Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Tipos Primitivos

Tipos básicos do C++:

`bool`	`int`	`unsigned long int`
`char`	`long int`	`float`
`unsigned char`	`unsigned short int`	`double`
`short int`	`unsigned int`	`long double`

É possível criar novos tipos compostos por tipos básicos.

Estruturas em C++

Agrupe várias variáveis de tipos diferentes em uma única entidade, chamada de estrutura.

```
struct Retangulo {
    int largura;
    int altura;
};
```

⚠️ Não se esqueça do `;` no final da definição.

- A palavra-chave `struct` é usada para declarar uma estrutura.
- Dentro da estrutura, podemos definir variáveis, chamadas de membros, que podem ter diferentes tipos de dados.

Por que usar Estruturas?

- Estruturas permitem agrupar diferentes tipos de dados em uma única entidade lógica, facilitando o gerenciamento e a manipulação de informações complexas.
- As estruturas podem ser reutilizadas em diferentes partes do programa, promovendo a consistência no código.
- Estruturas tornam o código mais legível e fácil de manter, pois agrupam dados relacionados de forma intuitiva, refletindo a lógica do problema.
- Estruturas são a base para conceitos mais avançados como classes e objetos na programação orientada a objetos.

Usando Estruturas

Definição

Declaração da estrutura.

```
struct Retangulo {
    int largura;
    int altura;
};
```

Instanciação

Variáveis do tipo da estrutura.

```
int main() {
    Retangulo caixa;
}
```

A estrutura `Retangulo` é vista como um novo tipo pelo C++.

Acessando Membros

O operador ponto (`.`) é utilizado para acessar e modificar os valores dos membros da estrutura.

```
int main() {
    Retangulo caixa;

    caixa.largura = 10;
    caixa.altura = 20;

    cout << "Largura: "
         << caixa.largura << "\n"
         << "Altura: "
         << caixa.altura << "\n";
}
```

❌ **Não funciona**

```
int main() {
    Retangulo caixa;
    cin >> caixa;   // erro
    cout << caixa;  // erro
}
```

O que aconteceria se `caixa.largura` não fosse inicializada ❓

Inicializando Membros de Estruturas

```
struct InfoCidade {
    string nome;
    char estado[3];
    int populacao;
};
```

Lista de Inicialização

```
InfoCidade floripa = {"Florianópolis", "UF", 537211};
```

Inicialização Parcial

```
InfoCidade floripa = {"Florianópolis"};
```

Se você deixar um membro da estrutura sem inicialização, todos os membros seguintes também devem ficar sem inicialização.

Comparando Estruturas

```
struct Retangulo {
    int largura;
    int altura;
};
```

✅ Correto

```
int main() {
    Retangulo caixa1 = {10, 20}, caixa2;
    caixa2 = caixa1;  // Cópia
    cout << caixa1.largura << " X "
         << caixa1.altura << "\n";
    cout << caixa2.largura << " X "
         << caixa2.altura << "\n";
    cout << (caixa1.largura == caixa2.largura &&
             caixa1.altura == caixa2.altura)
         << "\n";
    return 0;
}
```

❌ Incorreto

```
int main() {
    Retangulo caixa1 = {10, 20}, caixa2;
    caixa2 = caixa1;  // Cópia
    cout << caixa1.largura << " X "
         << caixa1.altura << "\n";
    cout << caixa2.largura << " X "
         << caixa2.altura << "\n";
    cout << (caixa1 == caixa2)  // Erro
         << "\n";
    return 0;
}
```

Exemplo com e sem `struct`

```
int main() {
    int emp_id = 5658845;

    cout << "Insira o número de horas "
         << "trabalhadas pelo funcionário "
         << emp_id << "\n";
    int horas;
    cin >> horas;

    cout << "Insira o valor pago por hora "
         << "trabalhada pelo funcionário "
         << emp_id << "\n";
    double valor_hora;
    cin >> valor_hora;

    double salario = horas * valor_hora;

    cout << "O salário bruto do funcionário "
         << emp_id << " é R$"
         << salario << "\n";

    return 0;
}
```

```
struct Funcionario {
    int emp_id;
    int horas;
    double valor_hora;
    double salario;
};
```

```
int main() {
    Funcionario funci = {5658845};

    cout << "Insira o número de horas "
         << "trabalhadas pelo funcionário "
         << funci.emp_id << "\n";
    cin >> funci.horas;

    cout << "Insira o valor pago por hora "
         << "trabalhada pelo funcionário "
         << funci.emp_id << "\n";
    cin >> funci.valor_hora;

    funci.salario = funci.horas * funci.valor_hora;

    cout << "O salário bruto do funcionário "
         << funci.emp_id << " é R$"
         << funci.salario << "\n";

    return 0;
}
```

