

Laboratório de programação

Funções e Módulos

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

Funções

Definição

Passagem de Parâmetros por Valor e por Referência

Módulos de Programas em C

Arquivos de Cabeçalho

Compilação de módulos

- Em linguagem C, a declaração de uma função pelo programador segue esta forma geral:

```
tipo_retornado nome_da_funcao(lista_de_parametros) {  
    sequência de declarações e comandos  
}
```

- **tipo_retornado** pode ser um tipo básico como **int**, **char**, etc. Ou na ausência de retorno **void**.
- **lista_de_parametros** pode ser vazia ou uma sequência de uma ou mais declarações de variáveis separadas por vírgulas, por exemplo:

```
int soma(int a, int b) ...
```
- Uma função, ela deve ser definida ou declarada antes de ser utilizada.

- Considere um programa que use a função **square** para calcular os quadrados dos números inteiros de 1 a 10.

```
1  #include <stdio.h>
2
3  /* Definição da funcao */
4  int square(int y){
5      return y * y;
6  }
7
8  int main(){
9      int x;
10     for (x = 1; x <= 10; x++)
11         printf("%d ", square(x));
12     return 0;
13 }
```

- Uma função definida pelo programador denominada **maiorDeTres** para determinar e retornar o maior entre três inteiros.

```
1  #include <stdio.h>
2
3  int maiorDeTres(int a, int b, int c) {
4      if (a >= b && a >= c) return a;
5      if (b >= a && b >= c) return b;
6      return c;
7  }
8
9  int main(){
10     int a, b, c;
11     printf("Entre com tres inteiros: ");
12     scanf("%d%d%d", &a, &b, &c);
13     printf("O maior e: %d\n", maiorDeTres(a, b, c));
14     return 0;
15 }
```

- Pode-se também declarar uma função depois da cláusula **main()**. Nesse caso, é preciso declarar antes o *protótipo da função*.
- O **protótipo de uma função** é uma declaração de função que omite o corpo mas especifica o seu nome, tipo de retorno e lista de parâmetros.

```
1  #include <stdio.h>
2
3  int square(int); // Protótipo da função square
4
5  int main(){
6      int x;
7      for (x = 1; x <= 10; x++)
8          printf("%d ", square(x));
9      return 0;
10 }
11 int square(int y){
12     return y * y;
13 }
```

Exercício 9

Faça um programa contendo uma função que receba dois números positivos por parâmetro e retorne a soma dos N números inteiros existentes entre eles.

Exercício 10

Crie uma função que receba como parâmetro dois valores inteiros X e Z , calcule e retorne X^Z .

Exercício 11

Crie uma função que extraia as partes inteira e decimal de um número real. Use a função apenas para extrair os valores e exiba-os apenas na função `main`.

- As duas maneiras de ativar as funções em muitas linguagens de programação são chamadas por **valor** e chamadas por **referência**.

```
1 int parametroPorValor(int x)
```

```
1 int parametroPorReferencia(int *x)
```


Código

```
...  
void funcao_a(int a) {  
    a = 2;  
}  
  
void funcao_b(int *b) {  
    *b = 2;  
}  
  
int main() {  
    int x = 5;  
    funcao_a(x);  
    funcao_b(&x);  
    return 0;  
}
```

Memória

#	Variável	Valor
284		
285	int x	
286		
⋮	⋮	⋮
588	int a	
589		
⋮	⋮	⋮
732	int *b	

Código

```
...  
void funcao_a(int a) {  
    a = 2;  
}  
  
void funcao_b(int *b) {  
    *b = 2;  
}  
  
➡ int main() {  
    int x = 5;  
    funcao_a(x);  
    funcao_b(&x);  
    return 0;  
}
```

Memória

#	Variável	Valor
284		
285	int x	
286		
⋮	⋮	⋮
588	int a	
589		
⋮	⋮	⋮
732	int *b	

Código

```
...
void funcao_a(int a) {
    a = 2;
}

void funcao_b(int *b) {
    *b = 2;
}

int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	
589		
⋮	⋮	⋮
732	int *b	

Código

