Laboratório de Algoritmos e Técnicas de Programação II

Aula 03 - Modularização: Declaração, escopo de variáveis e passagem de parâmetros

Álvaro Magri Nogueira da Cruz

Agenda

Introdução I

Dividir para conquistar

- Melhor maneira de manter um programa grande → **MÓDULOS**;
- **Módulo:** partes pequenas de um programa;
- Essa técnica também é chamada de **dividir para conquistar.**

Módulos em C

- Módulos em C = **funções**;
- Programa em C = funções "pré-definidas" (**biblioteca-padrão**) + **novas funções**.

- **Bibliotecas-padrão:** *string.h*, *stdio.h*, *stdlib.h*, etc.

Introdução II

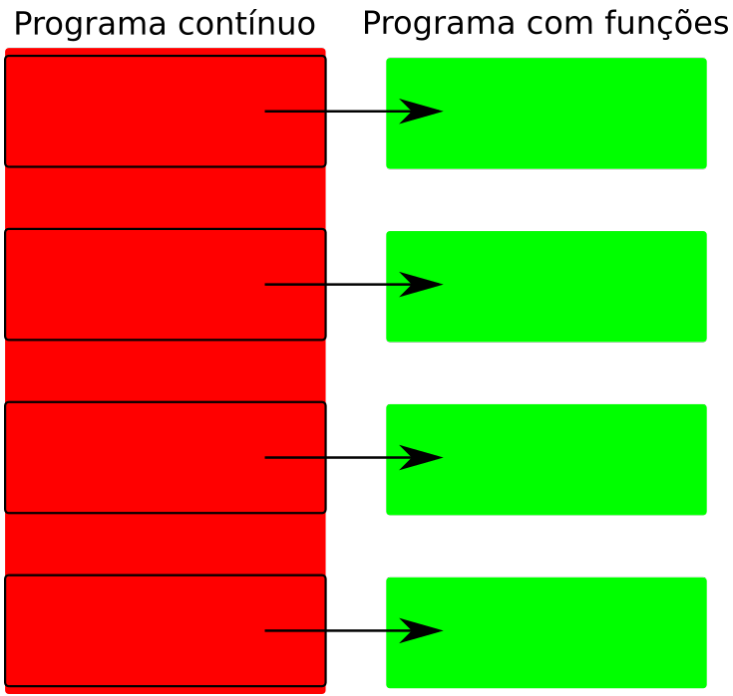


Figura 1: Programa contínuo versus com uso de funções.

Módulos de programa em C I

Ideia central

- Funções definidas pelo programador são **escritas** apenas **UMA VEZ**;
- Para ser utilizada ela é **INVOCADA** = **chamada de função**;
  - Especifica o nome da função;
  - Pode ou não oferecer **argumentos** (informações) para realizar uma tarefa.

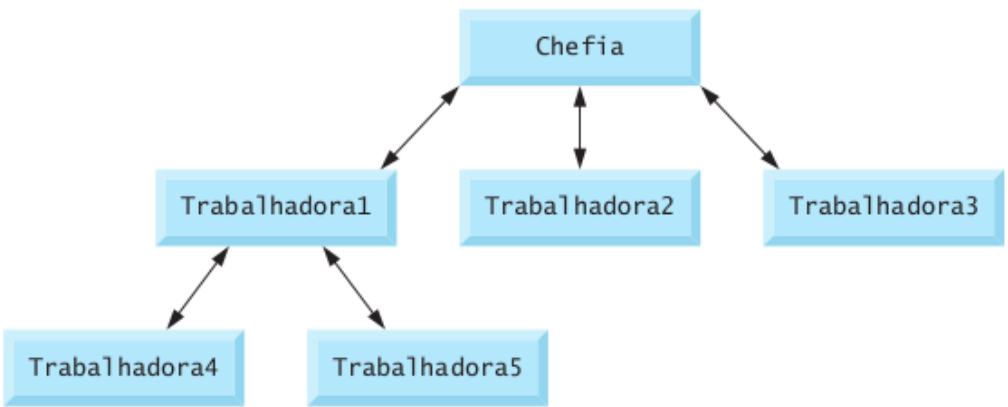


Figura 2: Relacionamento hierárquico entre função chefia e função trabalhadora.

