Задачи за задължителна самоподготовка

по

Обектно-ориентирано програмиране

Наследяване и виртуални функции

email: kalin@fmi.uni-sofia.bg

22 април 2016 г.

1. Задача 2.4.25. В софтуерна фирма има два вида служители – програмисти и мениджъри. Отдел "личен съста" поддържа следната информация за всеки от програмистите:

   - име;
   - стаж (в месеци);
   - дали знае C++;
   - дали знае C#

   и следната информация за всеки от мениджърите:

   - име;
   - стаж (в месеци);
   - колко човека управлява.

   Да се напише програма, която позволява на отдел "личен състав" да поддържа списък с всички програмисти и мениджъри във фирмата. Програмата да може да извършва следните операции:

   - постъпване на нов служител;
   - напускане на служител;
   - извеждане на списък с данни за всички служители.

Да се въведат примерни данни за служители в софтуерна фирма и над тях да се изпълнят следните операции:

- изтриване на всички служители, които имат стаж по-малко от 3 месеца;
- записване на данните в текстов файл;
- прочитане на данните от текстов файл;

2. Шахматни фигури.

   (а) Да се дефинира структура `ChessPosition`, описваща коректна позиция на фигура върху шахматна дъска ('A'-'H',1-8). Да се дефинира абстрактен клас `ChessPiece`, описващ шахматна фигура със следните операции:

   - `ChessPosition getPosition ()`: Дава позицията на фигурата на дъската
   - `[подходящ тип] allowedMoves ()`: Дава списък с всички възможни позиции, до които дадена фигура може да достигне с един ход
   - `bool wins (ChessPosition)`: Проверява дали фигурата "владее" дадена позиция, т.е. дали позицията е в списъка с възможни ходове на фигурата

   (б) Да се дефинират класовете `Rook` и `Knight` - наследници на `ChessPiece`, описващи съответно шахматните фигури топ и кон.

   (в) "Стабилна конфигурация" наричаме такава подредба на фигурите по дъската, при която никоя фигура да не е върху позволен ход на друга фигура (т.е. никои две фигури не се "бият"). Да се дефинира функция `allMoves ([подходящ тип] pieces[, ...])`, която по списъка `pieces`, съдържащ произволен брой разнородни шахматни фигури, отпечатва на конзолата всеки възможен ход на фигура от `pieces` такъв, че след изпълнението му списъка с фигури представлява стабилна конфигурация. Информацията за ходовете да съдържа типа на фигурата, старата позиция и новата позиция, например:

   ```
   Queen A1 -> B2
   Knight B3 -> A5
   ```

Забележка: Реализирайте всички конструктори и други операции, които смятате, че са необходими на съответните класове.
Забележка: Под "списък" се има предвид обект от класовете `List` или `DynArray`, разработени на лекции, или който е да е друг тип, който познавате и който представлява контейнер за обекти.

3. Походова игра

Нека GameBoard е дадена квадратна матрица `NxN` от цели числа, всеки елемент на която има стойност 0, 1 или 2. Елементите на матрицата със стойност 0 наричаме "земя", тези със стойност 1 наричаме "огън", а тези със стойност 2 - "вода". `GameBoard` ще наричаме "игрова дъска". Под "съседна позиция" на позицията (i, j) ще разбираме тези елементи на матрицата (i', j'), такива че $|i - i'| \leq 1$ и $|j - j'| \leq 1$.

В условието по-долу `N` и `GameBoard` да се приемат за предварително дефинирани глобални променливи.

(а) Да се дефинира структура `Point`, описваща позиция на игровата дъска, задаща реда и колоната на нейн елемент. Да се дефинира абстрактен клас (или интерфейс) `GamePlayer`, който описва играч на игровата дъска със следните операции:

- `Point getPosition ()`: Дава позицията на играча на дъската
- `[подходящ тип] allowedMoves ()`: Дава списък с всички възможни позиции, до които даден играч може да достигне с един ход
- `bool wins (Point)`: Проверява дали играча "владее" дадена позиция, т.е. дали позицията е в списъка с възможни ходове на играча

(б) Да се дефинира клас `Knight`, наследник на `GamePlayer`, описващ "сухопътен рицар". Позволените ходове на сухопътния рицар се задават със следното правило: Рицарят може да се премести в позициите, които са съседни на текущата му и които:

- са земя
- нямат огън в съседство

(в) Да се дефинира клас `SeaMonster`, наследник на `GamePlayer`, описващ "морско чудовище". Позволените ходове на морското

чудовище се задават със следното правило: Морското чудовище може да се премести във всички позиции, които са достижими от неговата при придвижване по хоризонтала или вертикала, което преминава само през вода. Например, ако вдясно от играча има три поредни позиции с вода, следвани от една позиция земя, то и трите водни позиции са достижими, но земната и всички вдясно от нея - не.

(г) "Стабилна конфигурация" наричаме такава подредба на играчите по дъската, при която никой играч да не е върху позволен ход на друг играч (т.е. никои два играча не се "бият"). Да се дефинира функция `allMoves ([подходящ тип] players[, ...])`, която по списъка `players`, съдържащ произволен брой разнородни играчи, отпечатва на конзолата всеки възможен ход на играч от `players` такъв, че след изпълнението му списъка с играчи представлява стабилна конфигурация. Информацията за ходовете да съдържа типа на играча, старата позиция и новата позиция, например:

```
Knight (0,0) -> (1,1)
SeaMonster (2,2) -> (5,2)
```

Забележка: Реализирайте всички конструктори и други операции, които смятате, че са необходими на съответните класове.
Забележка: Под "списък" се има предвид обект от класовете `List` или `DynArray`, разработени на лекции, или който е да е друг тип, който познавате и който представлява контейнер за обекти.

Някои от задачите са от сборника *Магдалина Тодорова, Петър Армянов, Калин Николов, "Сборник от задачи по програмиране на C++. Част втора. Обектно-ориентирано програмиране"*. За тези задачи е запазена номерацията в сборника.