5ªaula prática - Estruturas de dados lineares: listas, pilhas e filas

Instruções

- Faça download do ficheiro ***aed2122_p05.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***game.cpp, game.h, kid.cpp, kid.h, stackExt.h, performance.h, cycle.h*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)

1. Deve realizar a primeira alínea (alínea a) antes das restantes, que são independentes entre si.

"*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 (pode 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);
    unsigned getAge() const;
    string getName() const;
    char getSex() const;
    string write() const;
};
```

A classe **Game** também está definida:

```
class Game
{
    list<Kid> kids;
public:
    Game();
    Game(list<Kid>& l2);
    static unsigned numberOfWords(string phrase);
    void addKid(const Kid k1);
    list<Kid> getKids() const;
    string write() const;
    Kid loseGame(string phrase);
    list<Kid> removeOlder(unsigned a);
    void setKids(const list<Kid>& l2);
    bool operator==(Game& g2);
    list<Kid> shuffle();
};
```

a) Implemente os construtores da classe **Game** e ainda os membros-função:

void Game::addKid(const Kid k1)

Esta função adiciona a criança *k1* ao jogo.

list<Kid> Game::getKids() const

Esta função retorna a lista de crianças atualmente em jogo.

void Game::setKids(const list<Kid>& l1)

Esta função coloca em jogo a lista de crianças *l1*.

b) Implemente o membro-função:

Kid& Game::loseGame(string phrase)

Esta função realiza o jogo enunciado 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)

c) Implemente o membro-função:

queue<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** meninos. 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 filas auxiliares, uma para colocar as meninas e outra para colocar os meninos.
Ex: a lista *{rui, ana, maria, joao, vasco, luis, hugo, pedro, miguel, rita}*, origina o padrão *<menina, menino, menino>*, ficando a lista final igual a *{ana, rui, joao, maria, vasco, luis, rita, hugo, pedro}* e sendo criada a fila *{miguel}* com a criança que resta.

d) Implemente o membro-função:

list<Kid> Game::removeOlder(unsigned id)

Esta função remove do jogo as crianças de idade superior ao valor *id*, e retorna uma nova lista com as crianças que foram removidas. As crianças que ficam em jogo devem manter a sua posição relativa na lista *kids*.

e) Implemente o operador de igualdade:

bool Game::operator==(Game& g2)

Dois jogos são considerados iguais se possuem nas suas respetivas listas as mesmas crianças, na mesma ordem.

f) 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).

2. Implemente a classe **StackExt**, que implementa uma estrutura do tipo pilha (*stack*) com um novo método: *findMin*. O método *findMin* retorna o valor do menor elemento da pilha e deve ser realizado em tempo constante, O(1). Deve implementar os seguintes métodos:

```
bool empty() const          // verifica se a pilha está vazia
T& top()                    // retorna o elemento no topo da pilha
void pop()                  // remove elemento
void push(const T& val)     // insere elemento
T& findMin()                // retorna valor do menor elemento
```

Sugestão: Use a classe *stack* da biblioteca STL. Use duas stacks: uma para guardar todos os valores, e outra para guardar os valores mínimos à medida que estes vão surgindo.

Nota: os testes unitários podem ser demorados.