5ªaula prática

Estruturas de dados lineares: listas

- Faça download do ficheiro ***aed2223_p05.zip*** da página e descomprima-o (contém a pasta ***lib***, a pasta ***Tests*** com os ficheiros ***funListProblem.h***, ***funListProblem.cpp***, ***game.cpp, game.h, kid.cpp, kid.h*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)
- No CLion, abra um ***projeto***, selecionando a pasta que contém os ficheiros do ponto anterior.

1. Implemente a função:

list<int> FunListProblem::removeHigher(list<int> &values, int x)

Esta função remove da lista *values* os elementos superiores a *x* (exclusivé), e retorna uma nova lista com os elementos que foram removidos, de acordo com a ordem da remoção. Os elementos que permanecem na lista *values* devem manter a sua posição relativa.

Complexidade temporal esperada: $O(n)$

Exemplo de execução:
input*:* *values* = {7, 8, 12, 5, 2, 3, 5, 6} ; *x* = 5
output*:* *result* = {7, 8, 12, 6} ; *values* = {5, 2, 3, 5}

2. Implemente a função:

list<pair<int,int>> FunListProblem::overlappingIntervals(list<pair<int,int>> values)

Dada uma coleção de intervalos (lista *values*), a sua tarefa é juntar todos os intervalos sobrepostos.
Um intervalo é identificado por um par (*pair<int,int>*). Assim, o intervalo com início em 3 e final em 7 é identificado pelo par <3,7>. Considere que o início do intervalo é sempre não superior ao final do intervalo.
Sugestão: Comece por ordenar a lista *values*.

Complexidade temporal esperada: $O(n \times \log n)$

Exemplo de execução:
input*:* *values* = {<1,3>, <6,8>, <2,4>, <9,10>}
output*:* *result* = {<1,4>, <6,8>, <9,10>}

input*:* *values* = {<6,8 >, <1,9>, <2,4>, <4,7>}

output: *result* = {<1,9>}

3. "*Pim Pam Pum cada bola mata um pra galinha e pro peru quem se livra és mesmo tu*". Recorde este jogo de crianças, cujas regras são as seguintes:

- A primeira criança diz a frase, e em cada palavra vai apontando para cada uma das crianças em jogo (começando em si). Ao chegar ao fim da lista de crianças, volta ao início, ou seja, a ela mesma.
- A criança que está a ser apontada, quando é dita a última palavra da frase, livra-se e sai do jogo. A contagem recomeça na próxima criança da lista.
- Perde o jogo a criança que restar.

Use uma lista (vamos usar a classe *list* da STL) para implementar este jogo. Os elementos da lista são objetos da classe **Kid** (ver ficheiro *kid.h*).

```
class Kid {
    string name;
    unsigned age;
    char sex;
public:
    Kid();
    Kid(string nm, unsigned a, char s);
    Kid(const Kid &k1);
    //…
};
```

A classe **Game** também está definida:

```
class Game
{
    list<Kid> kids;
public:
    Game();
    Game(list<Kid>& l2);
    static unsigned numberOfWords(string phrase);
    Kid loseGame(string phrase);
    list<Kid> rearrange();
    list<Kid> shuffle();
    //…
};
```

3.1 Implemente o membro-função:

Kid& Game::loseGame(string phrase)

Esta função realiza o jogo de acordo com as regras enunciadas, quando a frase utilizada é *phrase* e retorna a criança que perde o jogo.

Use o membro-função *numberOfWords* (já fornecido) que determina o número de palavras existentes numa frase indicada em parâmetro:

int Game::numberOfWords(string phrase)

Complexidade temporal esperada: $O(n^2)$

Exemplo de execução:

input: *phrase* = "Pim Pam Pum Pim" , *kids* = {"Rui", "Ana", "Rita", "Joao", "Marta", "Vasco"}

// *kids* é uma lista de objetos *Kid*, aqui representada apenas pelo nome da criança, por simplificação

output*:* *result* = "Marta"

explicação:

- na 1ªronda sai o 4ºelemento (frase com 4 palavras), que é o "Joao"
- na 2ªronda, a contagem começa a seguir, na "Marta". Sai a "Ana"
- na 3ªronda, a contagem começa a seguir, na "Rita". Sai o "Rui"
- na 4ªronda, a contagem começa a seguir, na "Rita". Sai a "Rita"
- na 5ªronda, a contagem começa a seguir, na "Marta". Sai o "Vasco"
- resta a "Marta", que perde o jogo

3.2 Implemente o membro-função:

lista<Kid> Game::rearrange()

Esta função altera a disposição (e eventualmente número) de crianças em jogo, distribuindo as meninas e meninos ao longo da lista. As crianças são dispostas repetindo um mesmo padrão relativo ao sexo da criança. Assim, se existirem **n** meninas e **m** meninos: i) sendo **n**<**m**, o padrão será constituído **1** menina e **m**/**n** meninos; ii) sendo **n**>=**m**, o padrão será constituído **n**/**m** meninas e **1** menino. Deve ser mantida a disposição relativa dos meninos entre si, e a disposição relativa das meninas entre si. A primeira criança da lista é do sexo feminino. As crianças que restam são guardadas numa fila que é retornada pela função. Sugestão: use duas estruturas de dados auxiliares, uma para colocar as meninas e outra para colocar os meninos.

Complexidade temporal esperada: $O(n)$

Exemplo de execução:

input*:* *kids* = {"Rui", "Ana", "Maria", "Joao", "Vasco", "Luis"}

// *kids* é uma lista de objetos *Kid*, aqui representada apenas pelo nome da criança, por simplificação

output*:* *result* = {} ; *kids* = {"Ana", "Rui", "Joao", "Maria", "Vasco", "Luis"}

Explicação: a lista *{"Rui", "Ana", "Maria", "Joao", "Vasco", "Luis"}* , origina o padrão <*menina, menino, menino*>, ficando a lista final igual a *{"Ana", "Rui", "Joao", "Maria", "Vasco", "Luis"},* não restando nenhuma criança.

3.3 Implemente o membro-função:

list<Kid> Game::shuffle() const

Esta função cria uma nova lista onde as crianças do jogo são colocadas em uma posição determinada aleatoriamente (use a função *rand()* para gerar números aleatórios).