

Algoritmos - AULA 2

Sumário

1	Protótipos	1
1.1	Exemplo.....	2
2	Escopo de Variáveis.....	2
2.1	Variáveis locais.....	2
2.1.1	Exemplo	2
2.2	Parâmetros formais	3
2.3	Variáveis globais.....	3
2.3.1	Exemplo	3
3	Arquivos-Cabeçalhos.....	3
4	Exercícios.....	4

1 Protótipos

Até agora, nos exemplos apresentados, escrevemos as funções antes de escrevermos a função `main()`. Isto é, as funções estão fisicamente antes da função `main()`. Isto foi feito por uma razão. Imagine-se na pele do compilador. Se você fosse compilar a função `main()`, onde são chamadas as funções, você teria que saber com antecedência quais são os tipos de retorno e quais são os parâmetros das funções para que você pudesse gerar o código corretamente. Foi por isto as funções foram colocadas antes da função `main()`: quando o compilador chegasse à função `main()` ele já teria compilado as funções e já saberia seus formatos.

Mas, muitas vezes, não poderemos nos dar ao luxo de escrever nesta ordem. Muitas vezes teremos o nosso programa espalhado por vários arquivos. Ou seja, estaremos chamando funções em um arquivo que serão compiladas em outro arquivo. Como manter a coerência?

A solução são os protótipos de funções. Protótipos são nada mais, nada menos, que declarações de funções. Isto é, você declara uma função que irá usar. O compilador toma então conhecimento do formato daquela função antes de compilá-la. O código correto será então gerado.

Sintaxe Básica:

tipo nome (tipo parâmetro1, tipo parâmetro2,...);

Onde:

- tipo: determina o tipo do valor de retorno;
- nome: é o nome da função;
- tipo parâmetro: são variáveis, que recebem os valores passados quando chamamos a função.

1.1 Exemplo

```
#include <stdlib.h>
#include <stdio.h>
int calcula(int a, int b);
int main ( ) {
    int soma,a,b;
    printf("entre com dois numeros\n");
    scanf("%d%d",&a,&b);
    printf("Soma = %d\n",calcula(a,b));
    system("pause");
    return 0;
}
int calcula(int a, int b){
    return a + b;
}
```

Atenção: observe que a função `calcula()` está colocada depois da função `main()`, mas o seu protótipo está antes. Sem isto este programa não funcionaria corretamente.

2 Escopo de Variáveis

O escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa. O escopo de uma variável determina de quais partes do código a variável pode ser acessada.

2.1 Variáveis locais

O primeiro tipo de variáveis que veremos são as variáveis locais. São aquelas que só têm validade dentro do bloco na qual são declaradas. Sim. Podemos declarar variáveis dentro de qualquer bloco. Só para lembrar: um bloco começa quando abrimos uma chave e termina quando fechamos a chave.

Até agora só tínhamos visto variáveis locais para funções completas.

2.1.1 Exemplo

```
#include <stdlib.h>
#include <stdio.h>
void exemplo(int valor){
    valor *= 2;
    printf("valor na funcao exemplo = %d\n",valor);
}
int main ( ) {
    int valor = 3;
    printf("valor na funcao main = %d\n",valor);
    exemplo(valor);
    printf("valor na funcao main = %d\n",valor);
    system("pause");
    return 0;
}
```

```
valor na funcao main = 3
valor na funcao exemplo = 6
valor na funcao main = 3
Pressione qualquer tecla para continuar. . . _
```

2.2 Parâmetros formais

O segundo tipo de variável que veremos são os parâmetros formais. Estes são declarados como sendo as entradas de uma função. Não há motivo para se preocupar com o escopo deles.

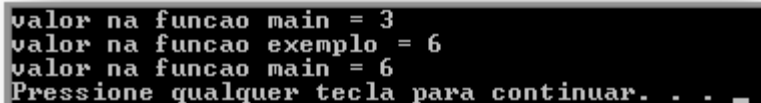
É fácil: o parâmetro formal é uma variável local da função. Você pode também alterar o valor de um parâmetro formal, pois esta alteração não terá efeito na variável que foi passada à função. Isto tem sentido, pois quando o C passa parâmetros para uma função, são passadas apenas cópias das variáveis. Isto é, os parâmetros formais existem independentemente das variáveis que foram passadas para a função. Eles tomam apenas uma cópia dos valores passados para a função.

2.3 Variáveis globais

Variáveis globais são declaradas fora de todas as funções do programa. Elas são conhecidas e podem ser alteradas por todas as funções do programa. Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.

