n-1);
}
```

Lag en ny versjon av nfac som 1) returnerer resultatet som et flyttall av type float og 2) kaster et passende unntak hvis n blir for stor. Største flyttall er gitt ved: numeric_limits<float>::max(). Inkluder også #include <limits>

b)

Oppgave: Modifiser funksjonene "**addElement" og "operator[]" PFARRAy kap.10 side 482- 486 s**lik at et unntak av typen OutOfRange kastes hvis tabellindeksen er utenfor grensen, eller hvis det er forsøk på innsetting utenfor grensen (maks kapasitet).

```
#ifndef PFARRAYD_H
#define PFARRAYD_H
#include <string>
using std::string;
class OutOfRange
{
public:
  OutOfRange(string thisMessage, int thisIndex);
  string getMessage() const;
  int getIndex() const;
private:
  string message;
  int index;
};
//Objects of this class are partially filled arrays of doubles.
class PFArrayD
{
public:
  PFArrayD( );
  //constructors
  void addElement(double element) throw(OutOfRange);
  //Precondition: The array is not full.
  //Postcondition: The element has been added.
  bool full( ) const { return (capacity == used); }
  //Returns true if the array is full, false otherwise.
  int getCapacity( ) const { return capacity; }
  int getNumberUsed( ) const { return used; }
  void emptyArray( ){ used = 0; }
  //Empties the array.
  double& operator[](int index) throw(OutOfRange);
  //Read and change access to elements 0 through numberUsed - 1.
  PFArrayD& operator =(const PFArrayD& rightSide);
  virtual ~PFArrayD( );
  private:
    double *a; //for an array of doubles.
    int capacity; //for the size of the array.
    int used; //for the number of array positions currently in use.
};
#endif //PFARRAYD_H
```

OBS! Det er metoden **testPFArray()** side 485 som skal fange opp unntakene.
Benytt gjerne main-programmet på side 484.

Oppgave 4

Ordfilen henter dere ned fra bokens nettsted.

a) Lærebok Programming Project nr 17 lærebok side 595.

Dere kan velge å gjøre b) eller c)

b) Lærebok Programming Project nr 10 lærebok side 933.

c) Lærebok Programming Project nr 10 lærebok side 933.