Pontifícia Universidade Católica de Goiás
Escola de Ciências Exatas e da Computação
CMP1048 - Técnicas de Programação
Max Gontijo de Oliveira
Lista de Exercícios 3:B - Classes

Uso de ponteiros de objetos. Em C++, podemos reservar espaço de memória para variáveis de duas maneiras: declarando ou dinamicamente. Com instâncias de casses (objetos) não é diferente. Veja os exemplos.

```
int a;   // Cria uma variável do tipo "int" com nome "a".
int *b;  // Cria uma variável do tipo "int*" com nome "b".
float c; // Cria uma variável do tipo "float" com nome "c".
Carro d; // Cria uma variável do tipo "Carro" com nome "d".
Arma e;  // Cria uma variável do tipo "Arma" com nome "e".
Arma *f; // Cria uma variável do tipo "Arma*" com nome "f".
```

Para criar dinamicamente, podemos seguir a mesma linha:

```
int *a = new int;        // Cria dinamicamente uma variável do tipo "int" e guarda o endereço dessa variável dinâmica na variável do tipo "int*" declarada com nome "a".
int *b = new int[1];     // Cria dinamicamente um vetor de "int" com apenas 1 posição e guarda o endereço desse vetor na variável do tipo "int*" declarada com nome "b". Esse comando com o tamanho específico de 1 tem exatamente o mesmo efeito que o comando anterior.
int *c = new int[5];     // Cria dinamicamente um vetor de "int" com 5 posições e guarda o endereço desse vetor na variável do tipo "int*" declarada com nome "c".
Arma *e = new Arma;      // Cria dinamicamente uma variável do tipo "Arma" e guarda o endereço dessa variável dinâmica na variável do tipo "Arma*" declarada com nome "e". Nesse caso, a classe "Arma" foi instanciada utilizando o construtor sem parâmetros.
Arma *f = new Arma();    // Cria dinamicamente uma variável do tipo "Arma" e guarda o endereço dessa variável dinâmica na variável do tipo "Arma*" declarada com nome "f". Nesse caso, a classe "Arma" foi instanciada utilizando o construtor sem parâmetros. Esse comando com os parênteses sem parâmetros tem o mesmo efeito que o comando anterior.
Arma *g = new Arma[5];   // Cria dinamicamente um vetor de "Arma" com 5 posições e guarda o endereço desse vetor na variável do tipo "Arma*" declarada com nome "g". Note que cada posição do vetor criado é um objeto do tipo "Arma".
Arma **h = new Arma*[5]; // Cria dinamicamente um vetor de "Arma*" com 5 posições e guarda o endereço desse vetor na variável do tipo "Arma**" declarada com nome "h". Note que cada posição do vetor criado é um ponteiro e não um objeto do tipo "Arma". Ou seja, cada posição do vetor pode receber o endereço de um objeto do tipo "Arma".
```

Quando se instancia uma classe, você deve informar qual é o construtor utilizado. Sintaticamente, tem diferença entre declarar e criar dinamicamente. Veja:

```
Arma a;    // Cria de forma declarativa um objeto do tipo "Arma" com o nome "a", utilizando o construtor sem parâmetros.
Arma b(5); // Cria de forma declarativa um objeto do tipo "Arma" com o nome "b", utilizando o construtor que recebe um parâmetro do tipo "int" (5, no caso).
Arma *c = new Arma(); // Cria de forma dinâmica um objeto do tipo "Arma" e armazena seu endereço na variável ponteiro (Arma*) que tem nome "c", utilizando o construtor sem parâmetros.
Arma *d = new Arma(5); // Cria de forma dinâmica um objeto do tipo "Arma" e armazena seu endereço na variável ponteiro (Arma*) que tem nome "d", utilizando o construtor que recebe um parâmetro do tipo "int" (5, no caso).
```

Quando se usa referências diretas ao objeto (como quando são declarados, por exemplo), a forma de acessar seus membros públicos é por meio de um ".". Quando se usa o endereço de um objeto, a forma de acesso é por meio de um ->". Veja:

```
Arma a;
Arma b(5);
Arma *c = new Arma();
Arma *d = new Arma(5);

a.recarregar();
a.disparar();

b.recarregar();
b.disparar();

c->recarregar();
c->disparar();

d->recarregar();
d->disparar();
```

