Algoritmos II

STRUCTS (ESTRUTURAS)

Conceito

- “Tipo” criado pelo programador que permite a armazenagem de dados de tipos diferentes.
- As structs consistem em criar apenas um dado que contém vários membros, que nada mais são do que outras variáveis. De uma forma mais simplificada, é como se uma variável tivesse outras variáveis dentro dela.
- Struct, ou estrutura, é um bloco que armazena diversas informações.

Vantagem de Utilização

- Pode-se agrupar de forma organizada vários tipos de dados diferentes. Por exemplo, dentro de uma estrutura de dados pode-se ter juntos um tipo float, um inteiro, um char ou um double.

```
1   #include <iostream>
2   using namespace std;
3
4   struct Jogador {
5       string nomeDoJogador;
6       int forca, destreza, inteligencia;
7       //E os outros itens que fossem necessários
8   };
9
10  int main () {
11
12      Jogador j1, j2; //E quantos jogadores fossem necessarios
13
14      return 0;
15  }
16
```

Utilizando struct é possível agrupar as informações. Caso não fossem utilizadas, seria necessário replicar as variáveis criadas para cada jogador (nomeDoJogador1, forca1, destreza1, inteligencia, nomeDoJogador2, …). Utilizando struct são agrupadas as informações comuns, e utilizada somente a struct na chamada.

Onde utilizar as Estruturas

- Structs são muito usadas quando tem-se elementos nos programas que precisam e fazem uso de vários tipos de variáveis e características.
- Usando struct, pode-se trabalhar com vários tipos de informações de uma maneira mais fácil, rápida e organizada, uma vez que não é necessário se preocupar em declarar e decorar o nome de cada elemento da struct.

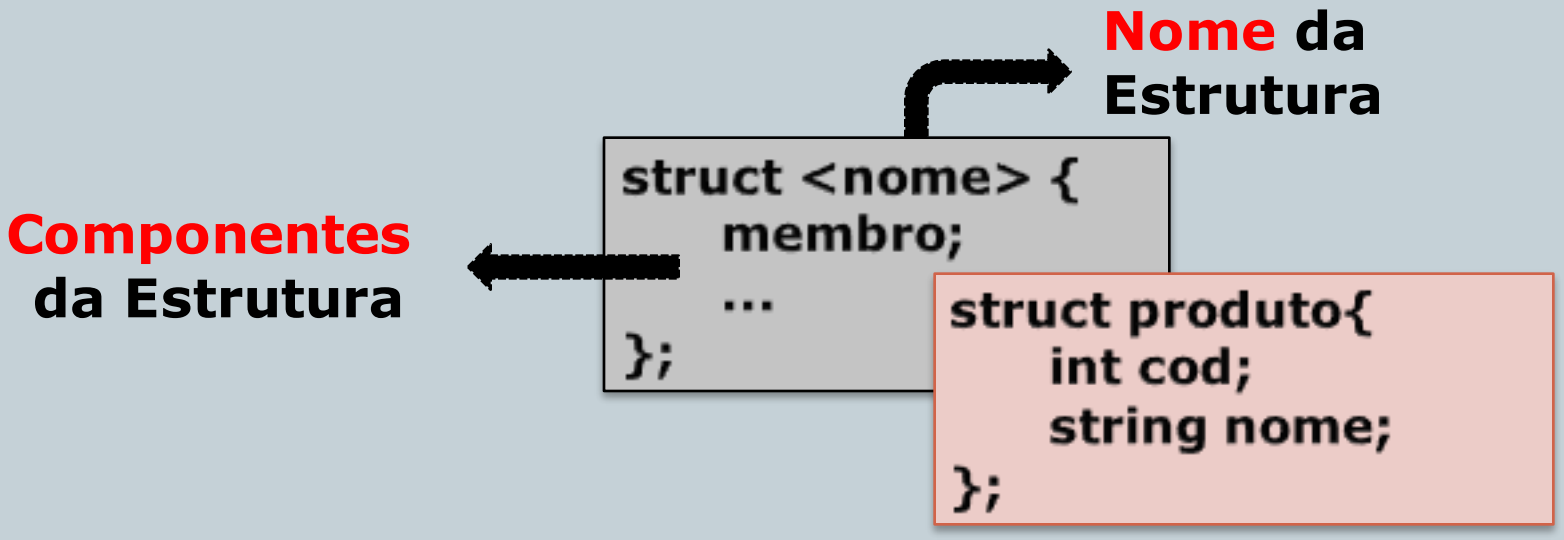
Onde utilizar as Estruturas

- Exemplo:
  - Vamos supor que você foi contratado para criar um aplicativo de uma escola. As structs servem para organizar as informações de uma maneira mais otimizada. Para isso, basta colocar as informações comuns na estrutura.
    - Quais seriam os elementos comuns que deveriam ser colocados nessa estrutura?
      - Como vai-se trabalhar com alunos, então é necessário colocar elementos na struct que representem os alunos: nome, notas, mensalidade, se esta mensalidade foi paga ou não etc. Assim, pode-se cria uma variável struct para cada aluno, ou um vetor de struct e, automaticamente, esse aluno terá as variáveis acima citadas.

