PROJECTO 2
BIBLIOTECA EASYSTD E POO

INTRODUÇÃO E OBJECTIVOS

A finalidade deste projecto passa por desenvolver uma biblioteca pessoal de utilitários, necessários em diversas circunstâncias e, de caminho, consolidar os conhecimentos de programação em C/C++. Além disso, deve também desenvolver um programa para gestão de um catálogo de viaturas de um *stand* automóvel.

Estes elementos extra são resumidos na secção **AVALIAÇÃO**. Nesta secção encontra também uma explicação sobre o que se entende por "extra" no contexto deste projecto.

PARTE I - FUNÇÕES DE BIBLIOTECA

A biblioteca **EasySTD** é uma biblioteca de funções de utilidade geral, que abrange temas como manipulação de texto, entradas e saídas, algoritmos e estruturas de dados, aleatoriedade/*randomness*, data/hora, etc. O objectivo desta biblioteca passa por simplificar o desenvolvimento em C++, fornecendo mecanismos que, ou não encontramos de todo na biblioteca padrão do C++, ou que integram esta bibioteca, mas cuja interface é complexa e demasiado complicada para os iniciados na linguagem C++, e, frequentemente, são mais difíceis de utilizar do que os mecanismos equivalentes noutras linguagens.

C e C++ são linguagens utilizadas em programação de sistemas, ou seja, em cenários onde o bom desempenho e a gestão criteriosa dos recursos do computador são requisitos muito importantes. Todavia, na biblioteca **EasySTD**, os requisitos principais são: utilidade, simplicidade, facilidade de implementação e legibilidade do código. No âmbito desta bibilioteca, bom desempenho passa a ser um requisito secundário.

A biblioteca deve abordar apenas cinco temas: Utilitários Gerais, Texto, Aleatoriedade, I/O e Datas. Todas as definições desta biblioteca devem integrar o *namespace* `easy_std`. Pode desenvolver uma biblioteca "tradicional", com arquivo cabeçalho `.hpp` e respectivos arquivos `.cpp`, ou desenvolver uma biblioteca *header-only* apenas em arquivos cabeçalhos.

De seguida listamos as funções a desenvolver, indicando a sua assinatura seguida de uma breve especificação.

Módulo `easy_utils.hpp` - Utilitários Gerais

1. `string` *`input`*`(const string& msg = ""s)`

   Exibe uma mensagem na saída padrão e lê uma linha de texto a partir da entrada padrão. A linha de texto é devolvida.

2.
```
template<typename T>
string to_string(const T& t)
```

Converte um objecto do tipo parametrizado `T` numa string. Deve utilizar as facilidades definidas em `sstream` (`#include <sstream>`).

Desenvolva especializações deste template para colecções do tipo `std::vector` e `std::list` e para `std::map`.

3.
```
template<typename T>
T convert(const string& str)
```

Converte uma string num um objecto do tipo parametrizado T. Deve utilizar as facilidades definidas em `sstream`.

Exemplos:
```
auto i = convert<int>("123");   // i é um int com valor 123
auto idade = convert<int>(input("Qual a sua idade? "));
```

4.
```
template<typename Container>
void print(const Container& cont, const print_params& p = prints_params())

template<typename Container>
void print(const Container& cont, ostream& out, const print_params& p = prints_params())

struct print_params {
    string sep{" "};
    string end{"\n"};
};
```

Exibe o conteúdo da colecção `cont` (o *container* ; a tradução literal para Português seria *contentor*) separando os elementos da colecção com o separador `sep` e terminando a exibição com o terminador `end`. `cont` é uma colecção de elementos como um `std::vector`, uma `std::string`, um `std::array`, etc. Ou seja, possui uma função membro `size` e é possível obter dois iteradores com `begin()` e `end()`. Este template aceita qualquer tipo de dados para os elementos da sequência. Por exemplo, as seguintes sequências devem ser aceites: `std::vector<int>`, `std::vector<double>`, `std::string`, `std::ustring16_t` (isto é, `std::basic_string<char16_t>`), etc.

