Introdução a Técnicas de Programação

Dados estruturados e enumerações

Prof. André Campos
DIMAp/UFRN

Criandos novos tipos de dados

Tipos primitivos

Tipo de dado básico fornecido pela linguagem (“átomo das informações”)

Na linguagem C++:

- **caractere**: char, unsigned char...
- **inteiro**: int, unsigned int, long...
- **ponto-flutuante**: float, double
- **...**

Tipos não-primitivos

Às vezes, é necessário representar um dado com várias informações

Exemplos:

- Pixel de uma imagem: elementos R, G e B
- Coordenada de um pixel: linha e coluna
- ...

“Tipo estruturado” (struct) em C++

Define um novo tipo de dado com diferentes informações

```
1  struct Nome_do_tipo {
2    // "variáveis" internas do tipo;
3  };
```

Após a declaração, pode-se definir variáveis do tipo e acessar seus campos usando o operador “ . ”

Exemplo

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  int main() {
6    Pixel pixel;
7    pixel.r = 255;
8    pixel.g = 0;
9    pixel.b = 0;
10   //...
11 }
```

Exemplo

Escreva uma função que lê 2 pontos no plano (x, y) e imprime a distância entre eles.

Exemplos de entrada e saída

Entrada	Saída
`distancia(10, 0, 0, 0)`	`10.00`
`distancia(4, 0, 0, 3)`	`5.00`
`distancia(19, -5, -6, 14)`	`31.40`

Exemplo - solução 1

Escreva uma função que lê 2 pontos no plano (x, y) e imprime a distância entre eles.

```
1  double distancia(double x1, double y1, double x2, double y2) {
2    double dx = x1 - x2;
3    double dy = y1 - y2;
4    return sqrt(dx*dx + dy*dy);
5  }
6
7  TEST_CASE("Testando a distância entre dois pontos") {
8    CHECK(distancia(10, 0, 0, 0) == doctest::Approx(10));
9    CHECK(distancia(4, 0, 0, 3) == doctest::Approx(5));
10   CHECK(distancia(19, -5, -6, 14) == doctest::Approx(31.4));
11 }
```

Exemplo - solução 2

```cpp
struct Point {
  double x, y;
};

double distancia(Point a, Point b) {
  double dx = a.x - b.x;
  double dy = a.y - b.y;
  return sqrt(dx*dx + dy*dy);
}

TEST_CASE("Testando a distância entre dois pontos") {
  CHECK(distancia(Point{10, 10}, Point{0, 0}) == doctest::Approx(10));
  CHECK(distancia(Point{4, 0}, Point{0, 3}) == doctest::Approx(5));
  CHECK(distancia(Point{19, -5}, Point{-6, 14}) == doctest::Approx(31.4));
}
```

Tipos dos campos

Os campos podem ser de diferentes tipos, inclusive de outros tipos **struct**

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  struct Image {
6    int width, height;
7    Pixel pixels[256][256];
8  };
```

Inicialização dos campos

É possível inicializar os valores dos campos usando { }, seguindo a mesma ordem da definição dos campos

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  struct Point {
6    double x, y;
7  };
8
9  int main() {
10   Point pt {1, 2};
11   Pixel px {255, 0, 0};
12   // ...
13 }
```

Inicialização dos campos

Usando arrays/matrizes...

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  struct Image {
6    int width, height;
7    Pixel pixels[256][256];
8  };
9
10 int main() {
11   Image img = {3, 3, {
12     {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}},
13     {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}},
14     {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}},
15   }};
16   // ...
17 }
```

Passagem de parâmetros

Os valores dos campos são copiados para uma variável local (do mesmo tipo).

Que valor será impresso?

