Ficha 8

Programação (L.EIC009)

Objectivos

- Exercícios versando a Standard Template Library (STL).

Recursos

- Slides das aulas teóricas: HTML PDF
- Documentação da STL
  - "Containers library"
  - "Iterators library"
  - "Algorithms library"

1

Considere a leitura de todas as palavras de um ficheiro de texto com o seguinte esqueleto:

```
#include <fstream>
#include <string>
#include <iostream>

int main(int argc, char**argv) {
    std::ifstream in(argv[1]);
    while (true) {
        std::string word;
        in >> word;
        if (word.empty()) break;
        ...
    }
    ...
    return 0;
}
```

1.1

Adicionando as palavras lidas do ficheiro a um "container" `std::vector`, no fim imprima todas as palavras na mesma ordem em que foram lidas.

1.2

Escreva uma variação do programa anterior, imprimindo as palavras na ordem inversa em que foram lidas. Use um "reverse" iterator (i.e., empregue `rbegin()` e `rend()`).

1.3

Escreve um programa que imprime agora as palavras lidas por ordem alfabética. Use `std::sort` para ordenar o vector.

1.4

Escreva uma variação do programa anterior, que imprime as palavras por ordem alfabética mas eliminando duplicados. Use a função `std::unique`).

Nota: `unique` não irá remover elementos de um vector, mas sim arrumar os elementos de tal forma que para um "container" `c` se

```
u_end = std::unique(c.begin(), c.end());
```

então os elementos sem repetições consecutivas ficam entre `c.begin()` inclusive e `u_end` exclusive.

1.5

Para imprimir os elementos de forma ordenada e sem repetições podemos também considerar o uso de um "container" `set` (header file `<set>`). Esta escolha é adequada visto que um objecto `set` representa um conjunto de elementos (não armazena duplicados portanto) e além disso garante uma iteração ordenada de elementos. Pode empregar a função membro `insert` para adicionar um elemento a um objecto `set`.

2

Uma média deslizante ("moving average") calcula o valor médio de uma série de medidas ao longo do tempo, com uma janela temporal associada para o cálculo da média. O conceito é bastante usado em vários domínios; por ex. tornou-se popular nos

últimos tempos falar da média de casos COVID dos últimos 7 dias :(

Programe a seguinte classe para cálculo de média deslizante usando internamente um "container" `std::list` ou `std::deque`.

```
...
class MovingAverage {
private:
  ...
public:
  MovingAverage(int window_size);
  void update(int value);
  int current() const;
};
```

- O parâmetro de construção `window_size` indica o tamanho da janela para cálculo da média, i.e. o nº máximo de medidas para cálculo da média.
- Uma chamada a `update(v)` oferece a medida `v` mais recente. O valor `v` deve ser adicionado ao "container" usado. Por sua vez o valor mais antigo deve ser descartado caso o número de elementos exceda `window_size`. Na sua implementação considere o uso das funções membro `push_back()` e `pop_front()`.
- Uma chamada a `current()` devolve o valor actual da média deslizante (tendo em conta os valores guardados).

Exemplo de evolução com `window_size=3`

(Sugere-se a escrita de um programa para validar a implementação.)

Passo	**`update`**	**`current()`**
1	30	30
2	32	31
3	28	30
4	12	24
5	28	22
6	27	22
7	26	27
8	20	24

9	30	25
10	40	30

2.2

Adicione a `MovingAverage` a função membro:

```
void fix_values(int min,int max)
```

Uma chamada à função deve acerte os valores armazenados no "container" tal que: - valores inferiores a `min` devem ser convertidos em `min` ; - e valores superiores a `max` devem ser convertidos em `max` .

Na sua implementação faça uso de um iterador sobre o "container" dos valores.

3

3.1

Escreva as seguintes funções template que operam sobre "containers" `deque` :

```
template <typename T>
void remove_all(std::deque<T>& dq, const T& v) { ... }

template <typename T>
void reverse(std::deque<T>& dq) { ... }
```

tal que:

- `remove_all(dq,v)` remova de `dq` todas as ocorrências de `v` - use um iterador sobre `dq` em combinação com `erase` ;
- `reverse(dq)` : deverá inverter a ordem dos elementos em `dq` - use dois iteradores simultâneos (um a começar no início, outro no fim) e troca de valores entre estes, i.e., algo como:

```
auto itr1 = dq.begin();
auto itr2 = -- dq.end();
while (itr1 < itr2) {
  ...
  itr1++;
  itr2--;
```

```
}
```

(o operador `<` detecta quando os iteradores se cruzam; porque `==` pode não funcionar?)