Módulos de programa em C II

Utilizando uma função em C

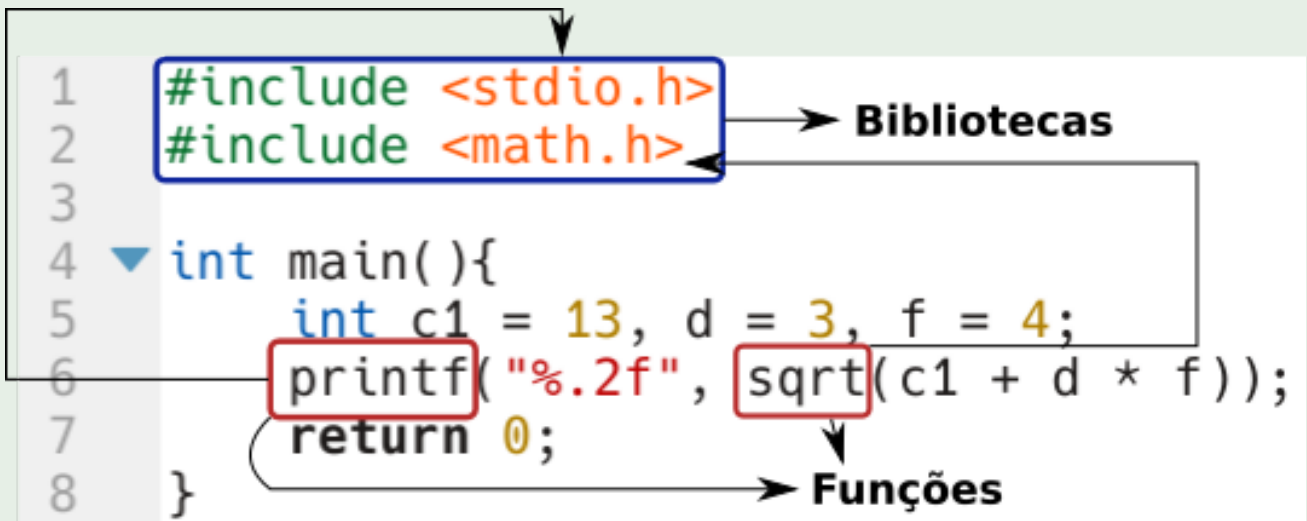


Figura 3: Utilizando funções de bibliotecas-padrão.

Funções

Características de uma função

- Permite a **modularização** de um programa;
- Todas as variáveis declaradas dentro de uma função **são locais**;
- Permite **passagem de parâmetros**, i.e, um meio e comunicação do meio externo para a função;
- Permite a **reutilização do software** - Engenharia de Software.

Definindo funções I

Como defino uma função personalizada?

1. Declaração do protótipo da função (Opcional);
2. Definição de função.

- **OBS1:** Usar protótipos nos dá mais flexibilidade com o código fonte e melhor organização;
- **OBS2:** Não é necessário declarar um protótipo da função, pode-se definir a função imediatamente;
- **OBS3:** Quando optamos pelo protótipo a definição vem após a estrutura principal (*main*) do programa.

Definindo funções II

```c
/*Criando e usando uma função definida pelo programador */
#include <stdio.h>

int square(int y); /* protótipo da função */
/* função main inicia execução do programa */
int main(void){
    int x; /* contador */
    /* loop 10 vezes e calcula e exibe quadrado de x a cada vez
    */
    for (x = 1; x <= 10; x++){
            printf("%d \t", square(x)); /* chamada da função */
    }/* fim do for */
    printf("\n");
    return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
/* definição de função square retorna quadrado do parâmetro */
int square(int y){ /* y é uma cópia do argumento à função */
    return y * y; /* retorna o quadrado de y como um int */
} /* fim da função square */
```

```
1       4       9       16      25      36      49      64      81      100
```

Figura 4: Função que calcula o quadrado de um número.

Definindo funções III

Protótipo da função

- Linha 4 da Figura 5;
  - **int** *square* (**int** y);
- O **int** nos parênteses informa ao compilador que *square* espera um valor inteiro da chamadora;
- O **int** à esquerda de *square* informa ao compilador que o retorno desta função é um inteiro.

Definindo funções IV

Definição da função

- Linha 16 a 18 da Figura 5;
  - **int** *square* (**int** y){
    return y*y;
    }
- O formato de uma definição é:

tipo-valor-retorno *nome-função*(lista de parâmetros){
definições (declarações de variáveis)
instruções
}

- *nome-função*, **tipo-valor-retorno** e lista de parâmetros, juntos, são comumente chamados de **cabeçalho** da função.

Definindo funções V

Exemplo de aplicação de função

- O usuário deve inserir três valores inteiros como entrada;
- O programa deve chamar uma função para selecionar o maior entre os três;
- O resultado deve ser exibido ao usuário.

