UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I

Laboratório 7

Herança, classes abstratas e 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 auxilia-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 junho 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 **3ª 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.

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 conterá 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

A *Notação Polonesa Inversa* (RPN, do Inglês *Reverse Polish Notation*), também conhecida como *notação pós-fixada*, foi inventada pelo filósofo e cientista da Computação australiano Charles Hamblin em meados dos anos 1950 e consiste em uma notação matemática para expressões nas quais operadores vêm após todos os seus operandos. Uma explanação detalhada sobre a Notação Polonesa Inversa pode ser encontrada em https://en.wikipedia.org/wiki/Reverse_Polish_notation.

Utilizando o *container* pilha disponível na STL, implemente um programa que permita avaliar uma determinada expressão escrita na RPN, fornecida como entrada ao programa através da linha de comando. Por razões de simplicidade, assuma que os operandos e operadores serão digitados sempre com um espaço entre eles. Alguns exemplos de expressões em RPN que podem inclusive ser utilizados a título de teste seriam:

Expressão em RPN	Equivalente na forma tradicional (infixa)	Resultado
2 3 + 5 *	(2 + 3) * 5	25
4 2 / 8 + 6 *	((4 / 2) + 8) * 6	60
5 3 * 2 * 10 /	((5 * 3) * 2) / 10	3

Questão 04

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.

Fazendo uso de predicados, implemente um programa que, dado um *container* de inteiros, encontre cada número primo do conjunto e imprima o seu valor. 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 poderá 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 05

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;
}
```

Questão 06

Implemente um programa em C++ (e as respectivas classes, atributos e métodos) para simular o rendimento de contas em um banco. Uma *conta* define a interface principal para qualquer tipo de conta bancária, possuindo, além dos dados comuns a uma conta bancária, as seguintes operações:

- *deposito*, que realiza o depósito de uma quantia na conta, incrementando o seu saldo;
- *saque*, que realiza o saque de uma quantia da conta, decrementando o seu saldo;
- *saldo*, que retorna o saldo atual da conta;
- *jurosPositivos*, que retorna o valor dos juros quando o saldo está positivo até a data atual;
- *jurosNegativos*, que contabiliza o valor dos juros quando o saldo está negativo até a data atual, e;
- *atualiza*, que calcula e aplica os juros sobre a conta.

Utilizando a conta anteriormente definida como base, pode-se definir dois tipos de conta, *conta corrente* e *conta poupança*, cada um com as seguintes características:

Conta corrente:

1. Neste tipo de conta, as atualizações dos juros são feitas pelo banco diariamente.
2. Permite taxa de juros diferente para saldos negativos e positivos.
3. Possui um atributo menor que zero chamado limite. Saques que levem o saldo para abaixo deste valor são recusados. Esta definição não implica que o saldo tenha que estar sempre acima do limite. Ela só vale para saques do cliente, mas não para débitos por conta de juros cobrados sobre saldos negativos.
4. O valor de limite é definido na criação da conta.
5. Este tipo de conta permite saldos negativos.
6. A taxa de juros para saldos positivos é zero, ou seja, não há rendimento.

Conta poupança:

1. Este tipo de conta possui uma data de aniversário, sendo que só neste dia é que se atualiza juros mensalmente.
2. Os juros acrescentados são referentes ao saldo após a última computação, o que significa que depósitos intermediários não rendem juros.
3. Só é permitido saldo maior ou igual a zero.

Além de permitir a criação e operação sobre contas correntes e contas poupança, o programa principal deve apresentar um item *Avançar Data*, através do qual deverá ser informada uma data no tempo a ser considerada como data atual. Isso fará com que todas as contas sejam devidamente atualizadas, levando em consideração a lógica interna de atualização de cada tipo de conta.