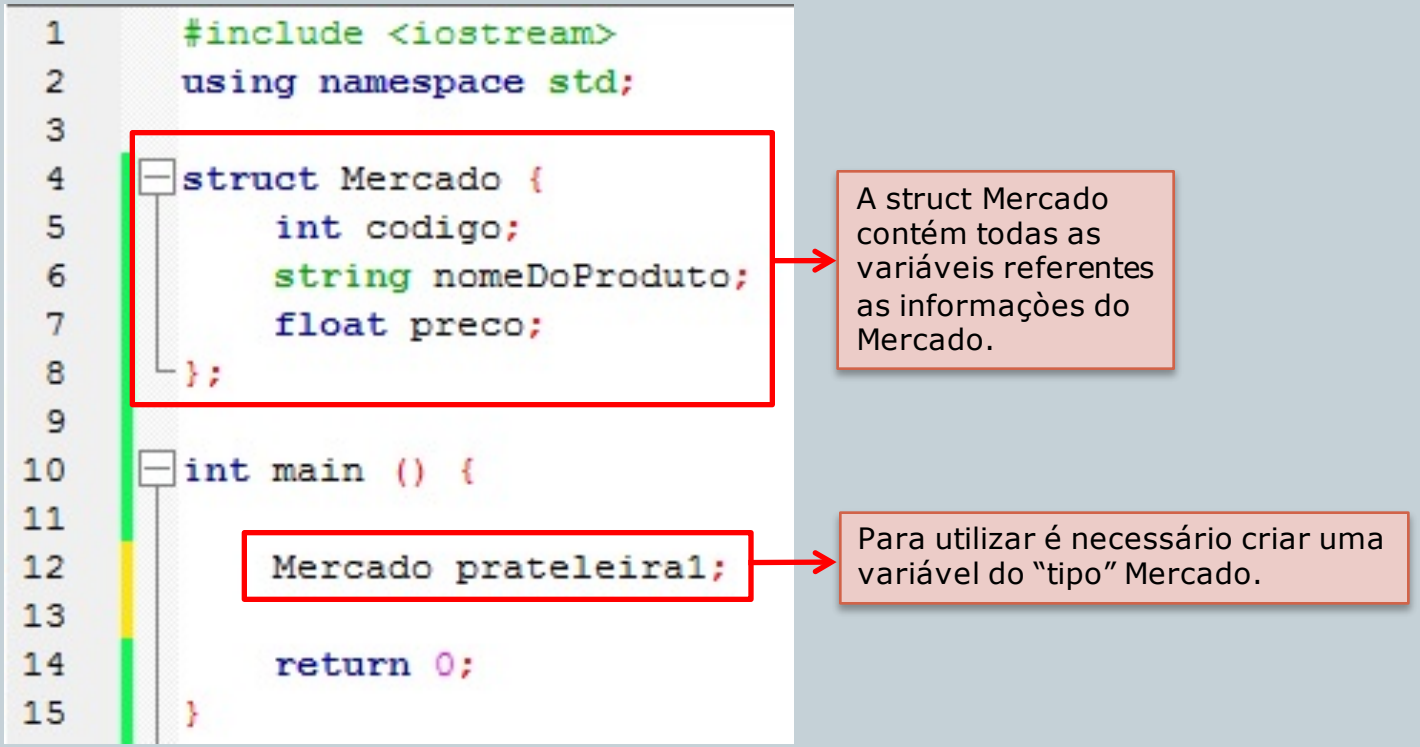
Sintaxe

- Para criar uma estrutura de dados é utilizada a palavra reservada struct.
- Toda estrutura deve ser criada antes de qualquer função ou mesmo da função principal main.
- Toda estrutura tem nome e seus membros são declarados dentro de um bloco de dados.
- Após a definição de seus membros no bloco de dados, termina-se a linha com um ponto-e-vírgula(;).

Sintaxe


Nome da Estrutura
struct <nome> {
membro;
…
};
Componentes da Estrutura
struct produto{
int cod;
string nome;
};

Sintaxe



```
1   #include <iostream>
2   using namespace std;
3
4   struct Mercado {
5       int codigo;
6       string nomeDoProduto;
7       float preco;
8   };
9
10  int main () {
11
12      Mercado prateleira1;
13
14      return 0;
15  }
```

Sintaxe

```
1   #include <iostream>
2   using namespace std;
3   #include <stdio.h>
4
5   struct Mercado {
6       int codigo;
7       string nomeDoProduto;
8       float preco;
9   };
10
11  int main () {
12
13      Mercado prateleira1;
14
15      cout<<"Codigo: ";
16      cin>> prateleira1.codigo;
17      fflush(stdin);
18      cout<<"Nome do Produto: ";
19      getline(cin, prateleira1.nomeDoProduto);
20      cout<<"Preco: ";
21      cin>> prateleira1.preco;
22
23      return 0;
24  }
```

