UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

TECNOLOGIAS DE PROGRAMAÇÃO DE SISTEMAS DE INFORMAÇÃO

Programação orientada a objectos

PROGRAMAÇÃO ORIENTADA A OBJECTOS| Prof. Doutora Frederica Gonçalves

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA

Fundo Social Europeu

- **Exercício:**

Elabore um programa em C++ que crie uma **classe** do tipo **aluno**, com os seguintes **objectos**: **char[ ]** nome_do_aluno, **int** número_do_aluno, **float** nota_esperada.
A introdução de informação deverá ser feita manualmente, através do **cout**, **cin/gets()**. Deve limpar o ecrã depois da introdução dos dados.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Reutilizando o exercício anterior, retire o objecto **float nota_esperada**, e coloque **float nota_1_teste**, **float nota_2_teste** e **float nota_final**().
- O método/função **nota_final**(), calcula a média dos testes.
- Realize a introdução dos dados manualmente, limpe o ecrã

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

– Realize o exercício anterior, com encapsulamento dos objectos **float nota_1_teste** e **float nota_2_teste** com o especificador de acesso ***private***.
Utilize as funções de acesso **get**() e **set**() para solucionar o problema colocado.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Realize um programa em C++, reutilizando a classe aluno, sem encapsulamento dos objectos, mas com os **protótipos float nota_1_teste**() e **float nota_2_teste**(), como funções – membro definidas fora da classe aluno.
- Defina, utilizando o operador de resolução de escopo **(::)**, as funções como atribuições aleatórios de notas, entre [5 - 20].
- Como objecto da classe aluno, declare ainda, uma função para mostrar os resultados finais (**void mostradados**())

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Chamadas a funções – membro de uma classe a partir dos seus objectos:**

  – Os **objectos** das classes são criados quando se declara uma **variável de classe**.

  – Os **objectos** são só gravados na memória, após a criação dessa mesma **classe**.

  – Cada **atributo/objecto** criado tem o seu próprio espaço de memória por cada **variável de classe** criada.

  – Quanto às **funções**, são criadas na memória num espaço partilhado por todas as **variáveis da classe**.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Chamadas a funções – membro de uma classe a partir dos seus objectos (ex.):**

```
class aluno {
        char nome [40];
        int idade;
        int ano(int ano_actual){
        return ano_actual – idade;}
} ;
```

```
aluno a1, a2;
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Chamadas a funções – membro de uma classe a partir dos seus objectos (ex.):**

```
cout << a1.nome;
cout << a2.nome;
```

– Esta chamada é particular a cada objecto.

```
cout << a1.ano(2008);
cout << a2.ano(2008);
```

– Esta chamada é comum a todos os objectos.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Construtores e destrutores de uma classe:**

Construtores:

– São **funções – membro** de uma **classe** com o objectivo de **criar objectos**;

– São denominadas com o **mesmo** nome da **classe**.

Destrutores:

– São **funções – membro** de uma **classe** para **destruir objectos** criados;

– São denominados com o mesmo nome da **classe** com um **(~)til** à frente.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Estrutura dos construtores e destrutores:**

Construtores:

nome_classe ([parâmetros]) {<bloco de instruções>}

Destrutores:

~nome_classe ([parâmetros]) {<bloco de instruções>}

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Exemplo construtores e destrutores:**

```
class aluno {
public:
        char nome[40] ;
        int numero ;
        aluno(){
                cout << "Entrou um aluno!\n";
                }
        ~aluno(){
                 cout << "Saiu um aluno!\n";
                 }
};
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Função que permite criar e destruir os “alunos”:**

```
…
void inscricao(int b){
    aluno a[b];
    for(int k = 1; k <= b; k++){
        cout << "Nome do aluno numero "<<k<<": ";
        cin >>a[k].nome; }
        system("pause"); system("cls");
        cout << "\tLISTA DOS ALUNOS INSCRITOS:\n";
        for(int k = 1; k <= b; k++){
            cout << "Nome: " <<a[k].nome<<'\t';
            cout << "Numero: "<<k<<'\n';}
}
```

- **Exercício:**
  - **Resolução:**

– Com a classe criada, e a função de entrada e saída de alunos definida.

– Crie um “main()”, que peça o número de alunos para inscrever, e chame a função “inscrição(i)”.

```
main(){
 cout << "Deseja inscrever quantos alunos? ";
     int j; cin >> j; inscricao(j); system("pause");
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Output:**

```
Deseja inscrever quantos alunos? 2
Entrou um aluno!
Entrou um aluno!
Nome do aluno numero 1: João
Nome do aluno numero 2: Joana
```

```
          LISTA DOS ALUNOS INSCRITOS:
Nome: João        Numero: 1
Nome: Joana       Numero: 2
Saiu um aluno!
Saiu um aluno!
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Sobrecarga ao nível dos construtores de uma classe:**

  – **Sem** a definição de **construtores**, o compilador gera automaticamente o **construtor vazio**; ex: “**aluno( ) { }**”

  – A **sobrecarga de construtores**, é a possibilidade de **criar** a **classe** de **diversas formas**, dependendo dos objectos nela contidos.

```
...
    aluno() { cout << "Entrou um aluno!\n";}
    aluno(int i) {
            numero = i;
            cout << “Entrou um aluno!\n”;
            cout << “Qual o nome do aluno com o n. “<<i<<“?\n”;
            gets(nome); }
    aluno(char *n, int i) {
            strcpy(nome, n); numero = i;
          cout << “Entrou um alno!\n”;
   }
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Sobrecarga ao nível dos construtores de uma classe (ex. de criação da classe):**

