



Laboratório 5

Standard Template Library (STL)

Objetivo

O objetivo deste exercício é colocar em prática conceitos de *containers*, iteradores e algoritmos da STL (*Standard Template Library*) na linguagem de programação C++.

Orientações gerais

Você deverá observar as seguintes observações gerais na implementação deste exercício:

- 1) Apesar da completa compatibilidade entre as linguagens de programação C e C++, seu código fonte **não** deverá conter recursos da linguagem C nem ser resultante de mescla entre as duas linguagens, o que é uma má prática de programação. Dessa forma, deverão ser utilizados **estritamente** recursos da linguagem C++.
- 2) Durante a compilação do seu código fonte, você deverá habilitar a exibição de mensagens de aviso (*warnings*), pois elas podem dar indícios de que o programa potencialmente possui problemas em sua implementação que podem se manifestar durante a sua execução.
- 3) Aplique boas práticas de programação. Codifique o programa de maneira legível (com indentação de código fonte, nomes consistentes, etc.) e documente-o adequadamente na forma de comentários. Anote ainda o código fonte para dar suporte à geração automática de documentação utilizando a ferramenta Doxygen (<http://www.doxygen.org/>). Consulte o documento extra disponibilizado na Turma Virtual do SIGAA com algumas instruções acerca do padrão de documentação e uso do Doxygen.
- 4) Busque desenvolver o seu programa com qualidade, garantindo que ele funcione de forma correta e eficiente. Pense também nas possíveis entradas que poderão ser utilizadas para testar apropriadamente o seu programa e trate adequadamente possíveis entradas consideradas inválidas.
- 5) Lembre-se de aplicar boas práticas de modularização, em termos da implementação de diferentes funções e separação entre arquivos cabeçalho (.h) e corpo (.cpp).
- 6) A fim de auxiliar a compilação do seu projeto, construa **obrigatoriamente** um Makefile que faça uso da estrutura de diretórios apresentada anteriormente em aula.

- 7) Garanta o uso consistente de alocação dinâmica de memória. Para auxiliá-lo nesta tarefa, você pode utilizar o Valgrind (<http://valgrind.org/>) para verificar se existem problemas de gerenciamento de memória.

Autoria e política de colaboração

Este trabalho deverá ser realizado **individualmente**. O trabalho em cooperação entre estudantes da turma é estimulado, sendo admissível a discussão de ideias e estratégias. Contudo, tal interação não deve ser entendida como permissão para utilização de (parte de) código fonte de colegas, o que pode caracterizar situação de plágio. Trabalhos copiados em todo ou em parte de outros colegas ou da Internet serão sumariamente rejeitados e receberão nota zero.

Apesar de o trabalho ser feito individualmente, você deverá utilizar o sistema de controle de versões Git no desenvolvimento. Ao final, todos os arquivos de código fonte do repositório Git local deverão estar unificados em um repositório remoto Git hospedado em algum serviço da Internet, a exemplo do GitHub, Bitbucket, Gitlab ou outro de sua preferência. A fim de garantir a boa manutenção de seu repositório, configure corretamente o arquivo `.gitignore` em seu repositório Git.

Entrega

Você deverá submeter um único arquivo compactado no formato `.zip` contendo todos os códigos fonte resultantes da implementação deste exercício, sem erros de compilação e devidamente testados e documentados, **até as 23h59 do dia 6 de novembro de 2017** através da opção *Tarefas* na Turma Virtual do SIGAA. Você deverá ainda informar, no campo *Comentários* do formulário de submissão da tarefa, o endereço do repositório Git utilizado.

Avaliação

O trabalho será avaliado sob os seguintes critérios: (i) utilização correta dos conteúdos vistos anteriormente e nas aulas presenciais da disciplina; (ii) **utilização correta dos elementos da STL**; (iii) a corretude da execução do programa implementado, que deve apresentar saída em conformidade com a especificação e as entradas de dados fornecidas, e; (iv) a aplicação correta de boas práticas de programação, incluindo legibilidade, organização e documentação de código fonte. A presença de mensagens de aviso (*warnings*) ou de erros de compilação e/ou de execução, a modularização inapropriada e a ausência de documentação são faltas que serão penalizadas. Este trabalho contabilizará nota de **até 2,0 pontos na 2ª Unidade** da disciplina.

