**Game**

Иванчо и Марийка много обичат да играят на играта “Колекционирай числата”. Играта е много проста, но в същото време доста занимателна за тях. В началото на играта се определят **капацитет** на колекцията. Това е броят елементи, които могат да се колекционират. Колекционирането се случва по доста прост начин - чрез хвърляне на два зара от съответния играч. Така получената двойка се записва в колекцията.

В играта основна важност играе операцията сумиране на двойки – това е операция при която се събират съответните елементи на две двойки и се получава нова двойка.

Разгледайте следната схема, за да се ориентирате как правилно да сумирате двойките:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | + | 3 | = | 5 |
| 5 | 3 | 8 |

При запълване на капацитета на колекцията, всички двойки числа се сумират, така че да се получи една двойка и тя става нов член на колекцията, а останалата част от двойките, които са били получени чрез заровете изчезват. **Сумирането на двойките се случва в момента на добавянето на нова двойка, за която няма място!**

Когато това се случи, играта все пак може да продължи. Новите елементи се добавят след двойката, получена от сумиране и при достигане на капацитета отново, двойките получени от заровете се сумират по същия начин, а резултатът се добавя след предната двойка получена от сумиране. **ВАЖНО: В сумирането на участват двойките, които са получени като сумиране, а само тези, които са добавени от заровете.**

Процесът се повтаря отново и отново, докато не се запълни цялата колекция с двойки, получени от сумиране или пък ако не бъде дадена команда за край на играта.

За да се определи финалния резултат на всеки участник се вземат всички негови двойки, получени като резултат от сумиране и се извършва сумиране върху тях, така че да се получи една крайна двойка. Победител е този, при който абсолютната стойност на разликата между двете числа е **най-малка**.

Вашата задача е да създадете програма, която да следи играта за **n** на брой играчи и да отговаря на определени команди:

**Първа подзадача (30 точки)**

В рамките на тази подзадача трябва да реализирате две комадни:

**Dice** <играч> <число1> <число2> - тази команда задава на кой играч какви числа са се паднали.

**CurrentPairSum** <играч> - тази команда показва сумата от всички двойки във формат: **(a, b)**, където a и b са съответно първия и втория елемент на получената двойка.

**Втора подзадача (30 точки)**

В рамките на тази подзадача реализирайте следната команда:

**CurrentState** <играч> - тази команда отпечатва всички двойки, които са съхранени в колекцията на дадения играч, без значение как са получени

**Трета подзадача (20 точки)**

**Winner** - тази команда отпечатва победителя във формат: “{играч} wins the game!”. Командата ще бъде викана само тогава, когато може да се определи еднозначно победител.

**Четвърта подзадача (20 точки)**

**CurrentStanding** - тази команда отпечатва класиране в намалящ ред, в което на всеки ред има информация за един играч във формат: “{играч} - (**a**, **b**)”. Където **a** и **b** са числата от двойката на съответния играч, с която той участва в класирането.

За ваше удобство получвате скелет за решението на тази задача.

**Клас Pair.cs**

Този клас трябва да описва всяка двойка. Класът трябва да поддържа свойства, които съхраняват първата и втората стойност на двойката. Конструктор с два параметъра – числата от двойката. ToString метод и Difference метод, който връща абсолютната стойност на разликата между първия и втория елемент на двойката.

|  |
| --- |
| Pair.cs |
| class Pair {          public int First { get; set; }          public int Last { get; set; }            public Pair(int first, int last)          {              //TODO: Add code          }            public override string ToString()          {              //TODO: Add code          }            public int Difference()          {              //TODO: Add code          }  } |

**Клас CapacityList.cs**

Този клас ще описва структурата, която съхранява всички двойки. По своята същност тя прилича на статичен списък, като тази структура трябва да има масив от обекти от клас **Pair**. Освен това трябва да разполага с конструктор, който създава масив от Pair обекти. Трябва да поддържате свойство Count, което да показва броят на двойките в масива, които **могат** да участват в класирането. Методът SumIntervalPairs() трябва да може да сумира всички двойки, започвайки от зададен стартов индекс – този метод ще се използва вътрешно от класа, за да определя сумата от двойките за всеки следващ елемент, който запълва. Методът Sum() трябва да може да сумира всички двойки, които имат право да участват в сумата за класирането. Методът Add(Pair item) има за цел да добавя нова двойка към  структурата и да извиква методът за сумиране, когато структурата е на път да се запълни. PrintCurrentState() отпечатва информация за текущите двойки в масива.

|  |
| --- |
| CapacityList.cs |
| class CapacityList {          private const int InitialCapacity = 2;          private Pair[] items;            private int startIndex = 0; //показва първият индекс, от който започваме да сумираме текущите елементи          private int nextIndex = 0; //показва поредният индекс, на който можем да поставим елемент            public CapacityList(int capacity = InitialCapacity)          {              this.items = new Pair[capacity];          }            public int Count          {              get;              private set;          }            public Pair SumIntervalPairs()          {               //TODO: сумирайте двойките от startIndex до nextIndex          }            public Pair Sum()          {               //TODO: сумирайте двойките от 0 до this.Count – всички двойки, които имат право да участват в класирането          }            public void Add(Pair item)          {               //TODO: Добавяне на двойката          }            public void PrintCurrentState()          {               //TODO: отпечатайте всички двойки от 0 до nextIndex          }   } |

**Вход**

На първи ред ще въведете капацитета на структурата. На всеки от следващите редове ще получавате някоя от командите. Въвеждането на команди приключва с командата **End**

**Изход**

За всяка от командите изведете изхода посочен по-горе.