Exemplos:

`vector<int> vals{10, 50, 23};`
`print(vals);`
`print(vals, {.sep = "//"});`
`print(vals, {.end = "$$"});`
`print(vals, {.sep = "--" , .end = "FIM\n"});`
`print("Alberto");`
`print("Alberto", {.sep = ""});`
	`10 50 23`
	`10//50//23`
	`10 50 23$$10--50--23FIM`
	`A l b e r t o`
	`Alberto`

Através do parâmetro `out` é possível escrever, por exemplo, num ficheiro:

```
ofstream file("dados.txt");
print(vals, file, {.sep = ","});   // escreve 10,50,23 no ficheiro dados.txt
```

extra: Desenvolva uma especialização para `initializer_list<T>`.

5. `template<typename Container, typename Item>` **(extra)**
`int` *`find_index`*`(const Container& cont, const Item& item, int start = 0)`

Localiza o `item` em `cont`, uma colecção do tipo sequência. Devolve o indíce do elemento caso ele exista, ou -1, caso contrário. O indíce é um valor entre `0` e `cont.size() - 1` (inclusive). O `item` é comparado por meio do operador ==. `start` indica que a pesquisa deve começar a partir deste valor inclusive.

Exemplos:

```
vector<int> vals{-5, 10, 31, 55, 10, 44};
find_index(vals, 10)        => 1
find_index(vals, 10, 2)     => 4
find_index(vals, -10)       => -1
find_index("Alberto"s, 'b') => 2
```

6. `template<typename Container, typename Item>` **(extra)**
`bool` *`in`*`(const Container& cont, const Item& item)`

Indica se o elemento está presente na colecção `cont`. O `item` é comparado por meio do operador ==.

(extra) Implemente uma especialização deste template para `std::map` e `std::unordered_map`. Em ambos os casos, a função `easy_std::in` indica se a chave faz parte do mapa.

Exemplos:

```
vector<int> vals{-5, 10, 31, 55, 10, 44};
in(vals, 10)        => true
in(vals, -10)       => false
in("Alberto"s, 'b') => true
in({"ana", "bruno", "ze"}, "bruno") => true

map<string, int> portos = {{"http", 80}, {"ftp", 21}, {"telnet", 25}};
in(portos, "ftp"s}    => true
```

```
in(portos, "smtp"s}  => false
```

Módulo `easy_text.hpp` - Texto: Strings e Caracteres

7. `bool` *`is_white_space`*`(char ch)`

Devolve `true` se o caractere `ch` for um caractere de espaçamento, ou seja, se for um dos seguintes: `' '`, `'\n'`, `'\t'` ou `'\r'`).

8. `bool` *`is_white_space`*`(const string& str)`

Devolve `true` se todos os caractere da string `str` forem de espaçamento. Deve utilizar a versão para um caractere.

Exemplo:

```
is_white_space("  \n\t ")   => true
is_white_space("  \n\t3 ")  => false
```

9. `bool` *`is_digit`*`(char ch)`

Devolve `true` se o caractere `ch` for um dígito decimal.

10. `bool` *`is_digit`*`(const string& str)`

Devolve `true` se todos os caractere da string `str` forem dígitos. Deve utilizar a versão para um caractere.

11. `char` *`to_lower`*`(char ch)`

Converte uma letra maiúscula para minúscula; implemente a versão complementar *`to_upper`*.

12. `string&` *`to_lower`*`(string& str)`

Converte uma letra maiúscula para minúscula; implemente a versão complementar *`to_upper`*. Esta função modifica a string recebida, mas também devolve-a para que possa ser utilizada em expressões do tipo `string`.

13. `string&` *`trim_left`*`(string& str)`

Apara a string `str` à esquerda e devolve-a; implemente a versão complementar *`trim_right`*. "Aparar" uma string à esquerda consiste em remover os caracteres de espaçamento à esquerda, que estão entre o início da string e o primeiro caractere que não é de espaçamento. Esta função modifica a string recebida, mas também devolve-a para que possa ser utilizada em expressões do tipo `string`. Se a string não possuir caracteres de espaçamento à esquerda, então é simplesmente devolvida.

14. `string& trim(string& str)`

Apara a string `str` à esquerda e à direita devolvendo essa string.

15. `string& reverse(string& str)`

Inverte a string `str` e devolve-a. Tal como com as funções trim, a string `str` é modificada mas também é devolvida. Desenvolva o algoritmo de raiz, sem recorrer a quaisquer funções de `algorithm` (como, por exemplo, `std::reverse` ...) ou de qualquer outra biblioteca.

16. `string reversed(const string& str)`

Devolve uma cópia invertida da string `str` com os caracteres por ordem inversa. Pode utilizar funções já feitas.

