Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Escola Politécnica - Curso de Bacharelado em Engenharia de Computação
Disciplina: Programação Orientada a Objetos - ECo – Professor: Roland Teodorowitsch
16 de março de 2023

Lista de Exercícios 3 (2023/1)

Este trabalho consiste em resolver a lista de exercícios das páginas a seguir, em C++.
Para a resolução e entrega devem ser seguidas as seguintes regras:

- criar uma pasta com o nome do aluno no formato *camelHump* (por exemplo, para João Pedro da Silva, usar `JoaoPedroDaSilva`);
- dentro dessa pasta criar programas em C++ para resolver cada um dos exercícios, salvando o código-fonte em um arquivo com o nome `Exercicio` seguido do número do exercício com três dígitos (por exemplo, `Exercicio001.cpp`, `Exercicio002.cpp`, ..., `Exercicio100.cpp`);
- no início de cada arquivo em C++, incluir um comentário informando o nome do arquivo, o nome do autor, a finalidade do programa e a versão (ou data) de criação (ou atualização);
- se houver dados para serem lidos, eles devem ser lidos na mesma ordem em que eles são citados no enunciado, escolhendo os tipos numéricos adequadamente;
- escrever os resultados sempre na mesma ordem em que eles são citados no enunciado, escolhendo os tipos numéricos adequadamente (números reais devem ser apresentados sempre com 4 casas decimais, salvo se indicado de outra forma);
- na versão final, tomar o cuidado de não imprimir nada diferente da saída esperada (não devem aparecer, por exemplo, mensagens pedindo que o usuário forneça ou digite determinado valor no terminal);
- a entrega deverá ser feita no dia e horário informado pelo professor em sala de aula e/ou definida na opção de entrega da plataforma moodle da PUCRS;
- cada aluno deverá submeter os códigos-fontes compactados no formato ZIP, usando o mesmo nome da pasta(por exemplo, para João Pedro da Silva, o arquivo compactado deverá chamar-se `JoaoPedroDaSilva.zip`).

15. Crie uma classe em C++ denominada `Pessoa` para gerenciar e armazenar as seguintes informações sobre uma pessoa: nome (cadeia de caracteres), idade (inteiro), altura (valor real correspondente à altura em centímetros) e taxa de crescimento anual (valor real correspondente à altura média em centímetros que a pessoa cresce por ano).

    A classe deve também disponibilizar os seguintes métodos:

    - construtor sem parâmetros (vazio): que inicializa todos as variáveis de instância com "valores nulos";
    - construtor que recebe como parâmetros o nome, a idade, a altura e a taxa de crescimento;
    - métodos para obter cada um dos os dados armazenados: `obtemNome`, `obtemIdade`, `obtemAltura` e `obtemTaxaCrescimento` – cada um desses métodos não recebe nenhum argumento e retorna a respectiva informação;
    - métodos para definir cada um dos os dados armazenados: `defineNome`, `defineIdade`, `defineAltura` e `defineTaxaCrescimento` – cada um desses recebe a respectiva informação que deve ser armazenada e não retorna nada;
    - método `envelhece`: que aumenta a idade da pessoa em 1 ano e acrescenta o valor da taxa de crescimento na altura da pessoa.

