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

Sintaxe

```
struct Nome_do_tipo {
  campos (variáveis internas do tipo);
};
```

Exemplo

```
struct Pixel {
  unsigned char r, g, b;
};
```

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

Exemplo

```
Pixel pixel;
pixel.r = 255;
pixel.g = 0;
pixel.b = 0;
```

Exemplo

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

Exemplos de entrada e saída

Entrada	Saída
10 0 0 0	10.00
4 0 0 3	5.00
19 -5 -6 14	31.40

Exemplo - solução 1

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

Exemplos de entrada

Entrada	Saída
10 0 0 0	10.00
4 0 0 3	5.00
19 -5 -6 14	31.40

```
double distance(double x1, double y1, double x2, double y2) {
  double dx = x1 - x2;
  double dy = y1 - y2;
  return sqrt(dx*dx + dy*dy);
}

int main() {
  double x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
  cout << distance(x1, y1, x2, y2) << endl;
  return 0;
}
```

Exemplo - solução 2

Escreva um programa que lê 2 pontos no
entre eles.

Exemplos de entrada e saída

Entrada	Saída
10 0 0 0	10.00
4 0 0 3	5.00
19 -5 -6 14	31.40

```
struct Point {
  double x, y;
};
double distance(Point a, Point b) {
  double dx = a.x - b.x;
  double dy = a.y - b.y;
  return sqrt(dx*dx + dy*dy);
}
int main() {
  Point p1, p2;
  cin >> p1.x >> p1.y >> p2.x >> p2.y;
  cout << distance(p1, p2) << endl;
  return 0;
}
```

Tipos dos campos

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

```
struct Pixel {
  unsigned char r, g, b;
};

struct Image {
  int width, height;
  Pixel pixels[256][256];
};
```

Inicialização dos campos

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

```
struct Pixel {
  unsigned char r, g, b;
};

struct Point {
  double x, y;
};

int main() {
  Point pt {1, 2};
  Pixel px {255, 0, 0};
  ...
}
```

Inicialização dos campos

Usando arrays/matrizes...

```
struct Pixel {
  unsigned char r, g, b;
};

struct Image {
  int width, height;
  Pixel pixels[256][256];
};

int main() {
  Image img = {3, 3, {
    {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}},
    {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}},
    {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}},
  }};
  ...
}
```

Passagem de parâmetros

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

Que valor será impresso?

```
struct Pixel {
  unsigned char r, g, b;
};
void toGray(Pixel p) {
  int gray = (p.r + p.g + p.b) / 3;
  p.r = gray;
  p.g = gray;
  p.b = gray;
}
int main() {
  Pixel p = {255, 0, 0};
  toGray(p);
  cout << (int)p.r << endl;
}
```

Passagem de parâmetros

Se for passada a referência, o parâmetro acessa a variável da função de origem.

Que valor será impresso?

```
struct Pixel {
  unsigned char r, g, b;
};
void toGray(Pixel &p) {
  int gray = (p.r + p.g + p.b) / 3;
  p.r = gray;
  p.g = gray;
  p.b = gray;
}
int main() {
  Pixel p = {255, 0, 0};
  toGray(p);
  cout << (int)p.r << endl;
}
```

Passagem de parâmetros

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

```
struct Pixel {
  unsigned char r, g, b;
};

struct Image {
  int width, height;
  Pixel pixels[256][256];
};
```

```
void printImage(Image img) {
  cout << img.width << " " << img.height << endl;
  for (int i = 0; i < img.height; i++) {
    for (int j = 0; j < img.width; j++) {
      Pixel p = img.pixels[i][j];
      cout << (int)p.r << " ";
      cout << (int)p.g << " ";
      cout << (int)p.b << " ";
    }
    cout << endl;
  }
}
```

Passagem de parâmetros

A referência otimiza os recursos computacionais (memória e processamento)

```
struct Pixel {
  unsigned char r, g, b;
};

struct Image {
  int width, height;
  Pixel pixels[256][256];
};
```

```
void printImage(Image &img) {
  cout << img.width << " " << img.height << endl;
  for (int i = 0; i < img.height; i++) {
    for (int j = 0; j < img.width; j++) {
      Pixel p = img.pixels[i][j];
      cout << (int)p.r << " ";
      cout << (int)p.g << " ";
      cout << (int)p.b << " ";
    }
    cout << endl;
  }
}
```

Passagem de parâmetros

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

```
struct Pixel {
  unsigned char r, g, b;
};

struct Image {
  int width, height;
  Pixel pixels[256][256];
};
```

```
void printImage(const Image &img) {
  cout << img.width << " " << img.height << endl;
  for (int i = 0; i < img.height; i++) {
    for (int j = 0; j < img.width; j++) {
      Pixel p = img.pixels[i][j];
      cout << (int)p.r << " ";
      cout << (int)p.g << " ";
      cout << (int)p.b << " ";
    }
    cout << endl;
  }
}
```

Enumerações

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

```
enum ChessPiece {
  Pawn, Knight, Bishop, Rook, Queen, King
};

int main() {
  ChessPiece p = Pawn;
  ChessPiece q = Queen;
  ...
}
```

Enumerações

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

```
enum ChessPiece {
  Pawn, Knight, Bishop, Rook, Queen, King
};

int main() {
  ChessPiece p = Pawn;
  ChessPiece q = Queen;
  cout << q << endl;  // imprime 4
}
```

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).