Lista 9: classes, polimorfismo & herança

Prof. Felipe Duque
felipe.duque@ufpe.br

1. (5 pontos) Vamos construir uma classe `Pilha` de `string`s em C++.
   (a) Crie um arquivo `Pilha.hpp` e declare os métodos públicos:
      - `Pilha(int max_chars)`: é o método construtor, que recebe a quantidade máxima de caracteres das `string`s a serem armazenadas.
      - `void imprime()`: imprime as `string`s armazenadas.
      - `int tamanho()`: retorna o tamanho da pilha.
      - `std::string retira()`: retira o elemento no topo da pilha, e retorna esse mesmo elemento retirado.
      - `int insere(std::string s)`: insere a `string s` na pilha. Deve verificar se a quantidade de caracteres é permitida. Retorna `1` caso a inserção tenha sido bem sucedida, e `0` caso contrário.

   (b) Declare as seguintes variáveis privadas:
      - `std::vector<std::string> dados`: vetor que realmente contém os dados da pilha.
      - `int max_chars`: quantidade máxima de caracteres das `string`s armazenadas.

   (c) Crie um arquivo `Pilha.cpp` e implemente a classe acima. Consulte a documentação de `std::vector` se precisar de algum método não visto em aula.

   (d) Crie um arquivo `main.cpp` e teste a classe `Pilha`.

2. (5 pontos) Agora, vamos criar uma classe-base `EstruturaDados`, da qual as classes `Pilha` e `Fila` serão derivadas.

(a) Crie o arquivo `EstruturaDados.hpp` e declare os métodos e variáveis adequados. (Quais métodos/variáveis são absolutamente iguais para as duas estruturas de dados? Algum método precisa ser implementado somente nas classes derivadas?, se sim, lembre-se de utilizar `virtual`.)

(b) Crie o arquivo `EstruturaDados.cpp` e implemente os métodos adequados do cabeçalho.

(c) Adeque a classe `Pilha` para se tornar uma classe derivada de `EstruturaDados`. Note que `Pilha` não precisa repetir as implementações de alguns métodos/variáveis de `EstruturaDados`.

(d) Crie e implemente a classe `Fila`, também derivada de `EstruturaDados`.

(e) Crie um arquivo `main.cpp` para testar essa hierarquia de classes.

Notas possivelmente úteis:

- O comando `vec.erase(vec.begin())` remove o elemento na frente do `std::vector vec`.
- O comando `vec.front()` retorna o elemento na frente do do `std::vector vec`.
- Quando começamos a ter vários `#include`s, torna-se uma boa ideia utilizar os *include guards*.