**Примери**

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснение** |
| 3  Dice Gosho 1 3  Dice Ivancho 5 4  Dice Mimi 2 3  Dice Gosho 2 5  Dice Ivancho 3 3  CurrentPairSum Gosho  Dice Gosho 1 1  CurrentPairSum Gosho  Dice Gosho 3 2  CurrentPairSum Gosho  Dice Ivancho 2 1  Dice Gosho 3 3  Dice Ivancho 2 2  CurrentPairSum Ivancho  Dice Mimi 6 6  Dice Mimi 3 2  Dice Mimi 1 6  CurrentPairSum Mimi  End | (0, 0)  (0, 0)  (4, 9)  (10, 8)  (11, 11) | Първият път сумата на Гошо е (0, 0), т. като досега той не е изтеглил достатъчно двойки, за да формира двойка за сума по правилата. Вторият път Гошо вече има 3 двойки, но те не се сумират, докато не се прибави и четвърта. Едва на третия път Гошо вече има получена сума.    Иванчо има двойка (10, 8) - тя е от двойките (5, 4)+(3, 3)+(2, 1), a двойката (2, 2) не участва тъй като тя е само добавена, а не получена от сумиране.    Мими има двойка (11, 11) - тя е получена от двойките (2, 3)+(6, 6)+(3, 2), а (1, 6) не участва в сумата. |
| 3  Dice Ivancho 1 3  Dice Ivancho 5 4  Dice Ivancho 2 3  Dice Ivancho 2 5  Dice Ivancho 3 3  Dice Ivancho 1 1  CurrentState Ivancho  Dice Gosho 3 3  Dice Gosho 1 1  Dice Gosho 3 2  Dice Gosho 2 1  Dice Gosho 3 3  Dice Gosho 2 2  CurrentState Gosho  End | (8, 10)  (5, 8)  (1, 1)  (7, 6)  (5, 4)  (2, 2) | Първите три двойки са тези на Иванчо, като първата двойка е получена като сума на: (1, 3)+(5, 4)+(2, 3), а втората като сума на (2, 5)+(3, 3), третата двойка е двойката, която се е паднала от заровете.    Вторите три двойки са на Гошо. Първата е сума на (3, 3)+(1, 1)+(3, 2), втората е сума на (2, 1)+(3, 3). |
| 3  Dice Ivancho 1 3  Dice Ivancho 5 4  Dice Ivancho 2 3  Dice Ivancho 2 5  Dice Ivancho 3 3  Dice Ivancho 1 1  CurrentState Ivancho  Dice Gosho 3 3  Dice Gosho 1 1  Dice Gosho 3 2  Dice Gosho 2 1  Dice Gosho 3 3  Dice Gosho 2 2  CurrentState Gosho  CurrentPairSum Ivancho  CurrentPairSum Gosho  Winner  End | (8, 10)  (5, 8)  (1, 1)  (7, 6)  (5, 4)  (2, 2)  (13, 18)  (12, 10)  Gosho wins the game! | В сумата за изчисляване на победител в този случай не участват (1, 1) за Иванчо и (2, 2) за Гошо, т.като не е въведена 7 двойка, която да принуди структурата да извърши сумиране и затова сумата за изчисляване на победител се получава само от по 2-те двойки, които са получени като резултат от сумиране.  Гошо печели играта, т.като при него разликата е 2, а при Иванчо - 5. |
| 3  Dice Gosho 1 3  Dice Ivancho 5 4  Dice Mimi 2 3  Dice Gosho 2 5  Dice Ivancho 3 3  CurrentPairSum Gosho  Dice Gosho 1 1  CurrentPairSum Gosho  Dice Gosho 3 2  CurrentPairSum Gosho  Dice Ivancho 2 1  Dice Gosho 3 3  Dice Ivancho 2 2  CurrentPairSum Ivancho  Dice Mimi 6 6  Dice Mimi 3 2  Dice Mimi 1 6  CurrentPairSum Mimi  CurrentStanding  Winner  End | (0, 0)  (0, 0)  (4, 9)  (10, 8)  (11, 11)  Mimi - (11, 11)  Ivancho - (10, 8)  Gosho - (4, 9)  Mimi wins the game! | Тестът е аналогичен на първия, но с добавени команди за класиране и победител, като този път победител е Мими, т.като нейната разлика е 0. |

## Requested files

#### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameSkeleton

{

class Program

{

static void Main(string[] args)

{

int capacity = int.Parse(Console.ReadLine());

Dictionary<String, CapacityList> players = new Dictionary<string, CapacityList>();

string command = "";

do

{

command = Console.ReadLine();

string[] commands = command.Split(' ').ToArray();

switch (commands[0])

{

case "Dice":

string player = commands[1];

int value1 = int.Parse(commands[2]);

int value2 = int.Parse(commands[3]);

if (players.ContainsKey(player))

{

players[player].Add(new Pair(value1, value2));

}

else

{

players.Add(player, new CapacityList(capacity));

players[player].Add(new Pair(value1, value2));

}

break;

case "CurrentPairSum":

string playerName = commands[1];

if (players.ContainsKey(playerName))

{

Console.WriteLine(players[playerName].Sum());

}

break;

case "CurrentStanding":

players = players.OrderBy(element => element.Value.Sum().Difference())

.ToDictionary(element => element.Key, element => element.Value);

foreach (var item in players)

{

Console.WriteLine(item.Key + " - " + item.Value.Sum());

}

break;

case "CurrentState":

string playerNameState = commands[1];

if (players.ContainsKey(playerNameState))

{

players[playerNameState].PrintCurrentState();

}

break;

case "Winner":

players = players.OrderBy(element => element.Value.Sum().Difference())

.ThenBy(element => element.Key)

.ToDictionary(element => element.Key, element => element.Value);

Console.WriteLine("{0} wins the game!", players.First().Key);

break;

}

} while (command != "End");

}

}

}