Para acessar um membro da struct chama-se pelo nome da variável criada, ponto, o nome da variável a ser acessada na struct.

variavelCriada.variavelStruct
prateleira1.preco

Estruturas

- Uma struct pode ser tanto global quanto local. A struct local será válida somente na função onde foi declarada, e a struct global por todas as funções abaixo de sua declaração.
- Uma struct pode ser atribuída a outra do mesmo tipo.

```
1   #include <iostream>
2   using namespace std;
3   #include <stdio.h>
4
5   struct Mercado {
6       int codigo;
7       string nomeDoProduto;
8       float preco;
9   };
10
11  int main () {
12
13      Mercado prateleira1 = {1,"Biscoitos",4.50};
14
15      Mercado prateleira2 = prateleira1;
16
17      cout<<prateleira2.nomeDoProduto;
18
19      return 0;
20  }
```

Vetores

- É possível criar vetores cujos elementos são estruturas.

```
1   #include <iostream>
2   using namespace std;
3   #include <stdio.h>
4
5   #define TAM 5
6
7   struct Mercado {
8       int codigo;
9       string nomeDoProduto;
10      float preco;
11  };
12
13  int main () {
14
15      Mercado produtos[TAM];
16
17      for (int i=0; i<TAM; i++) {
18          cout<<"Codigo: ";
19          cin>>produtos[i].codigo;
20          cout<<"Produto: ";
21          fflush(stdin);
22          getline(cin,produtos[i].nomeDoProduto);
23          cout<<"Preco: ";
24          cin>>produtos[i].preco;
25      }
26      return 0;
27  }
```

- Quando se cria um vetor do tipo de uma estrutura, em cada índice do vetor pode-se encontrar todos os dados da estrutura. Por exemplo, na estrutura abaixo:

▪ codigo ▪ nomeDo Produto ▪ preco	▪ codigo ▪ nomeDo Produto ▪ preco	▪ codigo ▪ nomeDo Produto ▪ preco	▪ codigo ▪ nomeDo Produto ▪ preco	▪ codigo ▪ nomeDo Produto ▪ preco
0	1	2	3	4

Vetor do tipo estrutura x Vetor dentro da estrutura

- Quando se cria um vetor do tipo estrutura em cada índice do vetor estão as informações da estrutura (no caso, em cada índice tem-se uma informação de codigo, nomeDoProduto e preco.

```
1   #include <iostream>
2   using namespace std;
3   #include <stdio.h>
4
5   #define TAM 5
6
7   struct Mercado {
8       int codigo;
9       string nomeDoProduto;
10      float preco;
11  };
12
13  int main () {
14
15      Mercado produtos[TAM];
16
17      return 0;
18  }
```

- Quando se cria um vetor dentro da estrutura é como uma variável da estrutura. No caso, tem-se uma variável produtos, e dentro dela será cadastrado um código, um nome e 5 preços desse produto.

```
1   #include <iostream>
2   using namespace std;
3   #include <stdio.h>
4
5   #define TAM 5
6
7   struct Mercado {
8       int codigo;
9       string nomeDoProduto;
10      float preco[TAM];
11  };
12
13  int main () {
14
15      Mercado produtos;
16
17      return 0;
18  }
```

Estruturas Aninhadas

```
1  #include <iostream>
2  using namespace std;
3
4  struct data {
5    int dia, mes, ano;
6  };
7
8  struct Produto {
9    int codigo;
10   data entrega;
11   data validade;
12 };
13
14 int main () {
15     Produto mercado1;
16     cout<<"Codigo: ";
17     cin>>mercado1.codigo;
18
19     cout<<"Data de Entrega: ";
20     cin>>mercado1.entrega.dia;
21     cin>>mercado1.entrega.mes;
22     cin>>mercado1.entrega.ano;
23
24     cout<<"Data de Validade: ";
25     cin>>mercado1.validade.dia;
26     cin>>mercado1.validade.mes;
27     cin>>mercado1.validade.ano;
28
29     return 0;
30 }
```

- Podem ser criadas estruturas aninhadas, ou seja, uma estrutura dentro da outra.

Foram criadas duas variáveis do tipo data. No formato normal seriam criadas 3 variáveis para entrega (dia, mes e ano) e 3 variáveis para validade (dia, mês e ano).

Em caso de uma estrutura aninhada a outra, a forma de chamada é:

variável.variavelStruct1.variavelStruct2

mercado1.entrega.dia
Mercado1.validade.dia

Passagem de Parâmetro e Retorno de Função

```
1  #include <iostream>
2  using namespace std;
3
4  struct teste {
5    int a, b,res;
6  };
7
8  teste passagem_referencia (teste &y) {
9    y.a = y.a + y.b;
10   return (y);
11 }
12
13 void imprime (teste y) {
14   cout<<y.a<<"\t"<<y.b;
15 }
16
17 int main() {
18   teste x;
19   x.a = 2;
20   x.b = 4;
21   imprime(passagem_referencia(x));
22   return 0;
23 }
```

- Pode-se passar uma variável do tipo estrutura por parâmetro, bem como uma função pode retornar uma estrutura.