17. `string& replace(string& str, char ch1, char ch2, int start = 0, int end = -1)`

Substitui todas as ocorrências do caractere `ch1` pelo caractere `ch2`; devolve a string `str`. Começa na posição dada por `start` e substitui até à posição dada por `end`. O valor -1 indica que end deve ser a posição do último caractere.

Exemplos:

```
replace("ana avelar", 'a', 'A')       => "AnA AvelAr"
replace("ana avelar", 'a', 'A', 4)    => "ana AvelAr"
replace("ana avelar", 'a', 'A', 0, 2) => "AnA avelar"
replace("ana avelar alves", 'a', 'A', 4, 9) => "ana AvelAr alves"
replace("ana avelar", 'b', 'A')       => "ana avelar"
```

18. `string& replace(string& str, const string& substr1, const string& substr2, int start = 0, int end = -1)`

Substitui em `str` todas as ocorrências da substring `substr1` por `substr2`; devolve a string `str`. *Em baixo não indicamos exemplos com os parâmetros `start` e `end` porque o comportamento da função pode ser inferido a partir da versão anterior.*

extra: implemente o algoritmo de raiz, sem recorrer a quaisquer funções que implementem total ou parcialmente esta funcionalidade ou funcionalidade similar (como `std::string::replace`, `std::replace`). Pode, no entanto, utilizar `std::find`, `std::string::find` e as funções membro de `std::string` que permitem modificar a string, como `std::string::insert`, `std::string::erase` e `std::string::clear`.

Exemplos:

```
replace("ana mariana", "ana", "ANA")     => "ANA mariANA"
replace("ana mariana", "ana", "anabela") => "anabela marianabela"
replace("diana mariana", "ana", "")      => "di mari"
```

19. `vector<string> split(const string& str, const string& delim)`

"Parte" uma string em pedaços delimitados por `delim`. Dada a string `str` com `"ABC DEF GHI"`, e `delim` igual a `" "` , a função *split* devolve um vector com `{"ABC", "DEF", "GHI"}`.

Outros exemplos:

```
split("ABC.DEF.GHI", ".")       =>  {"ABC", "DEF", "GHI"}
split("ABC.DEF.GHI", "+")       =>  {"ABC.DEF.GHI"}
split("ABC.DEF...GHI", ".")     =>  {"ABC", "DEF", "", "", "GHI"}
split(".ABC.DEF.GHI.", ".")     =>  {"", "ABC", "DEF", "GHI", ""}
split("ABC--DEF--GHI", "--")    =>  {"ABC", "DEF", "GHI"}
split("-ABC---DEF--GHI-", "--") =>  {"-ABC", "-DEF", "GHI-"}
split("ABCDEF", "")             =>  {"A", "B", "C", "D", "E", "F"}
```

20. `string join(const vector<string>& parts, const string& delim)`

O oposto de split. "Une" as várias strings no vector `parts` com o delimitador `delim` e junta-as todas numa só string. Dado o vector `parts` com `{"ABC", "DEF", "GHI"}` e `delim` igual a `"+"` , a função *join* devolve a string `"ABC+DEF+GHI"`.

Outros exemplos:

> ***NOTA***: *nos exemplos seguintes, utilizamos a notação {"ABC", "DEF", etc.} como notação abreviada para vector<string>{"ABC", "DEF", etc.}*

```
join({"ABC", "DEF", "GHI"}, "+")      => "ABC+DEF+GHI"
join({"ABC"}, ".")                    => "ABC"
join({"ABC", "DEF"}, ".")             => "ABC.DEF"
join(split("ABC.DEF.GHI", "."), "+")) => "ABC+DEF+GHI"
```

Módulo `easy_random.hpp` - Aleatoriedade/Randomness

Baseando-se nos seguintes exemplos

. http://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution

. http://en.cppreference.com/w/cpp/numeric/random/uniform_real_distribution

. http://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine/seed

pretende-se que desenvolva um conjunto de funções para gerar números aleatórios e implementar funcionalidades relacionadas. Deve definir as seguintes funções e variáveis:

NOTA*: Será fornecido o código para as primeiras quatro definições. Como tal, estas não contam para a avaliação.*

21. `static std::mt19937 rand_gen`

Variável global (interna) que representa o gerador de números aleatórios global.