    Insira a sua implementação da classe `Pessoa` no programa de teste mostrado a seguir. Inicialmente este programa lê os dados de uma pessoa (cada informação - nome, idade, altura e taxa de crescimento - em uma linha separada da entrada) e depois de outra (também com cada informação em uma linha). A seguir este programa envelhece ambas as pessoas cinco vezes (ou seja, 5 anos) e mostra as informações da pessoa que apresenta a maior altura após os 5 anos. Em caso de empate, as informações de ambas as pessoas são mostradas.

```cpp
// Exercicio015.cpp

#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

class Pessoa {
  // coloque aqui a sua implementacao...
};

void le_pessoa(Pessoa *p) {
    string s;
    getline(cin,s);
    p->defineNome(s);
    getline(cin,s);
    p->defineIdade(stoi(s));
    getline(cin,s);
    p->defineAltura(stod(s));
    getline(cin,s);
    p->defineTaxaCrescimento(stod(s));
}

void imprime_pessoa(Pessoa *p) {
    cout << p->obtemNome() << "␣(";
    cout << p->obtemIdade() << "␣anos;␣";
    cout << fixed << setprecision(4);
    cout << p->obtemAltura() << "␣cm;␣";
    cout << p->obtemTaxaCrescimento() << "␣cm)" << endl;
}

int main() {
    Pessoa *p1, *p2;

    // LEITURA
    p1 = new Pessoa();
    le_pessoa(p1);
    p2 = new Pessoa();
    le_pessoa(p2);
    // ENVELHECIMENTO
    for (int i=0; i<5; ++i) {
        p1->envelhece();
        p2->envelhece();
    }
    // RESULTADOS
    if (p1->obtemAltura() > p2->obtemAltura())
        imprime_pessoa(p1);
    else if (p1->obtemAltura() < p2->obtemAltura())
        imprime_pessoa(p2);
    else {
        imprime_pessoa(p1);
        imprime_pessoa(p2);
    }
    delete p2;
    delete p1;
```

```
    return 0;
}
```

Exemplos:

Entrada	Saída
Antonio Acura 6 108.0 5.5 Bernardo Bugatti 5 100.5 6.0	Antonio Acura (11 anos; 135.5000 cm; 5.5000 cm)
Carlos Citroen 7 103,5 5,5 Denise Daimler 5 100 7	Denise Daimler (10 anos; 135.0000 cm; 7.0000 cm)
Everton Esther 4 105 5 Francisca Ferrari 6 100 6	Everton Esther (9 anos; 130.0000 cm; 5.0000 cm) Francisca Ferrari (11 anos; 130.0000 cm; 6.0000 cm)
Gilson Geely 13 143.7 4.3 Heitor Hummer 14 137.8 4.9	Gilson Geely (18 anos; 165.2000 cm; 4.3000 cm)
Ilza Iveco 17 151.2 4.1 Jenilson Jiefang 15 161.2 2.2	Jenilson Jiefang (20 anos; 172.2000 cm; 2.2000 cm)

16. Crie uma classe em C++ denominada `Elevador` para gerenciar elevadores dentro de um prédio, armazenando as informações relevantes sobre o funcionamento de um elevador. A classe deve armazenar: o andar atual (0 corresponde a térreo), total de andares no prédio (excluindo o térreo – ou seja, em um edifício com 5 andares, por exemplo, o número de andares varia de 0 até 5, inclusive), capacidade do elevador (número máximo de pessoas que podem ocupar o elevador) e quantas pessoas estão ocupando o elevador no momento.

    A classe deve também disponibilizar os seguintes métodos:

    - construtor sem parâmetros (vazio): que inicializa todos as variáveis de instância com 0;
    - construtor que recebe como parâmetros a capacidade do elevador e o total de andares no prédio (os elevadores sempre começam no térreo e sem nenhum ocupante);
    - método `entra`: para acrescentar uma pessoa no elevador (só deve acrescentar se ainda houver espaço) – esse método não recebe nenhum argumento e não retorna nenhuma informação;
    - método `sai`: para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele) – esse método não recebe nenhum argumento e não retorna nenhuma informação;
    - método `sobe`: para subir um andar (não deve subir se já estiver no último andar) – esse método não recebe nenhum argumento e não retorna nenhuma informação;
    - método `desce`: para descer um andar (não deve descer se já estiver no térreo) – esse método não recebe nenhum argumento e não retorna nenhuma informação;
    - métodos para obter cada um dos os dados armazenados: `obtemAndarAtual`, `obtemTotalAndares`, `obtemCapacidade` e `obtemNumPessoas` – cada um desses métodos não recebe nenhum argumento e retorna a respectiva informação;
    - método `defineTotalAndares`: que define o número total de andares (excluindo o térreo) – esse método recebe o número total de andares e não retorna nada;
    - método `defineCapacidade`: que define o número máximo de pessoas que podem ocupar o elevador – esse método recebe a capacidade e não retorna nada; e
    - método `movimenta`: que recebe uma cadeia de caracteres (`string`) que contém os seguintes caracteres, que determinam sequências de métodos de operação do elevador que deverão ser chamados: `'^'` (sobre o elevador um andar), `'v'` (desce o elevador um andar, `'+'` (acrescenta uma pessoa no elevador), `'-'` (sai uma pessoa do elevador) – esse método não retorna nenhuma informação.