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4  void toGray(Pixel p) {
5    int gray = (p.r + p.g + p.b) / 3;
6    p.r = gray;
7    p.g = gray;
8    p.b = gray;
9  }
10 int main() {
11   Pixel p = {255, 0, 0};
12   toGray(p);
13   cout << (int) p.r << endl;
14   cout << (int) p.g << endl;
15   cout << (int) p.b << endl;
16   // ...
17 }
```

Passagem de parâmetros

Os valores dos campos são copiados para uma variável local (do mesmo tipo).

Que valor será impresso?

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4  void toGray(Pixel &p) {
5    int gray = (p.r + p.g + p.b) / 3;
6    p.r = gray;
7    p.g = gray;
8    p.b = gray;
9  }
10 int main() {
11   Pixel p = {255, 0, 0};
12   toGray(p);
13   cout << (int) p.r << endl;
14   cout << (int) p.g << endl;
15   cout << (int) p.b << endl;
16   // ...
17 }
```

Passagem de parâmetros

Os valores dos campos serão copiados mesmo que sejam arrays/matrizes

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  struct Image {
6    int width, height;
7    Pixel pixels[256][256];
8  };
```

```
1  void printImage(Image img) {
2    cout << img.width << " " << img.height << endl;
3    for (int i = 0; i < img.height; i++) {
4      for (int j = 0; j < img.width; j++) {
5        Pixel p = img.pixels[i][j];
6        cout << (int) p.r << " ";
7        cout << (int) p.g << " ";
8        cout << (int) p.b << " ";
9      }
10     cout << endl;
11   }
12 }
```

Passagem de parâmetros

A referência otimiza os recursos computacionais (memória e processamento)

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  struct Image {
6    int width, height;
7    Pixel pixels[256][256];
8  };
```

```
1  void printImage(Image &img) {
2    cout << img.width << " " << img.height << endl;
3    for (int i = 0; i < img.height; i++) {
4      for (int j = 0; j < img.width; j++) {
5        Pixel p = img.pixels[i][j];
6        cout << (int) p.r << " ";
7        cout << (int) p.g << " ";
8        cout << (int) p.b << " ";
9      }
10     cout << endl;
11   }
12 }
```

Passagem de parâmetros

Quando não alteramos os dados, podemos especificar que o parâmetro é `const`

```
1  struct Pixel {
2    unsigned char r, g, b;
3  };
4
5  struct Image {
6    int width, height;
7    Pixel pixels[256][256];
8  };
```

```
1  void printImage(const Image &img) {
2    cout << img.width << " " << img.height << endl;
3    for (int i = 0; i < img.height; i++) {
4      for (int j = 0; j < img.width; j++) {
5        Pixel p = img.pixels[i][j];
6        cout << (int) p.r << " ";
7        cout << (int) p.g << " ";
8        cout << (int) p.b << " ";
9      }
10     cout << endl;
11   }
12 }
```

Enumerações

Define um novo tipo que pode assumir apenas conjunto predefinido de valores

```
1  enum ChessPiece {
2    Pawn, Knight, Bishop, Rook, Queen, King
3  };
4
5  int main() {
6    ChessPiece p = Pawn;
7    ChessPiece q = Queen;
8    // ...
9  }
```

Enumerações

Associa os valores do tipo a valores inteiros, começando do 0.

```
1  enum ChessPiece {
2    Pawn, Knight, Bishop, Rook, Queen, King
3  };
4
5  int main() {
6    ChessPiece p = Pawn;
7    ChessPiece q = Queen;
8    cout << q << endl;  // imprime 4
9  }
```

Prática

1. Defina tipos para os possíveis valores nas cartas no baralho (Ás, 2, 3,... 10, Valete, Dama, Rei) e os possíveis naipes (Ouros, Copas, Paus, Espadas).
2. Defina o tipo `Carta` que é composto por duas informações: seu valor (Ás, 2,... Rei) e seu naipe.
3. Defina o tipo `Cartas` que representa um conjunto de cartas e é composto pela quantidade de cartas e array de até 52 cartas.
4. Implemente uma função que recebe um conjunto de cartas e verifica se há no mínimo 3 cartas do mesmo naipe em sequência (o ás vem tanto antes do 2, quanto depois do rei).