22. `int` *`randint`*`(int a, int b)`

Devolve um inteiro entre a e b inclusive, de acordo com uma distribuição uniforme

23. `double` *`random`*`(double a = 0.0, double = 1.0)`

Devolve um número double aleatório entre `a` e `b`. Por omissão, `a = 0.0` e `b = 1.0`.

24. `void` *`seed`*`(int value)`

Fornece um novo valor semente para o gerador de número aleatório `rand_gen`.

25.
```
template<typename Seq>
auto choice(const Seq& seq)
```

Escolhe aleatoriamente um elemento da sequência `seq`.

26.
```
template<typename Seq>
void shuffle_(Seq& seq)
```

"Baralha" aleatoriamente os elementos da sequência `seq`.

NOTA: *Não utilize a função `std::shuffle` da biblioteca `algorithm` (`#include <algorithm>`).*

27.
```
template<typename T, typename Seq>
vector<T> sample(const Seq& seq, int n):
```

Selecciona uma amostra de `n` valores da sequência `seq`. Os resultados são devolvidos num vector do tipo da variável de retorno ou do tipo de chamamento.

Módulo `easy_date.hpp` - Representação Simplificada de Datas e DataHora

Até à versão 20, o C++ não possuia nenhum tipo de dados para manipular datas de forma conveniente. A partir desta versão, o C++ passou a incluir a biblioteca `date.h` desenvolvida por Howard Hinnant. Esta biblioteca, que foi integrada no módulo `chrono` (desenvolvido também pelo mesmo Howard Hinnant), é extremamente poderosa, versátil e eficiente, mas é igualmente complicada de usar, principalmente por iniciados. Sendo assim, neste módulo vamos definir as classes `LocalDate` e `LocalDateTime` que deverão oferecer uma interface mais simples para representar um data/data-hora tal como expressa por um calendário de parede (ie, vamos ignorar fusos horários). Deve suportar, pelo menos as seguintes operações:

TBD.

Estas classes deverão ser definidas a partir dos mecanismos do módulo chrono.

Módulo `easy_io.hpp` - **Utilitários para I/O**

Aqui pretende-se que desenvolva um conjunto de tipos de dados e utilitários para facilitar a leitura e escrita de streams binárias e de texto.

TBD.

Considerações Gerais

Pode desenvolver uma biblioteca com separação entre declarações e definições/implementações, as primeiras no ficheiro `easy_???.hpp`, e as segundas no ficheiro `easy_???.cpp`.

Como alternativa, pode desenvolver uma *header-only library* com definições e, opcionalmente, com declarações. Neste caso, cada módulo da biblioteca (declarações mais definições/implementações) deverá ficar no ficheiro `easy_???.hpp` respectivo. Não se esqueça de prefixar as definições/implementações com a palavra-reservada `inline`. Além do mais, neste tipo de biblioteca não se recomenda a utilização da directiva **`using namespace`** no âmbito global, apenas dentro de funções ou classes.

Em ambos os casos, deve também criar o ficheiro `test_easy_std.cpp` contendo um `main` para testar devidamente os diversos módulos da sua biblioteca.

Não deve utilizar mecanismos da linguagem C (nem da respectiva biblioteca padrão; isto quer dizer que não pode utilizar strings de C, *arrays*, funções como `strcmp`, `strlen`, `atoi`, `atof`, `isupper`, etc.). A não ser onde explicitamente proibido, pode utilizar funcionalidades da biblioteca padrão do C++, como, por exemplo, funções de `algorithm`, métodos de `string` ou de `vector`.

PARTE II – GESTÃO DE VIATURAS

Aqui pretende-se que termine o exemplo para gestão de viaturas – **CarPP** - desenvolvido durante a formação. Deve concluir todas as funcionalidades que ficaram por implementar. Além disto deve incluir opções para actualizar os dados de uma viatura, bem como a funcionalidade para gravar o catálogo em suporte persistente.

A colecção de viaturas deve suportar pesquisas genéricas. Como extra, pode tornar a colecção de viaturas iterável, de modo a que a possa iterar através de um ciclo FOR-each. Pode definir um workspace à parte no VS Code para este exemplo, ou pode utilizar o workspace para a bilbioteca **EasySTD** (ver em baixo), criando então uma directoria **`viaturas`** dentro da directoria **`src`**. Caso defina um Worspace à parte, deve inclui-lo no ZIP em baixo indicado. Este workspace deve possuir estrutura idêntica à do criado para a biblioteca **EasySTD**.