Vamos criar um programinha que cria vários objetos dinâmicos e coloca em um vetor de ponteiros.

```
int n;

cout << "Quantidade de armas: ";
cin >> n;

cout << "Criando dinamicamente um vetor de ponteiros para Arma\n";
Arma **vet = new Arma*[n];

for (int i = 0;i < n;i++) {
   int c;
   cout << "Capacidade da arma " << i << ":";
   cin >> c;

   cout << "Criando dinamicamente um objeto Arma e armazenando o endereço em vet[" << i << "]\n";
   vet[i] = new Arma(c);
}

for (int i = 0;i < n;i++) {
   cout << "Recarregando a arma " << i << "\n";
   vet[i]->recarregar();
}

for (int i = 0;i < n;i++) {
   cout << "Disparando a arma " << i << "\n";
   bool continuar = true;
   while (continuar) {
      if (vet[i]->disparar()) {
         cout << "BANG!!\n";
      } else {
         cout << "CLICK!!\n";
         continuar = false;
      }
   }
}

for (int i = 0;i < n;i++) {
   cout << "Desalocando a arma " << i << "\n";
   delete vet[i];
}

cout << "Desalocando o vetor de ponteiros\n";
delete[] vet;
```

Consideração sobre strings e datas:

Nessa lista de exercícios, quando for mencionado que o tipo de uma variável é string, o aluno deverá considerar o tipo `string` do C++, o ponteiro para vetor de `char` (`char*`) ou um vetor de `char` propriamente dito. A escolha de qual tipo utilizar será do próprio aluno.

Quando for mencionado que o tipo de uma variável é uma data, o aluno deverá considerar que se trata de uma string no formato ***DD/MM/AAAA***.

1. Crie uma classe chamada `Pessoa` para representar uma pessoa, com os atributos CPF (`long long`), nome, data de nascimento, altura (em cm), peso (em kg). Para essa classe, crie ainda:

   - Um construtor que receba o CPF e o nome e inicialize os respectivos atributos do objeto com os valores recebidos.
   - Um método chamado `peso_kg` que retorna o peso da pessoa em quilogramas.
   - Um método chamado `peso_g` que retorna o peso da pessoa em gramas.
   - Um método chamado `altura_cm` que retorna a altura da pessoa em centímetros.
   - Um método chamado `altura_m` que retorna a altura da pessoa em metros.
   - Um método que se chame `calcula_idade` que receba por parâmetro uma data e que calcula e retorna a idade da pessoa na data passada por parâmetro.
   - Um método que se chame `calcula_imc` que não recebe nenhum parâmetro e que calcula e retorna o IMC da pessoa.
   - Um método chamado `imprimir` que não recebe parâmetros e imprime todos os atributos da pessoa. (apenas para testes mesmo)

2. Crie uma classe chamada `DocumentoIdentidade` que tenha os atributos CPF, nome e data de nascimento. Agora, crie uma nova classe chamada `Pessoa2` semelhante à criada na questão 1. A

diferença é que os atributos CPF, nome e data de nascimento deverão ser substituídos por uma única variável chamada `documentacao` do tipo `DocumentoIdentidade`.

3. Crie uma classe `Agenda` que armazena 10 pessoas (usar vetor de ponteiros da classe `Pessoa` do exercício 1). Declare e implemente os seguintes métodos para essa classe:

   - `bool incluir(Pessoa* p)`
     - Busca a primeira posição nula no vetor e armazena a pessoa `p` na posição encontrada. Retorna `true` se conseguiu encontrar um espaço livre e fez a inclusão ou `false` caso contrário.
   - `Pessoa* consultar(long long cpf)`
     - Busca na agenda (vetor) a existência de uma pessoa com o CPF igual ao informado. Retorna o ponteiro dessa pessoa caso encontre ou `NULL` caso contrário.
   - `boolean remover(long long cpf)`
     - Busca na agenda (vetor) a existência de uma pessoa com o CPF igual ao informado. Caso encontre um elemento com o CPF em questão, desaloque a memória desse objeto utilizando o comando `delete` e atribua `NULL` à posição desse objeto no vetor. Retorne `true` caso encontre ou `false` caso contrário. Note que após essa operação, o vetor de elementos poderá ter lacunas no meio. Essas lacunas poderão ser utilizadas para futuras inserções. (opcionalmente, o aluno pode fazer um "chega pra trás", eliminando as lacunas)
   - `void imprimir_lista()`
     - Itera sobre os elementos do vetor chamando, para cada elemento não nulo, o método de impressão de cada pessoa da agenda. Deverá haver uma quebra de linha a cada pessoa impressa.
   - `int espaco_livre()`
     - Retorna o número de espaços disponíveis na agenda.