Questão 01

Implemente uma função *template* `closest2mean` que receba como parâmetro um intervalo especificado por dois iteradores da categoria `InputIterator` e retorne um iterador para o elemento nesse intervalo cujo valor é o mais próximo da média do intervalo. Tal função deverá ter **obrigatoriamente** a seguinte assinatura:

```
template<typename InputIterator>
InputIterator closest2mean(InputIterator first, InputIterator last);
```

Um exemplo de utilização dessa função seria conforme mostrado no trecho de código a seguir:

```
#include <iostream>
using std::cout;
using std::endl;

#include <vector>
using std::vector;

int main() {
    vector<int> v { 1, 2, 3, 30, 40, 50 };
    auto result = closest2mean(v.begin(), v.end());
    cout << (*result) << endl;
    return 0;
}
```

Como a média dos valores inteiros contidos no intervalo considerado (do início ao fim do vetor `v`) é **21**, logo o programa imprime o valor **30** na saída padrão, referindo-se ao elemento mais próximo dessa média. Pesquise e utilize os algoritmos disponíveis na STL a fim de reduzir o seu código ao máximo.

Questão 02

Implemente uma função *template* `print_elements` que receba como parâmetros um *container* qualquer seguido de um *label* e um separador a serem usados na impressão de todos os elementos do *container*. Tal função deverá conter a seguinte assinatura:

```
template<typename TContainer>
void print_elements(const TContainer& collection, const char* label="",
    const char separator=' ');
```

Um exemplo de utilização dessa função seria conforme mostrado no trecho de código a seguir:

```
#include <set>
using std::set;

int main() {
    set<int> numeros;
    numeros.insert(3);
    numeros.insert(1);
    numeros.insert(4);
    numeros.insert(1);
    numeros.insert(2);
    numeros.insert(5);
}
```

```
    print_elementos(numeros, "Set: ");  
    print_elementos(numeros, "CSV Set: ", ', ');  
    return 0;  
}
```

Um exemplo de execução desse programa seria:

```
$ ./print  
Set: 1 2 3 4 5  
CSV Set: 1;2;3;4;5
```

Questão 03

Predicado é outro conceito importante e muito empregado na STL. Predicados são funções que retornam um valor booleano e são normalmente usadas em critérios de busca ou ordenação. Um predicado sempre retorna o mesmo resultado para o mesmo valor. Predicados podem ser implementados como funções booleanas ou *functors*.

Fazendo uso de uma *functor*, implemente um programa que, dado um *container* de inteiros, encontre o primeiro número primo do conjunto. Utilize esta função para imprimir todos os valores primos no *container*. Em outras palavras, o programa deverá ler um valor *N* fornecido como entrada via linha de comando e imprimir todos os valores primos de 1 a *N*. Você deverá utilizar o algoritmo `find_if` disponível na STL para encontrar o próximo número primo.

Um exemplo de execução do programa seria:

```
$ ./showprimos 50  
$ Numeros primos [1-50]: 1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Questão 04

Após analisar o código a seguir, **indique o seu propósito e descreva o uso dos elementos da STL.**

```
#include <iostream>  
using std::cout;  
using std::endl;  
  
#include <iterator>  
using std::back_inserter;  
  
#include <vector>  
using std::vector;  
  
#include <algorithm>  
using std::transform;  
  
int square(int val) {  
    return val * val;  
}
```

```
int main(int argc, char* argv[]) {
    vector<int> col;
    vector<int> col2;

    for (int i = 1; i <= 9; ++i) {
        col.push_back(i);
    }

    transform(col.begin(), col.end(), back_inserter(col2), square);

    for (vector<int>::iterator it = col2.begin(); it != col2.end(); ++it) {
        cout << (*it) << " ";
    }
    cout << endl;

    return 0;
}
```