Algoritmos e Programação de Computadores

Funções

Prof. Lucas Boaventura
lucas.boaventura@unb.br

Introdução

- Ao longo do nosso curso, mantivemos a maior parte dos nossos códigos na função “main”
- A medida que o código se torna mais complexo, é necessário dividir em pequenas partes
- As funções podem ser utilizadas para isso!

Introdução

- Uma função deve ser planejada para manter um pedaço de lógica do seu código
- Isso ajuda a estruturar o código e deixá-lo mais legível
- Além disso, permite que trechos de código sejam reutilizados

Introdução

- Ja estudamos partes da biblioteca padrão C, que possui diversas funções para nos ajudar:
  - scanf, printf, para entrada/saída de dados
  - strlen, strcpy, para manipulação de strings
- Agora, vamos aprender a criar as nossas próprias funções!

Função

- Uma função é composta por um nome único, assim como nome de variáveis, não pode começar por números, nem podem existir 2 funções com o mesmo nome
- Uma função também recebe parâmetros como argumentos
- Além disso, uma função tem um tipo de retorno

Função

- Exemplo de função

```
tipo_retornado nome(parametros)
{
    comandos da funcao
}
```

```c
#include <stdio.h>

int imprimir_mensagem()
{
    printf("Mensagem dentro da funcao\n");
    return 0;
}

int main()
{
    printf("Antes da funcao!\n");
    imprimir_mensagem();
    printf("Saida da funcao!\n");
    return 0;
}
```

Saída do código:

Antes da funcao!

Mensagem dentro da funcao

Saida da funcao!

```c
#include <stdio.h>

int quadrado(int n)
{
    return n * n;
}

int main()
{
    int n, m;
    scanf("%d", &n);
    m = quadrado(n);
    printf("O quadrado de %d e' %d\n", n, m);
    return 0;
}
```

Parâmetros

- Uma função pode receber nenhum ou vários parâmetros como argumento
- Neste exemplo, o retorno “0” é ignorado (não é atribuído o valor a uma variável)

```c
#include <stdio.h>

int imprime_mensagem()
{
    printf("Ola, mundo!\n");
    return 0;
}

int main()
{
    imprime_mensagem();
    return 0;
}
```

Função

- Exemplo de função com múltiplos parâmetros
- Uma função pode ser usada dentro de um printf

```c
#include <stdio.h>

int multiplica(int x, int y)
{
    return x * y;
}

int main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d\n", multiplica(x, y));
    return 0;
}
```

Protótipos

- O protótipo de uma função identifica o nome e parâmetros que ela recebe
- Ele é muito importante quando você for dividir seu código fonte em diversos arquivos (tópico muito avançado para APC)
- Mas pode ser necessário declarar o protótipo de uma função se ela for declarada depois da main

Protótipos

- Código possui um aviso “implicit declaration of function multiplica”

```c
#include <stdio.h>

int main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d\n", multiplica(x, y));
    return 0;
}

int multiplica(int x, int y)
{
    return x * y;
}
```

- Aviso removido!

```c
#include <stdio.h>

int multiplica(int x, int y);

int main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d\n", multiplica(x, y));
    return 0;
}

int multiplica(int x, int y)
{
    return x * y;
}
```

Protótipos

- O nome das variáveis que existem na função main e na função multiplica são iguais por mera coincidência!
- Na verdade, eles são completamente independentes e, normalmente, são diferentes

Protótipos

```c
#include <stdio.h>

int multiplica(int a, int b)
{
    return a * b;
}

int main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d\n", multiplica(x, y));
    return 0;
}
```

Corpo

- Uma função ele não precisa conter apenas uma instrução de return
- Comumente, possui mais elementos do que uma instrução
- Dentro de uma função é possível declarar variáveis que serão usadas dentro dela

Este return irá interromper o laço “for”

```c
char primeira_maiuscula(char str[])
{
    for (int i = 0; str[i] != '\0'; i++)
        if (str[i] >= 'A' && str[i] <= 'Z')
            return str[i];
    return '\0';
}

int main()
{
    char str[250];
    char letra;
    scanf("%s", str);
    letra = primeira_maiuscula(str);
    printf("%c\n", letra);
    return 0;
}
```

Procedimento

- Um procedimento é uma função que não retorna parâmetro
- Na linguagem C, pode-se utilizar a palavra chave “void” para indicar que não existe retorno

Procedimento

```c
void imprime_mensagem()
{
    printf("Por favor, digite um numero:\n");
}

int main()
{
    int n;
    imprime_mensagem();
    scanf("%d", &n);
    return 0;
}
```

Procedimento

- Também pode-se utilizar “void” para indicar que uma função não recebe parâmetros
- No entanto, isto está caindo em desuso...

