UNIVERSIDADE FEDERAL DE SANTA CATARINA
CTS – DEC – ARA
ENC – Engenharia de Computação
Disciplina : Programação II
Professor : Antonio Carlos Sobieranski

Avaliação P1 – Programação Orientada à Objetos **Aluno: _____**

Parte 2 – Prática

Avaliação a ser desenvolvida individualmente, em computador e em Linguagem C++. Submeter via moodle o código fonte da solução do problema. Código deve compilar corretamente para poder ser corrigido.

Considere uma classe base chamada "**EstruturaDados**". Essa classe base armazena uma lista de números. Considere também 2 classes derivadas, herdadas da classe base, chamadas "**Fila**" e "**Pilha**".

Levando em consideração que a lista dos números será um atributo da classe base, implementar as 2 heranças de modo que:

- Implementar métodos "**Resetar**" e "**GetTamanho**" na classe base (limpar lista, retornar tamanho, respectivamente).
- Implementar os métodos "**Inserir**" e "**Remover**" nas classes derivadas de modo a se comportarem de acordo com o propósito da classe derivada em questão (FIFO ou LIFO ?).
- Implementar *menu1*: nesse menu será solicitado qual tipo se deseja utilizar, se **"Fila"** ou **"Pilha"**, e abre *menu2*. Implementar também **"Sair".**
- Implementar *menu2*: uma segunda tela com novas opções surge, contendo "**Inserir**", "**Remover**", "**Limpar**", "**Imprimir**", "**Retornar Menu Anterior".**

- **Observação:** a partir do *menu2*, é possível retornar ao *menu1*, e com isso, trocar o tipo de estrutura de dados "*on the fly*" com a lista previamente preenchida (ora, sabemos quem é o primeiro e o último). É possível assim manter a lista funcionando mas com outra característica (ou seja, cambiamos FIFO ↔ LIFO a qualquer momento). Implemente esse mecanismo usando **HERANÇA com POLIMORFISMO**.

- Dicas:
  - Utilizar *std::vector* para gerenciamento da lista de números e facilitar as implementações.
  - Usar adequadamente os qualificadores de acesso *protected* para que a lista de números seja acessada nas classes derivadas, mas não no "mundo exterior"
  - Separar adequadamente os arquivos .cpp e .hpp
  - Exemplo de polimorfismo abaixo:

```
class Base
{
};

class Derived1 : public Base
{
public:
    void call() { cout << "I am the class derived 1. I'm happy today" << endl; };
};

class Derived2 : public Base
{
public:
    void call() { cout << "I am the class derived 2. I'm happy today too" << endl; };
};

int main()
{
        Base *b = new Base();

        Derived1 *d1 = (Derived1*) b;
        d1->call();

        Derived2 *d2 = (Derived2*) b;
        d2->call();

        delete b;
}
```