Arrays de Estruturas

É possível construir um array de estruturas.

```
struct Aluno {
    string nome;
    float nota_prova;
    float nota_trabalho;
};
```

```
Aluno alunos[3] = {{"Alice", 8.0, 10.0},
                   {"Bob",   9.5, 10.0},
                   {"Eva",   6.7, 8.2}};
```

- Permite armazenar múltiplas instâncias da estrutura em uma única variável.
- Simplifica o gerenciamento de dados relacionados.

Passando Estruturas para Funções

```
struct Retangulo {
    int largura;
    int altura;
};
```

Por Valor:

```
void print_retangulo(Retangulo);

void print_retangulo(Retangulo r) {
    cout << r.largura << " X "
         << r.altura << '\n';
}
```

Por Referência:

```
void construir_retangulo(Retangulo&);

void construir_retangulo(Retangulo &r) {
    cout << "Largura: ";
    cin >> r.largura;

    cout << "Altura: ";
    cin >> r.altura;
}
```

Passar Estruturas por Referência é Mais Eficiente

- Passar por valor copia toda a estrutura para dentro da função.
- Quando passamos por referência, não é feita cópia.
- Como a `struct` pode ter vários membros, copiar a `struct` pode ser lento.
- O ideal é usar passagem por referência com `const` se não for alterar o valor.

```
void print_retangulo(const Retangulo&);

void print_retangulo(const Retangulo &r) {
    cout << r.largura << " X "
         << r.altura << '\n';
}
```

Retornando Estruturas de Funções

```
struct Retangulo;
Retangulo criar_retangulo(int, int);

struct Retangulo {
    int largura;
    int altura;
    int area;
};

Retangulo criar_retangulo(int l, int a) {
    Retangulo r = {l, a, l * a};
    return r;
}
```

- É possível retornar uma `struct` de uma função.
- É necessário definir a `struct` antes da função.
- Uma `struct` pode ser usada para retornar vários valores.

II Lista - Exercício 6: Com Múltiplos Arrays

```
int main() {

    int emp_ids[] = {5658845, 4520125, 7895122,
                     8777541, 8451277, 1302850,
                     7580489};

    const int tamanho = sizeof(emp_ids) / sizeof(int);

    int horas[tamanho];
    double valor_hora[tamanho], salario[tamanho];

    for (int i = 0; i < tamanho; i++) {
        salario[i] = informar_valores(
            emp_ids[i],
            horas[i],
            valor_hora[i]
        );
    }

    for (int i = 0; i < tamanho; i++) {
        relatorio(emp_ids[i], salario[i]);
    }

    return 0;
}
```

```
double informar_valores(
    int emp_id,
    int &horas,
    double &valor_hora
) {
    std::cout << "Insira o número de horas "
              << "trabalhadas pelo funcionário "
              << emp_id << "\n";
    std::cin >> horas;
    std::cout << "Insira o valor pago por hora "
              << "trabalhada pelo funcionário "
              << emp_id << "\n";
    std::cin >> valor_hora;

    double salario = horas * valor_hora;
    return salario;
}

void relatorio(int emp_id, double salario) {
    std::cout << "O salário bruto do funcionário "
              << emp_id << " é " << salario
              << "\n";
}
```

II Lista - Exercício 6: Com Array de `struct`

```
struct Funcionario {
    int emp_id;
    int horas;
    double valor_hora;
    double salario;
};

int main() {
    int emp_ids[] = {5658845, 4520125, 7895122,
                     8777541, 8451277, 1302850,
                     7580489};
    const int tamanho = sizeof(emp_ids) / sizeof(int);

    Funcionario funcionarios[tamanho];

    for (int i = 0; i < tamanho; i++) {
        funcionarios[i] = informar_valores(emp_ids[i]);
    }

    for (const auto &funcionario : funcionarios) {
        relatorio(funcionario);
    }
    return 0;
}
```

```
Funcionario informar_valores(int id) {
    Funcionario f = {id};

    std::cout << "Insira o número de horas "
              << "trabalhadas pelo funcionário "
              << id << "\n";
    std::cin >> f.horas;
    std::cout << "Insira o valor pago por hora "
              << "trabalhada pelo funcionário "
              << id << "\n";
    std::cin >> f.valor_hora;

    f.salario = f.horas * f.valor_hora;

    return f;
}

void relatorio(const Funcionario &f) {
    std::cout << "O salário bruto do funcionário "
              << f.emp_id << " é "
              << f.salario << "\n";
}
```