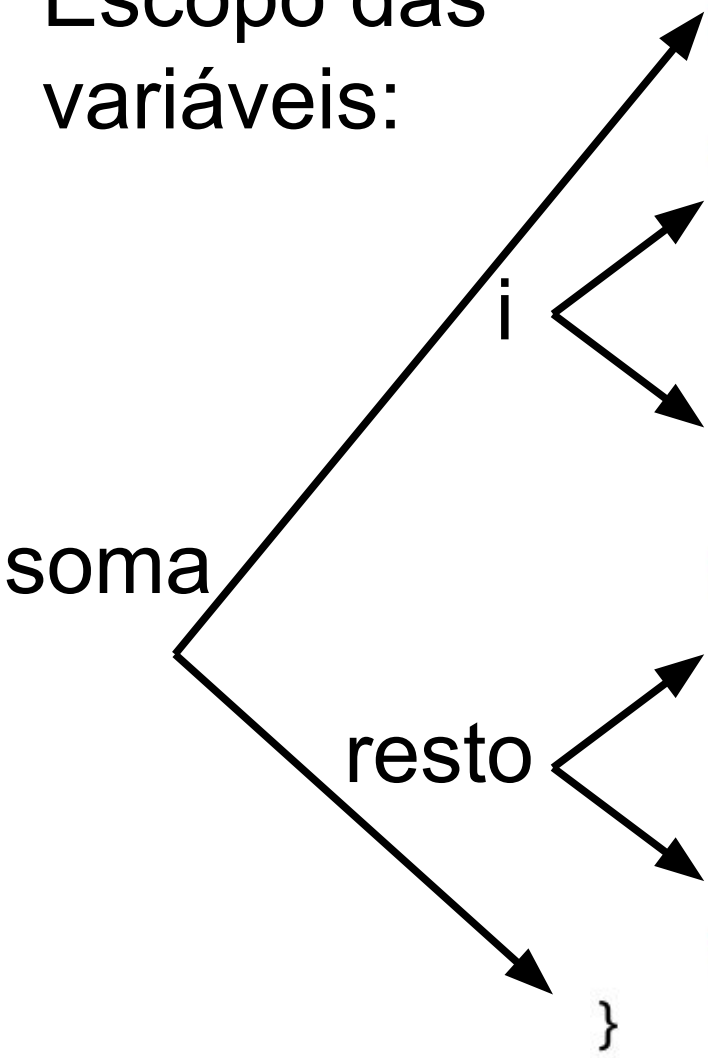
```
void imprime_mensagem(void)
{
    printf("Por favor, digite um numero:\n");
}
```

Escopo

- As variáveis declaradas dentro de uma função são conhecidas como variáveis locais
- O escopo de uma variável é onde essa variável pode ser acessada!
- Variáveis declaradas dentro de um laço for, possuem também um escopo limitado

Escopo Local

Escopo das variáveis:




```c
int func()
{
    int soma = 0;

    for (int i = 0; i < 10; i++)
    {
        soma += i;
    }

    if (soma % 2)
    {
        int resto = soma % 10;
        printf("%d\n", resto);
    }
    return 0;
}
```

Escopo Global

- Também é possível declarar variáveis globais, isto é fora de qualquer função
- Neste caso, todas as funções irão manipular a mesma variável quando usarem o nome

```c
#include <stdio.h>

int cont = 0;
int func()
{
    printf("Valor de cont no comeco da funcao: %d\n", cont);
    cont = cont + 1;
    printf("Valor de cont no final da funcao: %d\n", cont);
    return 0;
}

int main()
{
    printf("Valor de cont no comeco do progama: %d\n", cont);
    cont = cont + 1;
    func();
    cont = cont + 1;
    printf("Valor de cont no final do progama: %d\n", cont);
    return 0;
}
```

Escopo Global

- O programa imprime:
- Valor de cont no comeco do progama: 0
- Valor de cont no comeco da funcao: 1
- Valor de cont no final da funcao: 2
- Valor de cont no final do progama: 3

Passagem por Valor

- Na etapa inicial do curso, utilizaremos a passagem por valor
- Isso quer dizer que os parâmetros da função são uma cópia da variável original
- Nesse caso, modificações nas variáveis de parâmetro não modificam a variável original

Passagem por Valor

```c
#include <stdio.h>

int func(int x)
{
    printf("Valor de x no comeco da funcao: %d\n", x);
    x = x + 1;
    printf("Valor de x no final da funcao: %d\n", x);
    return 0;
}

int main()
{
    int x = 10;
    printf("Valor de x no comeco do progama: %d\n", x);
    func(x);
    printf("Valor de x no final do progama: %d\n", x);
    return 0;
}
```

Passagem por Valor

- O programa irá imprimir:
- Valor de x no comeco do progama: 10
- Valor de x no comeco da funcao: 10
- Valor de x no final da funcao: 11
- Valor de x no final do progama: 10

Passagem por Valor

- Faça um programa que leia um número N, crie um vetor de N posições, LÊ N inteiros do teclado e, em seguida, chama uma função, conforme o protótipo abaixo, que retorna o maior número dentro do vetor.

- 3 <= N <= 100
- int acha_maior(int vetor[], int tamanho_vetor)

Dúvidas?

- lucas.boaventura@unb.br