(`dq.rbegin()` não pode ser empregue: reverse iterators não podem ser comparados com iteradores "standard")

3.2

Codifique as mesmas funções para containers `list`. Será que é só trocar o tipo de `deque` para `list` ... ?

4

Escreva uma programa semelhante aos do exercício **1.5**, mas que imprima não só as palavras no ficheiro de input de forma ordenada, como o número total de ocorrências para cada palavra. Use um objecto `map<std::string, int>` para associar cada palavra lida ao correspondente número de ocorrências.

4.1

Empregue para o efeito as função membros `find` e `insert` de `map`. Esqueleto:

```
std::map<std::string,int> word_count;
...
std::string word;
...
auto itr = word_count.find(word);
if (itr == word_count.end()) {
   // Não existe, insere nova entrada no mapa
   word_count.insert( ... );
} else {
   // Actualiza (*itr).second
   ...
}
```

Depois no final, só precisa de iterar o mapa para imprimir os pares chave-valor (palavra, nº de ocorrências).

4.2

Numa solução mais simples tire partido do facto de `operator[]` criar se necessário uma chave (quando chave dada não existe ainda) e o valor inicial associado ser 0.

5

Considere dados referentes a jogos de futebol com o seguinte formato:

```
6
Liverpool Chelsea 4 2
Leeds Arsenal 2 1
Arsenal Liverpool 4 5
Chelsea Leeds 0 1
Liverpool Leeds 0 0
Chelsea Arsenal 3 1
```

O formato começa com o número `n` de jogos a considerar, seguidos por informação para os `n` jogos. A informação para cada jogo identifica o nome de duas equipas, e os golos correspondentes marcados por cada uma das equipas (ex. o primeiro jogo acima refere-se ao jogo entre Liverpool e Chelsea em que o Liverpool ganhou por 4 golos a 2).

5.1

Defina um tipo `struct Game` para representar a informação de um jogo e escreva um programa que leia dados de todos jogos com o formato acima para um objecto `vector<Game>`. No final o programa deve ordenar o vector - use `std::sort` para a ordenação - e depois imprimir os dados dos jogos (já de forma ordenada) considerando os seguintes critérios de ordenação:

1. jogos ordenados pelo nome da 1ª equipa e depois pelo nome da 2ª equipa:

   ```
   Arsenal Liverpool 4 5
   Chelsea Arsenal 3 1
   Chelsea Leeds 0 1
   Leeds Arsenal 2 1
   Liverpool Chelsea 4 2
   Liverpool Leeds 0 0
   ```

2. jogos ordenados pelo número total de golos marcados de forma crescente:

```
Liverpool Leeds 0 0
Chelsea Leeds 0 1
Leeds Arsenal 2 1
Chelsea Arsenal 3 1
Liverpool Chelsea 4 2
Arsenal Liverpool 4 5
```

3. a ordem inversa à anterior:

```
Arsenal Liverpool 4 5
Liverpool Chelsea 4 2
Chelsea Arsenal 3 1
Leeds Arsenal 2 1
Chelsea Leeds 0 1
Liverpool Leeds 0 0
```

6

Escreva um programa que leia dados de jogos de futebol com o mesmo formato que no exercício anterior e que no final imprima a classificação das equipas com a seguinte forma:

```
Liverpool 7 3
Leeds 7 2
Chelsea 3 -1
Arsenal 0 -4
```

Cada linha deverá conter o:

- nome de uma equipa;
- o número de pontos para a equipa - cada vitória num jogo vale 3 pontos, um empate 1, e uma derrota 0;
- e o saldo entre golos marcados e golos sofridos para a equipa.

As equipas devem ser listadas de forma decrescente pelo nº de pontos, e para equipas com os mesmos pontos pelo melhor saldo de diferença de golos. Não iremos considerar casos em que as equipas também estão empatadas tanto em pontos como em saldo de golos.

Sugestão de resolução:

- crie `struct TeamData` para guardar dados relevantes a cada equipa: nome

da equipa, número de pontos, saldo entre golos marcados e sofridos.

- use `map<std::string,TeamData>` ou `unordered_map<std::string,TeamData>` para ir actualizando os dados de equipas à medida que lê os dados de cada jogo;
- no final agrupe os elementos `TeamData` num objecto `vector<TeamData>` e ordene-o em função dos critérios de classificação.