AVALIAÇÃO

O projecto deve ser resolvido em grupos de dois formandos.

Elemento	Cotação Máxima (0..20)
Parte I – Easy STD	14
Parte II – Gestão de Viaturas	6

TODO: DEVE ENTREGAR UM RELATÓRIO CUJO PESO NA COTAÇÃO FINAL DO PROJECTO SERÁ DE X%.

PRAZOS E ENTREGAS

O trabalho final deve ser entregue até às 23h59m do dia 29/03/2024. Um atraso de N dias na entrega levará a uma penalização dada pela fórmula $0.5 \times 2^{(N-1)}$ ($N > 0$).

Nesta data deverá entregar o seguinte:

1. Um **ZIP** contendo o projecto do VS Code. O ZIP deve possuir os nomes dos membros do grupo sem acentos e separados por _ (underscore). Supondo que o projecto foi realizado pela Ana Alves e pelo Jose Sá, o ZIP deve ser designado de `AnaAlves_JoseSa.zip`.

   Este ZIP deve conter uma única directoria:

   `Projecto2`: directoria com o Workspace do VS Code.

   1.1 Por seu turno a directoria `Projecto2` deve conter as seguintes sub-directorias:

   `.vscode`: directoria com as definições do VS Code

   `src`: directoria com o código fonte

   `docs`: documentação do projecto mais o relatório (ver em baixo)

   `libs`: directoria opcional com bibliotecas extra utilizadas neste projecto (eg, docopt)

   - `lib`: módulos compilados de biblioteca (eg, `*.a`, `*.dll`, `*.lib`, `*.dylib`)
   - `include`:arquivos cabeçalhos de biblioteca (eg, `*.h`, `*.h++`, `*.hpp`)

   Não necessita da directoria `libs` se não utilizar nenhuma biblioteca externa, ou caso as instale numa localização conhecida pelo compilador.

   1.2 O conteúdo da directoria `/src` deve ser o seguinte:

`easy_std/test_easy_std.cpp` ☒ *contendo a função* ***main*** *para testes*
`easy_std/easy_utils.hpp` ☒ *pode conter toda a bilioteca (header only lib.) ou apenas delcs. e templates*
`easy_std/easy_text.hpp` ☒ *utilitários para texto (strings e chars)*
`easy_std/easy_random.hpp` ☒ *utilitários aleatoriedade*
`[viaturas/*.cpp` ☒ ficheiros relevantes para gestão de viaturas]

2. Relatório em PDF, cuja estrutura e formatação deverão ser a do modelo fornecido em anexo. O PDF com o relatório deve ser colocado na directoria `docs` em cima mencionada (ou seja, o relatório segue dentro do ZIP). Em termos de formatação adapte apenas a designação da acção e o nome dos módulos. Siga as recomendações relacionadas com a elaboração de um relatório dadas pelo formador Fernando Ruela. O relat¿rio deve incluir uma capa simples com o s¶mbolo do IEFP, refer´ncia ao Centro de Forma¯-o de Alc«ntara, data, indica¯-o do curso e da ac¯-o (eg, T° cnico de Inform@tica - Sistemas 06) e dos elementos que elaboraram o trabalho.

Em termos de conteúdo o seu relatório deve possuir as seguintes secções e anexos:

2.1 Introdução e Objectivos (não plagiar a deste enunciado!).

2.2 Implementação. Deve explicar de forma breve, e por palavra suas, a implementação de cada uma das funções e classes/estruturas desenvolvidas. Onde necessário, poderá elaborar fluxogramas, diagramas de actividade, diagramas de estado, etc, para melhor descrever os algoritmos utilizados. Esta secção deve ter uma sub-secção para cada parte do projecto.

A sec¯-o Implementação deve indicar aspectos de implementação que considere relevantes. É também aqui que deve dar instruções sobre como compilar o seu programa (num programa maior poderia ser pertinente criar uma sec¯-o Compilação e Instalação) e produzir um executável. Deve indicar para que plataforma desenvolveu e testou o seu programa, a versão do compilador e das bibliotecas utilizadas.

2.3 Conclusão. Além de seguir as recomendações sobre a elaboração de uma conclusão de relatório, deve também listar o que foi implementado e o que ficou por implementar.