```
...
void funcao_a(int a) {
    a = 2;
}


void funcao_b(int *b) {
    *b = 2;
}

int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	
589		
⋮	⋮	⋮
732	int *b	

Código



```
...  
void funcao_a(int a) {  
    a = 2;  
}  
  
void funcao_b(int *b) {  
    *b = 2;  
}  
  
int main() {  
    int x = 5;  
    funcao_a(x);  
    funcao_b(&x);  
    return 0;  
}
```

Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	5
589		
⋮	⋮	⋮
732	int *b	

Código

```
...
void funcao_a(int a) {
    a = 2;
}

void funcao_b(int *b) {
    *b = 2;
}


int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	2
589		
⋮	⋮	⋮
732	int *b	

Código

```
...  
void funcao_a(int a) {  
    a = 2;  
}  
  
void funcao_b(int *b) {  
    *b = 2;  
}  
  
int main() {  
    int x = 5;  
    funcao_a(x);  
    funcao_b(&x);  
    return 0;  
}
```



Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	2
589		
⋮	⋮	⋮
732	int *b	

Código

```
...
void funcao_a(int a) {
    a = 2;
}

➔ void funcao_b(int *b) {
    *b = 2;
}

int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	2
589		
⋮	⋮	⋮
732	int *b	#285

Código

```
...
void funcao_a(int a) {
    a = 2;
}

void funcao_b(int *b) {
    *b = 2;
}

int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

O * é o operador de
desreferenciamento

Memória

#	Variável	Valor
284		
285	int x	5
286		
⋮	⋮	⋮
588	int a	2
589		
⋮	⋮	⋮
732	int *b	#285

Código

```
...
void funcao_a(int a) {
    a = 2;
}

void funcao_b(int *b) {
    *b = 2;
}

int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

Memória

#	Variável	Valor
284		
285	int x	2
286		
⋮	⋮	⋮
588	int a	2
589		
⋮	⋮	⋮
732	int *b	#285



Código

```
...
void funcao_a(int a) {
    a = 2;
}

void funcao_b(int *b) {
    *b = 2;
}

int main() {
    int x = 5;
    funcao_a(x);
    funcao_b(&x);
    return 0;
}
```

Memória

#	Variável	Valor
284		
285	int x	2
286		
⋮	⋮	⋮
588	int a	2
589		
⋮	⋮	⋮
732	int *b	#285

```
1  #include <stdio.h>
2
3  int next(int a){
4      return a + 1;
5  }
6
7  int main(){
8      int x0, x1;
9
10     x0 = 7;
11     printf("x0 = %d\n", x0);
12
13     x1 = next(x0);
14     printf("x1 = %d\n", x1);
15
16     return 0;
17 }
```

```
1  #include <stdio.h>
2
3  void increment(int *a){
4      *a = *a + 1;
5  }
6
7  int main(){
8      int x;
9
10     x = 7;
11     printf("x = %d\n", x);
12
13     increment(&x);
14     printf("x = %d\n", x);
15
16     return 0;
17 }
```

```
1  #include <stdio.h>
2
3  void minmax(int v[], int len, int *min, int *max){
4      int i;
5      for (i=1, *min=v[0], *max=v[0]; i<len; i++) {
6          if (v[i] < *min) *min = v[i];
7          if (v[i] > *max) *max = v[i];
8      }
9  }
10
11 int main(){
12     int n[] = {28,7,14,9,31,18}, min, max;
13
14     minmax(n, 6, &min, &max);
15     printf("min = %d, max = %d\n", min, max);
16
17     return 0;
18 }
```

Exercício 12

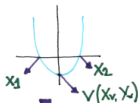
Crie uma função que receba como parâmetro: a , b e c por valor e x_1 e x_2 por referência, todos os parâmetros reais.

A função deve calcular as raízes de uma equação do segundo grau com coeficientes a , b e c e calcular as raízes x_1 e x_2 .

O retorno da função deverá ser um número inteiro indicando a quantidade de raízes da função.

