Laboration 3

Mål: Du ska på att konstruera och använda egna klasser i C++. Du ska också träna på användning av datamedlemmar, medlemsfunktioner, konstruktorer, destruktorer, uppräkningstyper och överlagrade operatorer för för dessa.

Läsanvisningar

Läs kap 7–8 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U1, U3a-b, U4 och U5. Läs igenom och sätt dig in i uppgifterna U2, U3c och U6. Uppgift U1 och U2 är frivillig.

Uppgifter

U1. a) Konstruera en `Rektangel`-klass som har rektangelns längd och bredd (båda flyttal) som datamedlemmar. Klassen ska ha följande medlemsfunktioner:

- `sattVarden` som ger datamedlemmarna värden (via parametrar)
- `area` som räknar ut och returnerar rektangelns area (som funktionsresultat)
- `visa` som skriver ut rektangelns längd och bredd

Inför en `main`-funktion som skapar ett `Rektangel`-objekt och använder medlemsfunktionerna. Rektangelns längd och bredd läses in från tangentbordet.

b) Komplettera `Rektangel` med medlemsfunktionen `omkrets` som returnerarrektangelns omkrets. Testa funktionen i `main`.

c) Inför ytterligare en medlemsfunktion, `forstora`, som har ett heltal (`faktor`) som parameter och som multiplicerar rektangelns längd och bredd med detta heltal (medför att rektangeln förstoras). Komplettera `main` så att den använder funktionen.

U2. Skriv en klass, `Mynt`, som kan användas för att simulera myntkast. Klassen skall ha de publika (synliga) medlemsfunktionerna:

```
Mynt()             // Initierar myntobjekt så att en slumpmässig sida
                   // kommer upp (krona/klave)
void kasta()       // Simulerar ett myntkast
void visa()        // Skriver texten "Krona" eller "Klave", beroende på
                   // myntets tillstånd, dvs vilken sida som är vänd uppåt
Myntsida uppsida() // Returnerar myntets tillstånd
```

- Representera de två tillstånden med en uppräkningstyp,
  `enum Myntsida krona, klave`
  som definieras i klassens publika (synliga) del
- Klassens datamedlemmar skall vara privata
- Lägg klassdefinitionen i en fil med namnet `mynt.h`, och definitionerna av medlemsfunktionerna i `mynt.cpp`

a) Skriv ett program, som låter användaren göra upprepade kast med ett mynt. Resultatet av varje kast skall skrivas ut.

b) Skriv ett program som simulerar n kast (där n är ett positivt heltal) med två mynt (kasten visas) och rapporterar hur många av utfallen som blev lika.

c) Generalisera föregående uppgift så att m mynt kan användas (använd en dynamiskt allokerad array av myntobjekt).

U3. I en tidigare uppgift implementerades att spela 15-spelet i ett kommandofönster. Denna uppgift skall nu anpassas till grafiskt användargränssnitt. Klasser för det grafiska gränssnittet finns färdiga och kan laddas ner från kursens hemsida. Där finns filen `BrickData.h` som definierar klassen `BrickData` och de operationer som är används av gränssnittet för att implementera spellogiken. Komplettera klassen med lämpliga medlemsvariabler och implementera medlemsfuktionerna i filen `BrickData.cpp`

```
class BricksData
{
    public:
        enum Movement {Up = -4, Left = -1, None = 0, Right = 1, Down = 4};

        // Initialize data structures to the initial position:
        //  1  2  3  4
        //  5  6  7  8
        //  9 10 11 12
        // 13 14 15
        BricksData();

        // If 'toMove' is a legal brick to move then update the data
        // structures, add the moved bricks to 'brickList', and return the
        // direction of the move
        // Otherwise return the move direction 'None'
        Movement Move(int toMove, std::vector<int> &brickList);

        // Add all brick numbers in order according to their current
        // position (use '0' for the empty square)
        // The order for the initial positions is:
        // 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
        void Positions(std::vector<int> &brickList);

        // Reset the data structures to the initial position
        void Reset();

        // Shuffle the bricks to random positions
        void Shuffle();

        // Return 'true' if the puzzle is solved, i.e. all bricks are in
        // their initial positions
        bool Check();

    private:
};
```

a) Implementera konstruktorn `BricksData()` samt operationerna `Reset()`, `Positions()` och `Check()`.

