Inlämningsuppgift 1

Inledning:

Denna uppgift belyser konceptet **ADT (Abstract Data Type)** och specifikt de abstrakta datatyperna **Cirkulär Lista** och **Stack**.

I denna uppgift kommer du att implementera en lista vilken har ett cirkulärt beteende. Listan ska användas för ett kortspel benämnt 99 (se beskrivning i Del II). Listan och Stacken ska användas i ett kortspel med namnet 99.

Implementationsdetaljer:

- Template
- Inre klass
- Pekare
- Abstrakt klass
- Rent virtuella funktioner

På kursens hemsida finner du följande filer vilka ska ingå i din lösning:

- ICircularDoubleDirectedList.h
- NinetyNine.cpp
- Card.h
- Card.cpp
- Player.h
- Player.cpp
- IStack.h

Du finner även följande filer vilka du ska använda för att till viss del testa din implementation:

- TestFunctionalityOfList.cpp
- TestDeepCopyingOfList.cpp

Krav på uppbyggnad:

Listan ska som datastruktur länka noder cirkulärt i två riktningar. En inre klass `Node` ska definieras.

Gränssnittet för listan ges av klassmallen `ICircularDoubleDirectedList` enligt:

```
enum direction{NEXT, PREV};
template <typename T>
class ICircularDoubleDirectedList
{
public:

	virtual ~ICircularDoubleDirectedList() {};
	virtual void add(T& item) = 0;
	virtual void remove(T& item) = 0;
	virtual int size() const = 0;
	virtual T& currentItem() = 0;
	virtual void changeDirection() = 0;
	virtual void move() = 0;
};
```

Gränssnittet för stacken ges av `IStack` enligt:

```
template <typename T>
class IStack
{
public:
	virtual ~ IStack () {};
	virtual void push(const T& element) = 0;
	virtual T pop() = 0;
	virtual T peek() const = 0;
	virtual bool isEmpty() const = 0;
};
```

Datastruktur kan för stacken vara antingen enkellänkning eller en array. Observera att stacken är obegränsad avseende antalet element stacken kan innehålla.

Del I:

Klassmallen `ICircularDoubleDirectedList` finns i filen ICircularDoubleDirectedList.h. Denna ska ingå i din lösning, d.v.s i det Visual Studio projekt som kommer att innehålla filerna för lösningen till uppgiften.

Du ska dessutom skapa en klassmall benämnd `CircularDoubleDirectedList` vilken ärver publikt från `ICircularDoubleDirectedList`.

I klassmallen `CircularDoubleDirectedList` definieras klassen för dubbelriktade noder. Dubbelriktade noder länkas för att erhålla en dubbellänkad cirkulär lista. Dessutom deklareras följande medlemsvariabler:

- `current` vilken pekar ut aktuell nod
- `nrOfItems` vilken innehåller antalet element i listan
- `currentDirection` vilken innehåller aktuell riktning

När en lista skapas ska dess aktuella riktning vara NEXT.

Beskrivning av medlemsfunktioner:

- `add(…)`
  - Om listan är tom skapas en nod för `item` vilken blir den enda noden i listan. Den nya noden är därefter aktuell nod.
  - En ny nod skapas för `item` vilken placeras **efter** aktuell nod, d.v.s blir efterföljare till aktuell nod. Den nya noden ska därefter bli aktuell nod.
- `remove(…)`
  - Om listan är tom kastas strängen ”Exception: call of remove on empty list” som undantag.
  - Den nod som innehåller `item` tas bort. Om den nod som tas bort är den nod som är aktuell (`current` pekar på denna) ska aktuell nod bli efterföljande nod.
- `size()`
  - Returnerar antalet element som finns i listan.
- `currentItem()`
  - Om listan är tom kastas strängen ” Exception: call of currentItem on empty list” som undantag.
  - Returnerar det element som är aktuellt (som finns i den nod som `current` pekar på).
- `changeDirection()`
  - Ändrar riktningen för listan
- `move ()`
  - Om listan är tom kastas strängen ” Exception: call of move on empty list” som undantag.
  - Ändrar `current` till ”nästa” nod enligt den riktning som ges av `currentDirection`.

Därutöver ska även konstruktor, destruktor, kopieringskonstruktor och tilldelningsoperator implementeras. Djup kopiering (deep copying) ska användas.