Definindo funções VI

```c
#include <stdio.h>
int maximum(int x, int y, int z); /* protótipo de função */

int main(void){
    int number1; /* primeiro inteiro */
    int number2; /* segundo inteiro */
    int number3; /* terceiro inteiro */
    printf("Digite três inteiros: ");
    scanf("%d", &number1);
    scanf("%d", &number2);
    scanf("%d", &number3);
    /* number1, number2 e number3 são argumentos
    da chamada da função maximum */
    printf( "Máximo é: %d\n", maximum(number1, number2, number3));
    return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
/* Definição da função maximum */
/* x, y e z são parâmetros */
int maximum(int x, int y, int z){
    int max = x; /* considera que x é o maior */
    if ( y > max ){ /* se y é maior que max, atribui y a max */
            max = y;
    } /* fim do if */
    if ( z > max ){ /* se z é maior que max, atribui z a max */
            max = z;
    } /* fim do if */
    return max; /* max é o maior valor */
} /* fim da função maximum */
```

Definindo funções VII

Códigos : bash — Konsole

Arquivo Editar Exibir Favoritos Configurações Ajuda

```
bruna@bruna:~/Documentos/Documentos/UNESP/Lab. Prog. II/Aulas/Aula 03/Códigos$ ./maximum
Digite três inteiros: 20
50
100
Máximo é: 100
bruna@bruna:~/Documentos/Documentos/UNESP/Lab. Prog. II/Aulas/Aula 03/Códigos$ ./maximum
Digite três inteiros: 50
20
100
Máximo é: 100
bruna@bruna:~/Documentos/Documentos/UNESP/Lab. Prog. II/Aulas/Aula 03/Códigos$ ./maximum
Digite três inteiros: 100
20
50
Máximo é: 100
bruna@bruna:~/Documentos/Documentos/UNESP/Lab. Prog. II/Aulas/Aula 03/Códigos$ ./maximum
Digite três inteiros: 50
100
20
Máximo é: 100
bruna@bruna:~/Documentos/Documentos/UNESP/Lab. Prog. II/Aulas/Aula 03/Códigos$
```

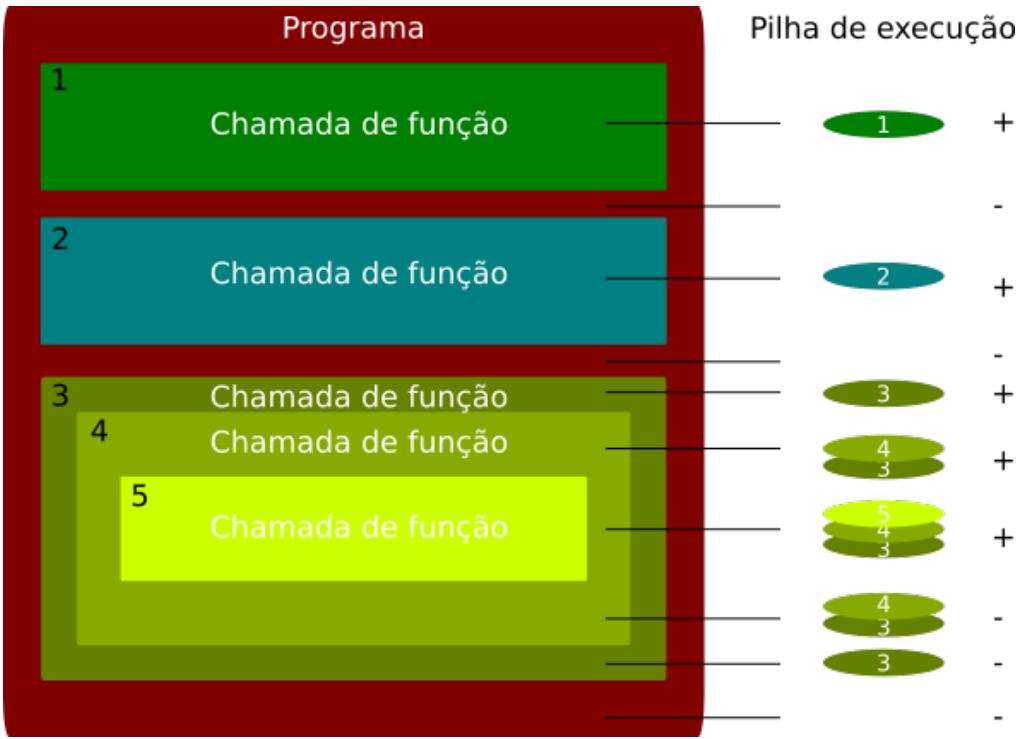
Pilha de chamada de funções I

Definição

- Pilha: estrutura de dados (**LIFO**) na qual o último item empilhado na pilha é o primeiro a ser desempilhado.