```
aluno a1;
aluno a2 (“Ana”);
aluno a3 (“Joao”, 3);
aluno a4 (4);
```

– Dependendo dos construtores anteriores, qual a criação da classe aluno que não está definida?

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Possíveis soluções:**

```
...
    aluno(char *n, int i = 18) {
            strcpy(nome, n); numero = i;
             cout << “Entrou um alno!\n”;
    }
```

ou

```
...
    aluno(char *n) {
            strcpy(nome, n); cout << “Entrou um alno!\n”;
            cout << “Qual o numero do(a) aluno(a) ”<<nome<<“!\n”;
            cin >> numero;}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Variáveis – membro do tipo static:**
  - São **variáveis/objectos** de uma **classe**, **partilhadas** por todos os **objectos** dessa mesma **classe**;
  - Existe **uma variável** para todos os **objectos**.

```
…
class aluno {
public:
        static float crit_t1; //vars. public
        static float crit_t2; //satic
        char nome[40] ; int numero;
        float nota;
        float nota_criterio(float i,float j,float l,float m){
                return (i*j)+(l*m);}
}
        float aluno::cirterio_teste1;       //declaração
        float aluno::criterio_teste2;       //externa
…
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Exercício:**
  - Reutilize o exemplo anterior:
    - Crie dois construtores da class aluno:
      - aluno();
      - aluno(**char** *n, **int** i).
    - Crie no **main**() mais um aluno, e verifique as notas desse mesmo aluno usando os mesmos critérios;
    - Mude os critérios e verifique as notas.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Exercício (cont.) :**

  – Utilize as seguintes funções para atribuir notas aos alunos:

```
…
main () {
    srand((unsigned) time (NULL));
…}
float aluno::nota() {
     float n; n = rand () % 1501;
     return n/100 + 5;
     }
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Output possível com mudanças de critérios:**
  - **Critérios 1: teste 1 - 50%, teste 2 - 50%;**
  - **Critérios 2: teste 1 - 60%, teste 2 - 40%;**

```
Entrou um aluno!
Nota final: 12.895
Entrou um aluno!
Nota final: 9.365
Prima qualquer tecla para continuar . . .

Notas finais com novos criterios:
Aluno 1: 11.746
Aluno 2: 9.812
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Variável – membro *static* definida como "*private*:":**

```
..
class aluno {
private:
        static int nalunos; //var. private static
public:
        char nome[40] ; int numero;
        aluno(){cout << "Entrou um aluno!\n";
                nalunos++; }
        ~aluno(){ cout << "Saiu um aluno!\n";
                 nalunos--; }
        int getnalunos() {return nalunos;}
};
int aluno::nalunos = 0;
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Variável – membro *static* definida como "*private*:":**

```
...
main () {
    aluno a1;
    cout <<"\nNumero de alunos:" << a1.getnalunos() << '\n';
    {
        aluno a2;
        cout <<"\nNumero de alunos:" << a2.getnalunos() << '\n';
        aluno a3;
        cout <<"\nNumero de alunos:" << a3.getnalunos() << '\n';
        }
     cout <<"\nNumero de alunos:" << a1.getnalunos() << '\n';
     system("pause");
}
```

- **Passagem de objectos para funções:**

– Realizar a **passagem de objectos**, criados a partir de uma **classe**;

– Os objectos passam como **argumentos** pela chamada de uma **função**.

```
…
class aluno {
private:
        float notas;
public:
       char nome[40] ; int numero;
       void muda_nota(float v) {notas = v;}
       void nota() {cout << "A nota e: "<< notas << '\n';}
};
void nota_aluno(aluno a){
    a.nota(); a.muda_nota(9.5);
    a.nota(); }
…
```

- **Exemplo (cont.):**

```
...
main () {
    aluno a1;
    a1.muda_nota(9.4);
    nota_aluno(a1);
    cout << "E capaz de nao ter resultado essa mudanca!\n";
    a1.nota();
    system("pause");
}
```

- **Passagem de objectos para funções:**

```
void nota_aluno (aluno a)
```

– o parâmetro “**a**” é declarado do tipo **aluno**;

– a **função nota_aluno**, recebe uma **cópia** do **objecto** do **tipo aluno**.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Utilização de ponteiros para objectos e passagem por referência:**

  – É igual à **passagem de objectos para funções**, mas por meio de **ponteiros**;

  – A passagem, atribui um **endereço** do **objecto** e permite aceder aos membros desse objecto através do **ponteiro**.

```
...
class aluno {
private:
      float notas;
public:
      char nome[40] ; int numero;
      void muda_nota(float v) {notas = v;}
      void nota() {cout << "A nota e: "<< notas << '\n';}
};
void nota_aluno(aluno *a){
    a->nota(); a->muda_nota(9.5);
    a->nota(); }
...
```

- **Exemplo (cont.):**

```
...
main () {
    aluno a1, *p;
    p = &a1;
    a1.muda_nota(9.4);
    nota_aluno(p);
    cout << "Agora sim, ja conseguimos mudar a nota!\n";
    a1.nota();
    system("pause");
}
...
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Utilização de ponteiros para objectos e passagem por referência:**

```
void nota_aluno (aluno *a)
```

– O **parâmetro** da função é o ponteiro "***a**";

– A **passagem** é feita pelos **argumentos por referência**;

```
a->nota()        //indica a nota do aluno
a->muda_nota(9.5) //renumera o aluno
```

– A função **nota()** executa a **entrada do aluno**;

– A função **muda_nota(9.5)** permite a **renumeração** da nota do aluno com o **parâmetro/número** "9.5".


UNIVERSIDADE da MADEIRA
CTeSP
CURSOS TÉCNICOS
SUPERIORES PROFISSIONAIS