Aula 20 - Herança e Polimorfismo em C/C++

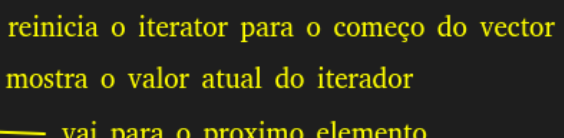
1) Crie uma interface* em C++ chamada Forma com métodos abstratos perímetro e área. Em seguida, crie classes Círculo, Retângulo e Triângulo que herdem de forma e implementem os métodos. Faça um programa que crie instâncias das subclasses e chame as funções para testar se elas estão funcionando como esperado. Por fim, crie um vector cujo tipo seja Forma* (ponteiro para Forma). Adicione neste vector instâncias das subclasses criadas usando new. Faça um loop para mostrar os perímetros e as áreas dos elementos do vector.

** Em C++, interfaces são classes em que todos os métodos são públicos e abstratos. Diferente de Java, interfaces são declaradas como classes e não usando uma palavra específica para este propósito. Métodos abstratos em C++ são métodos virtuais, declarados com "=0;" no final e sem implementação.*

2) Implemente classes Pilha e Fila que contenham apenas métodos Push (adicionar) e Pop (remover). Para implementar a classe Pilha, faça herança privada da classe vector. Para implementar a Fila faça herança privada da classe deque (https://cplusplus.com/reference/deque/deque/pop_front/) que implementa uma lista duplamente encadeada. Faça uma segunda implementação usando composição ao invés de herança (lembrando que apenas os métodos push e pop devem ser visíveis).

3) Iteradores são estruturas de dados usadas para iterar sobre sequências de dados. O exemplo abaixo ilustra como um iterator da classe vector é usado em C++. Na linha 16, é criado um iterador para iterar sobre os elementos do vector v. Na linha 17, o iterator é inicializado com o início do vector v. O iterador sempre se refere à um elemento da sequência. Após a linha 17, it está associado ao 3, o primeiiro elemento de v. A linha 19 mostra o valor atual do iterador. Observe que o operador * foi sobrecarregado para significar "o valor de", como em um ponteiro. O operador ++ foi sobrecarregado na classe iterator para significar "ir para o próximo elemento".

```cpp
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    vector<int> v;

    v.push_back(3);
    v.push_back(5);
    v.push_back(2);
    v.push_back(7);

    vector<int>::iterator it;
    it = v.begin();

    cout << *it << endl; // 3
    it++;
    cout << *it << endl; // 5
    it++;
    cout << *it << endl; // 2
    it++;
    cout << *it << endl; // 7

    return 0;
}
```



Como ilustrado no loop abaixo (extraído de https://cplusplus.com/reference/vector/vector/begin/), em C++, verificamos se ainda existem elementos para iterar comparando o iterador com o fim na sequência:

```cpp
// vector::begin/end
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  for (int i=1; i<=5; i++) myvector.push_back(i);

  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Iteradores uniformizam a forma de iterar sobre elementos de estruturas de dados. Veja no exemplo da página a seguir como o iterador da classe deque pode ser usado da mesma forma que o iterador da classe vector: https://cplusplus.com/reference/deque/deque/begin/ . Note, contudo, que como o vector e o deque são estruturas de dados diferentes, seus iteradores possuem implementações específicas.

O código abaixo ilustra como o mesmo seria feito em Python. A função iter recebe a lista e retorna um iterador para seus elementos. A função next retorna o elemento atual do iterador e move o iterador para o próximo elemento.

```
1
2   l = [3, 5, 2, 7]
3
4   it = iter(l)
5
6   val = next(it)
7   print(val)
8
9   val = next(it)
10  print(val)
11
12  val = next(it)
13  print(val)
14
15  val = next(it)
16  print(val)
17
```

Em Python, o fim da iteração é gerenciado usando o mecanismo de exceções. Quando a sequência termina, o método next lança a exceção StopIteration:

```
1
2   l = [3, 5, 2, 7]
3
4   it = iter(l)
5
6   while True:
7       try:
8           val = next(it)
9           print(val)
10      except StopIteration:
11          break
```

Exercício: Crie uma interface chamada Iterator<T> com os métodos abstratos:

a. void reset(): para reiniciar o processo de iteração.
b. bool hasNext(): para verificar se a sequência terminou.
c. T value(): para recuperar o valor atual da sequência.
d. void next(): para ir para o próximo elemento.

Crie uma subclasse StringIterator que herde de Iterator<char> que receba uma string no construtor e implemente os métodos de forma que o iterador permita iterar sobre os caracteres da string.

Crie uma subclasse FileIterator que herde de Iterator<string> que receba o nome de um arquivo no construtor e perrmita iterar sobre as linhas do arquivo.