Estruturas Aninhadas

Estruturas dentro de outras estruturas permitem representar relações mais complexas entre dados.

```
struct Ponto {
    int x, y;
};

struct Linha {
    Ponto inicio;
    Ponto fim;
};

void print_linha(const Linha &l) {
    std::cout << "("
              << l.inicio.x << ", "
              << l.inicio.y << ") <-> ("
              << l.fim.x << ", "
              << l.fim.y
              << ")\n";
}
```

```
int main() {
    Ponto p1 = {4, 10};
    Ponto p2 = {33, 2};
    Linha l1 = {p1, p2};

    Linha l2 = {{10, 20},
                {13, 50}};

    Linha l3 = {{7},
                {88, 27}};

    print_linha(l1);
    print_linha(l2);
    print_linha(l3);

    return 0;
}
```

Exemplo: Registro de Cliente (1/2)

```
struct Data {
    unsigned short dia, mes, ano;
};

struct Endereco {
    std::string logradouro, cidade;
    char estado[3];
    char cep[9];
};

struct Cliente {
    std::string nome;
    Data nascimento;
    Endereco endereco;
    Data abertura_conta;
};

Data pegar_data(const char msg[]) {
    Data data;

    std::cout << "Informe a data de " << msg
              << " (formato: DD MM AAA)\n";
    std::cin >> data.dia >> data.mes >> data.ano;

    return data;
}
```

```
void imprimir_data(const Data &d) {
    std::cout << d.dia << "/"
              << d.mes << "/"
              << d.ano << "\n";
}

Cliente registrar_cliente() {
    std::string nome;
    std::cout << "Nome do cliente: ";
    std::getline(std::cin, nome);

    Cliente c = {
        nome,
        pegar_data("nascimento"),
        {},
        pegar_data("hoje"),
    };

    return c;
}

void imprimir_cadastro(const Cliente &c) {
    std::cout << "Cliente: " << c.nome << "\n"
              << "Data de nascimento: ";
    imprimir_data(c.nascimento);
    std::cout << "Cliente desde: ";
    imprimir_data(c.abertura_conta);
}
```

Exemplo: Registro de Cliente (2/2)

```
int main(int argc, char const *argv[]) {
    auto c = registrar_cliente();

    imprimir_cadastro(c);

    return 0;
}
```

⚙️ Exercício (continua ➡️)

Desenvolva um sistema para gerenciar o cadastro de produtos.

informações do produto

- Código do produto (`int`)
- Nome do produto (`std::string`)
- Quantidade em estoque (`int`)
- Preço unitário (`double`)

1. Crie uma estrutura para armazenar as informações.
2. Criar uma função para pegar os dados do produto.
3. Criar uma função que exiba as informações de um produtos.
4. Declarar um array de 5 produtos e exibir as informações desses produtos.

Código Base:

```cpp
#include <iostream>

struct Produto;

Produto registrar_produto();
void imprimir_relatorio(const Produto&);

int main() {
    Produto produtos[5];

    // Registre os produtos
    for (auto &p : produtos) {
        p = registrar_produto();
    }

    std::cout << "Relatório de Produtos\n";
    for (const auto &p : produtos) {
        imprimir_relatorio(p);
    }

    return 0;
}
```

Exemplo de Saída Esperada:

```
Código do produto: 101
Nome do produto: Caneta Azul
Quantidade em estoque: 200
Preço unitário: 1.50

Código do produto: 102
Nome do produto: Caderno
Quantidade em estoque: 150
Preço unitário: 5.75

...

Relatório de Produtos:
Código: 101 | Nome: Caneta Azul | Quantidade: 200 | Preço: 1.5 | Valor Total: 300
Código: 102 | Nome: Caderno | Quantidade: 150 | Preço: 5.75 | Valor Total: 862.5
...
```

Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Evandro Chagas Ribeiro da Rosa