

CCP130

Desenvolvimento de Algoritmos

Prof. Danilo H. Perico





typedef

typedef

- ***typedef*** é uma palavra chave em C utilizada para criar um novo tipo baseado em um já existente
- O objetivo do ***typedef*** é criar um novo tipo que tenha um nome relevante para o problema em questão

typedef

- Sintaxe:

```
typedef tipo_existente nome_do_tipo;
```

typedef

- Exemplos:

```
typedef int capacidade;
```

```
typedef int size_t;
```

```
typedef float moeda;
```

typedef

- O *typedef* também pode ser utilizado em conjunto com *structs*:

```
typedef struct{  
    int x;  
    int y;  
}ponto;
```

Nome do novo tipo



typedef

- O *typedef* também pode ser utilizado em conjunto com *structs*:

```
typedef struct{  
    int x;  
    int y;  
}ponto;
```

Então, para criar uma variável do tipo ponto:

```
ponto p;
```



enum

enum

- Assim como o *typedef* e *struct*, o *enum* é um tipo de dado definido pelo programador
- Seu nome vem da palavra *enumeration*. Assim, um *enum* é um tipo enumerado e consiste num conjunto de constantes inteiras
- Cada uma destas constantes inteiras é representada por um *nome*

enum

- Tipos enumerados são usados quando conhecemos o conjunto de valores que uma variável pode assumir
- As enumerações são definidas de forma semelhante às estruturas:

```
enum identificador_enum{  
    lista de enumeração  
} lista_variáveis;
```

enum

- Exemplos: `enum moeda {real, dolar, nickel, libras};`
`enum {ensolarado, chuvoso, nebuloso} tempo;`
`enum dispositivo {`
 notebook,
 tablet,
 celular,
 desktop}
`disp_usuario;`

enum

- Exemplos: `enum moeda {real, dolar, nickel, libras};`

`enum {ensolarado, chuvoso, nebuloso} tempo;`

`enum dispositivo {
 notebook,
 tablet,
 celular,
 desktop}`

`disp_usuario;`

O **identificador** do `enum` e a **lista de variáveis** nunca poderão ser omitidos ao mesmo tempo!

enum

- Declarando um enum, depois uma variável do tipo enum.

```
//Definindo o tipo enum
enum situacao {otima, boa, regular, ruim, pessima};

//Declarando a variável que utiliza o enum definido
anteriormente
enum situacao s_dinheiro;

//Atribuindo um valor a variável:
s_dinheiro = otima;
```

enum

- O ponto chave para entendermos o enum é pensar que cada símbolo representa um valor inteiro:

```
void main() {  
    //Atribuindo um valor a variável:  
    s_dinheiro = otima;  
    printf("%d", s_dinheiro); //Imprime 0  
    printf("%d", otima);      //Imprime 0  
    printf("%d", pessima);    //Imprime 4  
}
```

enum

- Podemos mudar a representação inteira de um símbolo

```
//Definindo o tipo enum
```

```
enum situacao {otima, boa=9, regular, ruim, pessima};
```

```
//Declarando a variável que utiliza o enum já definido
```

```
enum situacao s_dinheiro;
```

```
void main() {
```

```
    //Atribuindo um valor a variável:
```

```
    s_dinheiro = otima;
```

```
    printf("%d", s_dinheiro); //Imprime 0
```

```
    printf("%d", otima);      //Imprime 0
```

```
    printf("%d", boa);        //Imprime 9
```

```
    printf("%d", pessima);    //Imprime 12
```

```
}
```

enum

- Outro exemplo:

```
enum moeda {real, dolar, nickel=100, libra, euro};
```

real	0
dolar	1
nickel	100
libra	101
euro	102

enum - Problema

- Até agora, vimos que um enum representa um conjunto de símbolos. Cada símbolo representa um número inteiro.

```
enum situacao {otima, boa, regular, ruim, pessima};  
enum situacao s_dinheiro;  
void main() {  
    s_dinheiro = otima;  
    printf("%d", s_dinheiro);  
}
```

Problema: E se eu quiser imprimir o nome do símbolo?

enum - Problema

- Até agora, vimos que um enum representa um conjunto de símbolos. Cada símbolo representa um número inteiro.
- **Problema:** E se eu quiser imprimir o **nome do símbolo**?

```
enum situacao {otima, boa, regular, ruim, pessima};  
enum situacao s_dinheiro;  
void main() {  
    s_dinheiro = otima;  
    printf("%s", s_dinheiro); // Funciona??  
}
```

enum - Problema

- Até agora, vimos que um enum representa um conjunto de símbolos. Cada símbolo representa um número inteiro.

- **Problema**

Não!

```
enum si...na};  
enum si...  
void main() {  
    s_dinheiro = otima;  
    printf("%s", s_dinheiro); // Funciona??  
}
```

enum - Problema

- Uma opção:

```
enum situacao {otima, boa, regular, ruim, pessima};
enum situacao s_dinheiro;
void main() {
    s_dinheiro = otima;
    switch(s_dinheiro) {
        case otima:
            printf("otima");
            break;
        case boa:
            printf("boa");
            break;
        ...
    }
}
```

Exemplo em sala

- Fazer um programa que utiliza um *enum* para descrever os meses do ano

Ponteiros para Funções

Ponteiros para funções

- A ideia dos ponteiros pode ser estendida para apontar para funções
- Assim, um ponteiro de função é uma variável que armazena o **endereço de uma função**
- Esta função poderá ser chamada a qualquer momento a partir do ponteiro
- A vantagem de se utilizar ponteiros para funções é ter um código mais genérico, podendo ser utilizado em diversos lugares

Ponteiros para funções

- Sintaxe:

```
tipo_retorno (* nome_ponteiro) (tipos param.)
```


Ponteiros para funções

- Sintaxe:

tipo_retorno (* *nome_ponteiro*) (*tipos param.*)



Não se esqueça dos parênteses!