    Os elevadores devem manter a sua consistência, ou seja: não podem nem subir acima do último andar, nem descer abaixo do térreo (andar 0), nem receber mais pessoas do que a sua capacidade e nem ser ocupado por um número negativo de pessoas. Qualquer operação que levar a uma dessas situações deve ser desconsiderada.

    Insira a sua implementação da classe `Elevador` no programa de teste mostrado a seguir. Inicialmente este programa lê a capacidade e o total de andares de um elevador, e também uma cadeia de caracteres com a sequência de operações a ser realizada no elevador. A seguir são feitos testes básicos, para então colocar o elevador em operação (usando a sequência de operações lida) e, por fim, mostrar o número atual de pessoas ocupando o elevador e o andar do elevador.

```
// Exercicio016.cpp

#include <iostream>

using namespace std;

class Elevador {
  // coloque aqui a sua implementacao...
};

int main() {
  Elevador *e;
  int capacidade, totalAndares;
  string operacao;

  // LEITURA
  cin >> capacidade;
  cin >> totalAndares;
  cin >> operacao;
  // TESTE DE FUNCIONAMENTO
  e = new Elevador();
  if ( e->obtemAndarAtual()!=0 || e->obtemCapacidade()!=0 ||
       e->obtemNumPessoas()!=0 || e->obtemTotalAndares()!=0 )
      return 1;
  e->defineCapacidade(capacidade);
  if ( e->obtemAndarAtual()!=0 || e->obtemTotalAndares()!=0 ||
       e->obtemNumPessoas()!=0 || e->obtemCapacidade()!=capacidade )
      return 1;
```

```
  e->defineTotalAndares(totalAndares);
  if ( e->obtemAndarAtual()!=0 || e->obtemTotalAndares()!=totalAndares ||
       e->obtemNumPessoas()!=0 || e->obtemCapacidade()!=capacidade )
      return 1;
  for (int i=0; i<=totalAndares+2; ++i) {
      e->sobe();
      if ( e->obtemAndarAtual() > totalAndares ) return 1;
  }
  for (int i=0; i<=totalAndares+2; ++i) {
      e->desce();
      if ( e->obtemAndarAtual() < 0 ) return 1;
  }
  for (int i=0; i<=capacidade+2; ++i) {
      e->entra();
      if ( e->obtemNumPessoas() > capacidade ) return 1;
  }
  for (int i=0; i<=capacidade+2; ++i) {
      e->sai();
      if ( e->obtemNumPessoas() < 0 ) return 1;
  }
  // OPERACAO
  e->movimenta(operacao);
  // SAIDA
  cout << e->obtemNumPessoas() << "␣" << e->obtemAndarAtual() << endl;
  delete e;
  return 0;
}
```

Exemplos:

Entrada	Saída
8 10 +^+^+^+^--^^^^^+^--vvvvv++++vvv--	3 2
4 9 +^+^+^+^--^^^^^+^--vvvvv++++vvv--	2 1
12 12 +^^++++^^^^+++^^^^^--^+vvvvvvv+^^^^^^+++vvvv----	7 7
8 6 +^^++++^^^^+++^^^^^--^+vvvvvvv+^^^^^^+++vvvv----	4 2
6 20 ++^^^^^+v+-^^^^^^^^^-+^^^^++^^^+vvvv--vvvvvvvvvv+	5 6
4 15 ++^^^^^+v+-^^^^^^^^^-+^^^^++^^^+vvvv--vvvvvvvvvv+	3 1