2.3.1 Exemplo

```
#include <stdlib.h>
#include <stdio.h>
int valor = 3;
void exemplo() {
    valor *= 2;
    printf("valor na funcao exemplo = %d\n", valor);
}
int main() {
    printf("valor na funcao main = %d\n", valor);
    exemplo();
    printf("valor na funcao main = %d\n", valor);
    system("pause");
    return 0;
}
```



```
valor na funcao main = 3
valor na funcao exemplo = 6
valor na funcao main = 6
Pressione qualquer tecla para continuar. . . _
```

3 Arquivos-Cabeçalhos

Arquivos-cabeçalhos são aqueles que temos mandado o compilador incluir no início de nossos exemplos e que sempre terminam em .h. A extensão .h vem de header (cabeçalho em inglês).

Já vimos exemplos como stdio.h, conio.h, string.h. Estes arquivos, na verdade, não possuem os códigos completos das funções. Eles só contêm protótipos de funções. É o que basta. O compilador lê estes protótipos e, baseado nas informações lá contidas, gera o código correto.

O corpo das funções cujos protótipos estão no arquivo-cabeçalho, no caso das funções do próprio C, já estão compiladas e normalmente são incluídas no programa no instante da "linkagem". Este é o instante em que todas as referências a funções cujos códigos não estão nos nossos arquivos fontes são resolvidas, buscando este código nos arquivos de bibliotecas.

Se você criar algumas funções que queira aproveitar em vários programas futuros, ou módulos de programas, você pode escrever arquivos-cabeçalhos e incluí-los também.

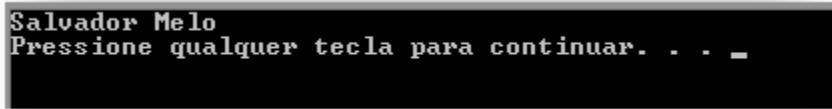
3.1 Exemplo

Arquivo: teste.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "exemplo.h"
int main ( ) {
    mostra();
    system("pause");
    return 0;
}
```

Arquivo: exemplo.h

```
#include <stdlib.h>
#include <stdio.h>
void mostra( ){
    printf("Salvador Melo\n");
}
```



```
Salvador Melo
Pressione qualquer tecla para continuar. . . _
```

3.2 Uso das aspas

Usualmente, os arquivos-cabeçalho são salvo na pasta padrão, geralmente chamada de pasta include. Neste caso você deve usar o sinal de menor e maior para indicar o nome do arquivo-cabeçalho.

Se o arquivo-cabeçalho estiver na mesma pasta que o programa que o chama, você deve usar as aspas duplas com a diretiva include.

4 Exercícios com variáveis globais

- Desenvolva um programa que lê três notas de um aluno e as guarda em uma variável global, depois chama uma segunda função que calcula e mostra a média aritmética dessas notas.
- Desenvolva um programa que lê três notas de um aluno e as guarda em variáveis globais. A função main então deve chamar uma segunda função que calcula e mostra a média aritmética dessas notas e guarda numa variável global. Depois crie uma terceira função que informa se o aluno foi aprovado ou reprovado (nota mínima para aprovação: 6,0).
- Crie um programa que lê um valor de ângulo e pergunta se o mesmo está em graus ou radianos. Se estiver em graus chama uma função que converte para radianos e mostra o resultado. Se estiver em radianos, chama outra função que converte para graus e mostra o resultado.
- Faça um programa que lê dois valores diferentes, guarda esses valores em variáveis globais e chama uma função para saber qual dos dois é o maior.

5 Exercícios com arquivos-cabeçalho

- a. Crie um arquivo-cabeçalho com duas funções: uma que calcula a parte inteira do quociente quando o inteiro a é dividido pelo inteiro b ; outra que calcula o resto da divisão inteira de a por b . Construa um programa que inclui este arquivo – cabeçalho e chame as duas funções.
- b. Crie um arquivo-cabeçalho com uma função chamada `calculaPotencia`, que recebe o valor da base e do expoente, e retorna o valor da potência. Por exemplo, `calculaPotencia(2,3) = 2*2*2`. Construa um programa que inclui este arquivo – cabeçalho, que lê o valor da base e do expoente, chame a sua função do arquivo-cabeçalho e mostre o valor da potência. Atenção: não pode usar a função `pow()`.
- c. Crie um arquivo-cabeçalho com uma função chamada `calculaPrimo` que recebe um número e verifica se o mesmo é primo ou não. Se for primo, deve retornar 1, senão deve retornar 0. Construa um programa que inclui este arquivo – cabeçalho, lê um número inteiro do teclado e verifica se o mesmo é primo ou não, usando a função que você criou no arquivo cabeçalho. Atenção: um número é primo se for divisível apenas por 1 e por ele mesmo.