Ponteiros para funções

- Exemplos de ponteiros para funções:

```
void (*ptr) (int, float);
```

```
int (*ptr2) ();
```

```
char* (*ptr_func) (char*, int*);
```

```
float (*ptr_func2) (float*, float*);
```

Qual é o endereço de uma função?

- O endereço de uma função é dado pelo seu nome somente (*sem os parênteses*)
- Assim, para descobrir o endereço da função *imprime* do código a seguir, basta:

```
void imprime(int i) {  
    printf("%d\n", i);  
}  
  
int main() {  
    printf("Endereço de imprime é: %p", imprime);  
}
```

Atribuindo um valor em um ponteiro de função

```
int fazAlgo(int n){  
    //faz alguma coisa  
}  
  
int main(){  
    //Criar o ponteiro para função  
    //que retorna int e tenha um int como parâmetro  
    int (*ptr)(int);  
  
}
```

Atribuindo um valor em um ponteiro de função

```
int fazAlgo(int n){  
    //faz alguma coisa  
}  
  
int main(){  
    //Criar o ponteiro para função  
    //que retorna int e tenha 1 int como parâmetro  
    int (*ptr)(int);  
    //atribui ponteiro ptr para função fazAlgo  
    ptr = fazAlgo;  
}
```

Atribuindo um valor em um ponteiro de função

```
int fazAlgo(int n){
    //faz alguma coisa
}

int main(){
    //Criar o ponteiro para função
    //que retorna int e tenha 1 int como parâmetro
    int (*ptr)(int);
    //atribui ponteiro ptr para função fazAlgo
    ptr = fazAlgo;
    //chama a função pelo ponteiro
    int i = ptr(9);
}
```



Operador Ternário



Operador Ternário

- **Estrutura de Condição** simplificada!
- Mesma ideia do *if / else*
- Verifica a condição e pode retornar um valor em **apenas uma linha!**
- Sintaxe:

condição ? verdadeiro : falso

Operador Ternário

- Exemplo com **if / else**: Verificar se o valor de **soma** é maior ou menor que 10

```
if(soma > 10)
{
    printf("O valor da soma é maior que 10\n");
}
else
{
    printf("Valor menor ou igual a 10");
}
```

Operador Ternário

- Exemplo com **operador ternário**: Verificar se o valor de **soma** é maior ou menor que 10

```
✓ int main(){  
    int soma = 8;  
    soma > 10 ? puts("O valor da soma eh maior que 10\n") :  
               puts("Valor menor ou igual a 10") ;  
}
```

Operador Ternário

- Exemplo com **if / else**: Cálculo do aumento de salário: 10% se ganha até R\$ 2000.00; 5% se ganha R\$ 2000.00 ou mais.

```
#include <stdio.h>

int main(){
    float salario = 2500;
    float aumento;
    if(salario >= 2000)
        aumento = salario * 0.05;
    else
        aumento = salario * 0.10;
}
```

Operador Ternário

- Exemplo com **operador ternário**: Cálculo do aumento de salário: 10% se ganha até R\$ 2000.00; 5% se ganha R\$ 2000.00 ou mais.

```
#include <stdio.h>

int main(){
    float salario = 2500;
    float aumento = salario >= 2000 ? salario*0.05 : salario*0.10;
    printf("%.2f", aumento);
}
```



Exercícios



Exercícios



1. Escreva um programa em C para encontrar o máximo entre dois números usando ***operador ternário***.
2. Faça uma calculadora com 2 operações básicas, soma e subtração, utilizando ***operador ternário***.

Exercícios



3. Escreva um programa em C que leia as medidas dos lados de um triângulo e escreva se ele é **equilátero**, **isósceles** ou **escaleno**. Utilize *enum* para os tipos dos triângulos.
 - Triângulo **equilátero**: 3 lados iguais.
 - Triângulo **isósceles**: 2 lados iguais.
 - Triângulo **escaleno**: 3 lados diferentes.

Exercícios



4. Faça um programa em C com uma função que recebe o dia, o mês (como **enum**) e o ano e retorna o próximo dia.
5. Faça um programa que tenha uma **struct** aluno com nome, número de matrícula e curso. Receba do usuário a informação de 5 alunos, armazene em um vetor dessa estrutura e imprima os dados na tela. Utilize **typedef** para fazer a **struct**.

Exercícios



6. Crie um procedimento chamado *iterate* que recebe um **vetor de int**, o **número de elementos** e um **ponteiro para um procedimento** que somará 1 em cada um dos números do vetor. Exemplo:

```
... add1(...) {  
    .....  
}  
  
int main() {  
    int i[]={1,2,3,4,5};  
    iterate(i,5, add1);  
  
    //No final, o vetor i deverá ser {2,3,4,5,6}  
}
```

Sintaxe - vetores de ponteiros de funções: *tipo_retorno* (* *nome_ponteiro*[**x**]) (*tipos param.*)