Klassmallen `IStack` finns i filen IStack.h. Denna ska ingå i din lösning, d.v.s i det Visual Studio projekt som kommer att innehålla filerna för lösningen till uppgiften.

Du ska dessutom skapa en klassmall benämnd `Stack` vilken ärver publikt från `IStack` (du kan här använda en av de Stack-implementationer du gjort i laboration 1, du byter namn på denna till Stack).

Medlemsfunktionerna `pop()` resp `peek()` ska, om stacken är tom, kasta strängen ”Exception: empty queue” som undantag.

Testning:

Genomför tester för listan med följande testprogram:

- TestFunctionalityOfList.cpp
- TestDeepCopyingOfList.cpp

Testningen ska genomföras i debug-läge för att upptäcka eventuella minnesläckor. Vid exekvering framgår det via utskrifter vad som testas och om det som testas är ok. Om någon test inte är ok anges förväntade utskrifter och därefter den utskrift som ges av programmet när din lista används. Dessa utskrifter ska vara desamma.

Testerna är inte fullständiga och du kan behöva göra ytterligare tester.

Inga minnesläckor får förekomma.

Del II:

Listan och stacken ska nu användas för att hantera en del i implementationen för kort spelet 99. Du behöver tillföra din implementationer av klassmallarna `CircularDoubleDirectedList.h` och `Stack.h`. Du ska dessutom implementera delar i filen NinetyNine.cpp.

Beskrivning av kortspelet 99

En vanlig kortlek används (52 kort med valörerna spader, hjärter, ruter och klöver). Varje spelare har alltid 3 kort på handen. Det finns en korthög på vilken den spelare som är i tur lägger ett av sina kort. Korthögens värde påverkas enligt det lagda kortets värde/regel. Spelaren tar ett nytt kort från kortleken. Därefter är det nästa spelares tur. Spelet fortgår till en spelare tvingas lägga ett kort som gör att korthögens summa överstiger värdet 99.

Olika kort är förknippade med speciella värden/regler enligt:

dam och kung ökar värdet på korthögen med 10
knekt sätter korthögens värde till 99
10 minskar korthögens värde med 10
9 bibehåller korthögens värde men vänder håll på turordningen för spelarna
8 bibehåller korthögensvärde
ess ökar korthögens värde med 1
övriga kort ökar korthögens värde kortets värde (ex-vis en 5:a ökar korthögens värde med 5)

Ett exempel på några steg i spelet 99 för tre spelare kallade A, B och C vilket är den ordning de ska följer:

Varje spelare får 3 kort. När spelet börjar lägger A kortet Hjärter 4 vilket innebär att korthögens värde är 4. A tar ett nytt kort från kortleken. Nästa spelare som är B lägger kortet Spader Dam. Korthögens värde blir då 14. B tar ett nytt kort från kortleken. Spelare C lägger Ruter Knekt och korthögens värde blir då 99. C tar ett nytt kort från korthögen. Nästa spelare är A som lägger kortet Ruter 9. Korthögens värde påverkas inte men ”spelordningen” vänder, dvs nästa spelare blir C. A tar ett nytt kort från korthögen. C lägger kortet 10 och korthögens värde blir 89. C tar ett nytt kort. Nästa spelare som är B lägger Ruter Kung. Korthögens värde blir 99. B tar ett nytt kort från korthögen. A lägger Hjärter 2 vilket gör att korthögens värde blir över 99 och A har därmed förlorat.

Spelarna B och C fortsätter men först placeras A’s kort och korthögens kort i kortleken vilken också blandas.

Slutligen återstår en spelare vilken är segraren.

Filen NinetyNine.cpp

Det finns kommentarer i denna fil för det som du ska implementera.

Slutprodukt och inlämning

Slutprodukten är ett fungerande program som motsvarar beskrivningen av spelet 99 där dina filer (listan och stacken) ingår tillsammans med de filer som tillhandahållits. Dina implementationer av listan och stacken ska vara testade. Du kan testa listan med de testprogram som tillhandahållits. Dina implementationer får inte ge några minnesläckor.

Du lämnar in de filer som motsvarar slutprodukten som en packad fil vilken du namnger med ditt för och efternamn följt av NinetyNine.