- Passo a passo do uso da pilha de execução do programa (**pilha de chamada de função**):
  1. Programa **chama** uma função;
  2. Empilha endereço de retorno na pilha;
  3. Se houver uma śerie de chamadas de função:
     - Os endereços de retorno são empilhados na ordem LIFO.
  4. A cada término um endereço é desempilhado.
- **ATENÇÃO**: Memória limitada, pode ocorrer estouro de pilha (**stack overflow**)!

Pilha de chamada de funções II



Chamando funções I

- Existem duas maneiras de chamar funções:
  - Chamada por **valor**;
  - Chamada por **referência**.

Chamando funções II

Chamada por valor

- Argumentos passados por valor → cópia do valor é passada para a função;
  - As mudanças feitas na cópia **não afetam** o valor original da variável chamadora.
- Evita **efeitos colaterais**!

...

int num1, num2, max;

num1 = 10;

num2 = 20;

max = maximum (num1, num2);

...

Chamando funções III

Chamada por referência

- Em C, **todas as chamadas** são feitas por valor;
- No entanto, tem como **simular** a chamada por referência:
  - Operadores de endereço (&);
  - Operadores de indireção (*, ponteiros).
- É passada para a função uma referência da variável, toda alteração na referência altera o conteúdo da variável original.

Chamando funções IV

Exemplo de chamada por referência

```c
#include <stdio.h>
//função que soma 10 ao valor recebido
void soma10p(int *x){
  *x = *x + 10;
  printf("Valor de x apos a soma = %d \n",*x);
}

int main(void){
  int numero;
  printf("Digite um numero: ");
  scanf("%d", &numero);
  printf("O numero digitado foi: %d \n",numero);
  soma10p(&numero);  //chamada da função com ponteiro como parâmetro
  printf("Agora o numero vale: %d \n",numero);
  return 0;
}
```

Chamando funções V

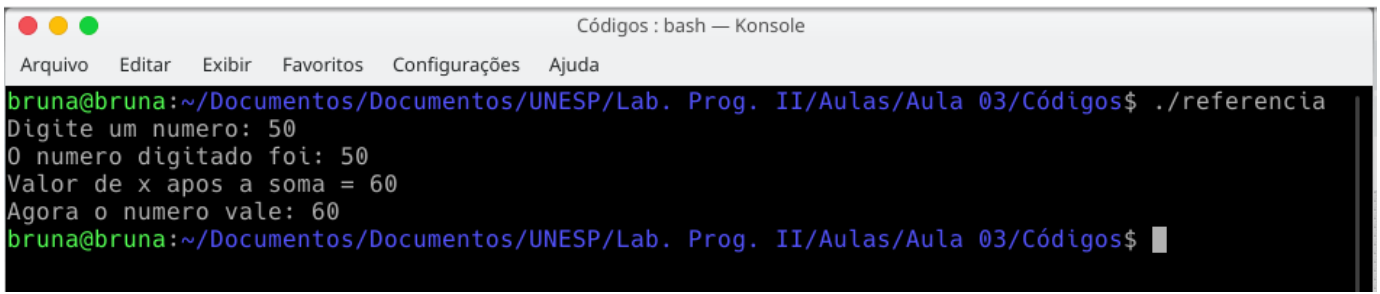


Figura 5: Execução do exemplo de chamada por referência com o número 50.

Exercícios

1. Escreva um programa no qual este deva receber 10 números, descobrir qual o maior, menor e média. Escreva uma função para cada uma dessas operações;
2. Escreva um programa que gere um vetor de 100 posições com números aleatórios três vezes. A cada vez que este vetor for atualizado imprima seus valores na tela. Para isso é interessante criar uma função para imprimir o vetor.

Referências I

1. **Deitel H. M., e Deitel P.J; "C: Como programar". 6.ed. Pearson Prentice Hall, 2011. 818p.**
2. Jean Paul Tremblay & Richard P. Bunt. "Ciência dos Computadores - Uma abordagem algorítmica". McGraw-Hill.
3. Jaime Evaristo. "Aprendendo a Programar / Programando em Turbo Pascal". Edufal - Editora da Univ. Federal de Alagoas. Maceió, 1996.
4. Harry Farrer et al. "Pascal Estruturado (da série "Programação Estruturada de Computadores")". Editora Guanabara Dois. Rio de Janeiro, 1985.
5. Stephen O'Brien. "Turbo Pascal 6 Completo e Total". Makron Books.

Referências II

6. Celes, W., Cerqueira, R., Rangel, J.L. "Introdução a Estrutura de Dados". Elsevier, 2004.
7. Feofiloff, P. "Algoritmos em Linguagem C". Elsevier, 2009. 208p.
8. Schildt, H. "C Completo e Total". 3ª ed. Pearson. 1996. 852p.