b) Implementera operationen `Move()`. Enbart flyttning av enstaka brickor behöver im-

plementeras (i startpositionen är 12 och 15 de tillåtna brickorna att flytta). Gränssnittet klarar dock av att flyttar flera brickor åt gången. I startpositionen blir då även t.ex 13 och 8 tillåtna drag. Effekten blir samma som en sekvens av dragen 15, 14, 13 resp 12, 8.

c) Implementera operationen `Shuffle()`. Det är önskvärt (men inte nödvändigt) den nya spelställningen är lösbar. Detta kan man uppnå t.ex. genom att implementera en paritetsfunktion (t.ex. `"http://mathworld.wolfram.com/15Puzzle.html"`) och använda denna för att bestämma ordningen mellan de två sista brickorna. En annan metod att säkerställa att pariteten bevaras under blandningen är att göra tillräckligt många slumpmässiga drag.

U4. I en datoriserad svensk-engelsk ordbok beskrivs ordpar av följande klass:

```
class Word {
public:
   Word(char *sw, char *eng);   //Svenskt sw, engelskt eng
   ~Word();
   const char *get_sw() const;  //Hämta svenskt ord
   const char *get_eng() const; //Hämta engelsk översättning
private:
   char *swedish;
   char *english;
};
```

a) Implementera klassen `Word`. Tänk på att medlemsvariablerna `swedish` och `english` är pekare till textsträngar som måste allokeras, kopieras och avallokeras på ett korrekt sätt i konstruktorn och destruktorn.

b) Skriv en klass `Dictionary` som beskriver ordboken. Operationer: Lägg in ett nytt svenskt ord (med engelsk översättning), tag reda på den engelska översättningen av ett svenskt ord. Du får använda valfri metod för att lagra ordobjekten i ordboken. Följande funktioner ur C-biblioteket `<cstring>` kan komma till användning:

```
int strlen(char* s);
void strcpy(char* dest, char* source);
int strcmp(char* s1, char* s2);
```

U5. a) Konstruera en klass Kvadrat i vilken ingår:

- Datamedlem: kvadratens sida (heltal)
- Konstruktorer: Standardkonstruktor, en konstruktor med kvadratens sida som argument, kopieringskonstruktor (om sådan behövs, om inte motivera!)
- Medlemsfunktion som returnerar kvadratens area

b) Överlagra följande operatorer:

- Tilldelningsoperatorn (om det behövs)
- Öknings- och minskningsoperatorerna `++` och `-` (prefixvarianten räcker).
- Låt `++` betyda ökning av kvadratens sida med `1` och `-` minskning med `1` (om inte sidan blir `< 0`)
- `+` och `-` som betyder ökning resp minskning av kvadratens sidlängd med ett heltal (sidan får dock ej bli `< 0`)
- Jämförelseoperatorerna (`<`, `>`, `==` osv) med vänfunktioner

Skriv även ett testprogram.

U6. Implementera klassen `Personnummer`. Den ska använda teckenarray eller charpekare för att lagra personnumret och ha en boolsk variabel som anger om personnumret är OK eller inte. Överlagra operatorerna « och » för in- och utmatning av personnummer. kontrollera personnumret med en privat funktion (hur långt kontrollen ska drivas bestämmer du själv, men den ska åtminstone avgöra om kontrollsiffran är riktig). Överlagra även operatorn `!` så att den returnerar `true` om personnumret är felaktigt (utnyttja den boolska variabeln i Personnummer). Klassen ska kunna användas på följande sätt:

```
Personnummer persnum;
cout << "Ange personnummer: ";
cin >> persnum;
while (!persnum) {
  cerr << "Personnumret är felaktigt!\n";
  cout << "Ange personnummer: ";
  cin >> persnum;
}
cout << "Personnumret " << persnum << " är korrekt.\n";
```

Du bestämmer själv om personnummer bara ska innehålla siffror eller om det även ska (eller får) innehålla ett bindestreck före de 4 sista siffrorna.
Kontrollsiffran i ett personnummer kan bestämmas på följande sätt:

1. Utgå från de första 9 siffrorna och multiplicera siffrorna på udda plats (1, 3 osv) med 2 och siffrorna på jämn plats (2, 4 osv) med 1.
2. lägg ihop alla *siffrorna* i dessa produkter (dvs 12 räknas som 1+2=3).
3. Tag entalssiffran i den framräknade summan. Kontrollsiffran är då 10-entalssiffran, utom om entalssiffran är 0, då också kontrollsiffran är 0.