   Dica: Lembre-se de inicializar as posições do vetor com `NULL` utilizando um construtor.

4. Crie um programa que instancie e teste a agenda criada na questão 3. O programa deverá entrar em um laço onde será lido uma operação (`char`) e, dependendo da operação lida, ler outras informações e chamar um dos métodos específicos da agenda.

   - `I` → criar uma pessoa, ler cpf, nome, data de nascimento, altura e peso e incluir na agenda.
   - `C` → ler cpf, consultar na agenda e imprimir a pessoa encontrada.
   - `R` → ler cpf e remover a pessoa da agenda.
   - `L` → listar todas as pessoas da agenda.
   - `Q` → imprimir na tela o número de espaços livres na agenda.
   - `M` → imprimir a média de peso, a média de altura e a média idade de todas as pessoas armazenadas na agenda.
   - `X` → sai do laço e encerra o programa.

5. Desafio desafiador que vai desafiar para caramba! Faça as alterações necessárias na classe `Agenda` da questão 3 que assegure que a agenda tenha capacidade ilimitada[1]. Uma solução que não é aceitável é simplesmente iniciar a agenda com um vetor gigantesco. Sua agenda deve se preocupar em ter uma capacidade que vai aumentando na medida em que se faz necessário. Evidentemente, o método `int espaco_livre()` não fará mais sentido de existir, uma vez que a classe terá capacidade ilimitada.

6. Crie as seguintes classes:

   - `Espada`

[1] Na verdade, virtualmente ilimitada, uma vez que a memória não é infinita.

  - Atributos
    * `nome` (string)
    * `dano` (float)
  - Métodos
    * `float golpear()`
      · Retorna a intensidade do golpe realizado pela espada.

- `Escudo`
  - Atributos
    * `nome` (string)
    * `defesa` (float)
  - Métodos
    * `float defender()`
      · Retorna a intensidade da defesa realizada pelo escudo.

- `Guerreiro`
  - Atributos
    * `nome` (string)
    * `vida` (float)
    * `resistencia` (float)
    * `forca` (float)
    * `espada` (do tipo `Espada`)
    * `escudo` (do tipo `Escudo`)
  - Métodos
    * `bool atacar(Guerreiro* adversario)`
      · Remove uma quantidade de `vida` do guerreiro adversário. Caso a `vida` do guerreiro adversário tenha se esgotado (`vida` ≤ 0), o método deve retornar `true`. Caso contrário, deverá retornar `false`.

O método `golpear` da classe `Espada` obtém um novo número randômico[2] (float) entre 0 e 1 e retorna o valor desse número randômico multiplicado pelo dano da espada.

O método `defender` da classe `Escudo` obtém um novo número randômico (float) entre 0 e 1 e retorna o valor desse número randômico multiplicado pela defesa do escudo.

Seja g_1 o guerreiro do qual método `atacar` foi chamado e g_2 o guerreiro adversário (passado como parâmetro para o método), o método `atacar` da classe `Guerreiro` deverá diminuir a vida do adversário de acordo com a seguinte expressão:

$$g_2.vida = g_2.vida - \frac{g_1.forca \times g_1.espada.golpear()}{g_2.resistencia \times g_2.escudo.defender()}$$

Dica: para gerar números randômicos, utilize as funções `srand` e `rand` da biblioteca `<cstdlib>`. A função `srand` define uma semente[3] e deveria ser chamado uma única vez por execução do programa. Assim, recomenda-se que chame no início do `main`, uma única vez. A função `rand` gera números randômicos calculados com base na semente definida em `srand`. O número retornado por `rand` é um grande número inteiro. Procure uma estratégia para transformar esses números inteiros em números float de 0 a 1. Como os números gerados pela função `rand` dependem da semente, se a semente for fixa, a ordem dos números randômicos gerados será sempre a mesma. Assim, `srand` deveria receber um número diferente a cada execução. Uma boa estratégia para isso é usar a função `time` da biblioteca `ctime` para gerar essa semente. Assim, a chamada única à função `srand` no início do `main` seria assim: `srand(time(NULL));`

[2]Número randômico é um número pseudo-aleatório produzido pelo computador.

[3]Em se tratando de números randômicos, uma semente é um número do qual parte o cálculo de todos os números randômicos seguintes.

7. Utilizando as classes desenvolvidas na questão 6, crie um programa que instancie dois guerreiros, leia os atributos de cada guerreiro bem como de cada arma e de cada escudo e coloque os dois para lutar um contra o outro. A luta deverá consistir em alternar ataques entre os dois guerreiros até que um deles tenha a vida zerada. O programa deverá mostrar, a cada ataque, a situação dos pontos de vida de cada guerreiro antes e depois do ataque.

8. Um número complexo é representado por duas partes: real e imaginária. É possível fazer quaisquer operações aritméticas com números complexos, tais como as operações com número reais. Crie uma classe chamada `NumeroComplexo` que tenha apenas dois atributos float: parte real e parte imaginária. Essa classe deverá ter dois construtores: um sem parâmetros, que inicializa as partes real e imaginária com o valor 0; e um outro que recebe as partes real e imaginária como parâmetro e atribui esses valores para os respectivos atributos.

   Agora, crie os seguintes métodos para essa classe:

   - `float parte_real()` → retornar o valor da parte real do número complexo.
   - `float parte_imginaria()` → retornar o valor da parte imaginária do número complexo.
   - `NumeroComplexo soma(NumeroComplexo n)` → recebe um outro número complexo e realiza a soma entre os números complexos, retornando o resultado em um outro objeto `NumeroComplexo`.
   - `NumeroComplexo subtrai(NumeroComplexo n)` → recebe um outro número complexo e realiza a subtração entre os números complexos, retornando o resultado em um outro objeto `NumeroComplexo`.
   - `NumeroComplexo multiplica(NumeroComplexo n)` → recebe um outro número complexo e realiza a multiplicação entre os números complexos, retornando o resultado em um outro objeto `NumeroComplexo`.
   - `NumeroComplexo divide(NumeroComplexo n)` → recebe um outro número complexo e realiza a divisão entre os números complexos, retornando o resultado em um outro objeto `NumeroComplexo`.
   - `void imprimir()` → imprime o número complexo no formato `a + bi`, onde `a` é a parte real e `b` é a parte imaginária.

   Crie um programa para testar sua classe.

   Modo Hard
   Crie um método na classe `NumeroComplexo` chamado `divide` que recebe um outro número complexo e realiza a operação de divisão entre os números complexos, retornando o resultado em outro objeto `NumeroComplexo`.

9. Crie uma classe chamada `Matriz` que tenha apenas três atributos: quantidade de linhas (`int`), quantidade de colunas (`int`) e ponteiro para uma matriz dinâmica (`float**`). Essa classe deverá ter um construtor que recebe apenas as dimensões da matriz (número de linhas e número de colunas) e esse construtor deverá inicializar os respectivos atributos com esses valores, bem como criar a matriz dinâmica com as dimensões fornecidas. Além disso, o construtor deverá inicializar cada posição da matriz dinâmica com o valor zero. Essa classe dever ter, ainda, os seguintes métodos:

   - `float get(int i, int j)` → retornar o valor da linha `i` e coluna `j` da matriz.
   - `void set(int i, int j, float x)` → atribui o valor de `x` para a posição da linha `i` e coluna `j` da matriz.
   - `Matriz soma(Matriz B)` → recebe uma outra matriz, realiza a soma de matrizes retornando um outro objeto `Matriz` com o resultado dessa soma de matrizes. Caso a operação não seja possível, o método deverá retornar um objeto `Matriz` com dimensões 0×0.
   - `Matriz soma(int n)` → uma sobrecarga que recebe um `int`, e retorna um outo objeto `Matriz` com os valores de cada posição da matriz original somados com o valor do parâmetro.

- `Matriz subtrai(Matriz B)` → recebe uma outra matriz, realiza a subtração de matrizes retornando um outro objeto `Matriz` com o resultado dessa subtração de matrizes. Caso a operação não seja possível, o método deverá retornar um objeto `Matriz` com dimensões 0×0.
- `Matriz subtrai(int n)` → uma sobrecarga que recebe um `int`, e retorna um outo objeto `Matriz` com os valores de cada posição da matriz original subtraídos do valor do parâmetro.
- `Matriz multiplica(Matriz B)` → recebe uma outra matriz, realiza a multiplicação de matrizes retornando um outro objeto `Matriz` com o resultado dessa multiplicação de matrizes. Caso a operação não seja possível, o método deverá retornar um objeto `Matriz` com dimensões 0×0.
- `Matriz transposta()` → retorna um outo objeto `Matriz` que representa a matriz transposta da matriz original.
- `Matriz imprimir()` → imprime os elementos da matriz no terminal. Cada linha da matriz, em uma nova linha do terminal.

Crie um programa para testar sua classe.