Válida para:
EQUAÇÕES DO 2º GRAU

$$ax^2 + bx + c = 0$$



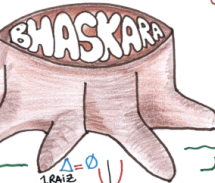
COORDENADAS DO VÉRTICE

$$X_v = \frac{-b}{2a}$$

$$Y_v = \frac{-\Delta}{4a}$$

Fórmula

de



$\Delta = 0$
1 RAÍZ

$\Delta > 0$
2 RAÍZES

$\Delta < 0$
0 RAÍZES

Estudo das raízes

Como resolver

1 → DISCRIMINANTE (DELTA)

$$\Delta = b^2 - 4ac$$

2 → BHASKARA

$$X = \frac{-b \pm \sqrt{\Delta}}{2a}$$

3 → RESULTADO (RAÍZES)

$$X_1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$X_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

Exercício 13

Implemente a função `void decToBits(int num, int *bitA, int *bitB)` que recebe como parâmetro um número inteiro de 0 a 3 (`num`) inclusive e atribui às variáveis `bitA` e `bitB` os bits do número binário correspondente. Exemplo: `num=2` \rightarrow `bitA=1` e `bitB=0`.

- Programas em C, quase sempre, são escritos usando funções disponíveis na biblioteca padrão do C (*C standard library*).
- A C standard library é formada por diversas funções de uso comum, tais como:

Cabeçalho	Descrição
<math.h>	Contém protótipos de funções matemáticas
<stdlib.h>	Contém protótipos de funções para conversão de números em texto e texto em números, alocação de memória, números aleatórios entre outras
<string.h>	Contém protótipos de funções para processamento de strings.
<time.h>	Contém protótipos de funções para manipular horários e datas.

Tabela: Exemplos de bibliotecas da linguagem C

- Os arquivos de extensão `.h` são chamados de **arquivos de cabeçalho** (ou *headers*).

Função	Descrição
<code>sqrt(x)</code>	Raiz quadrada de \sqrt{x}
<code>exp(x)</code>	Exponencial (e^x)
<code>log(x)</code>	Logaritmo natural ($\log_e(x)$)
<code>log10(x)</code>	Logaritmo na base 10 ($\log_{10}(x)$)
<code>fabs(x)</code>	Valor absoluto $ x $
<code>ceil(x)</code>	Teto ($\lceil x \rceil$)
<code>floor(x)</code>	Piso ($\lfloor x \rfloor$)
<code>pow(x,y)</code>	Potência (x^y)
<code>sin(x)</code>	Seno ($\text{sen}(x)$)
<code>cos(x)</code>	Cosseno ($\cos(x)$)
<code>tan(x)</code>	Tangente ($\tan(x)$)

Tabela: Exemplos de funções matemáticas da biblioteca `math.h`

- O programador pode criar arquivos de cabeçalho personalizados.
- Um header definido pelo programador pode ser incluído usando a diretiva de pré-processador `#include`.
- **Por exemplo**, o arquivo de cabeçalho `FuncoesUteis.h` pode ser incluído em nosso programa da seguinte forma

```
#include "FuncoesUteis.h"
```

- Os headers contém definições de protótipos de função que você pode adicionar à sua implementação.
- Mas eles precisam acompanhados de um arquivo de implementação que (com extensão `.c`):

`FuncoesUteis.c`

Código: FuncoesUteis.h

```
1 void Int_printArray(int v[], int length);  
2 void Float_printArray(float v[], int length);
```

Código: FuncoesUteis.c

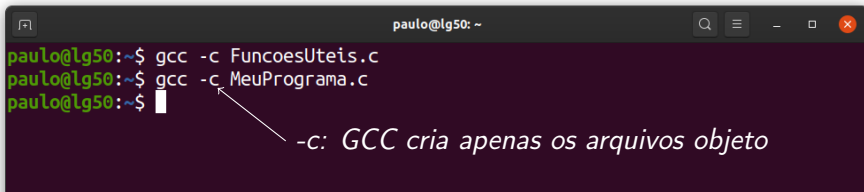
```
1 #include <stdio.h>  
2 #include "FuncoesUteis.h"  
3 void Int_printArray(int v[], int length) {  
4     int i;  
5     for (i=0; i < length; i++)  
6         printf("%d ", v[i]);  
7     printf("\n");  
8 }  
9 void Float_printArray(float v[], int length) {  
10    int i;  
11    for (i=0; i < length; i++)  
12        printf("%f ", v[i]);  
13    printf("\n");  
14 }
```

- A união de arquivos header e suas implementações formam o que chamamos de **Módulos** em C.
- **Por exemplo**, o programa abaixo faz uso do módulo `FuncoesUteis` para realizar a impressão de dois vetores.

Código: `FuncoesUteis-main.c`

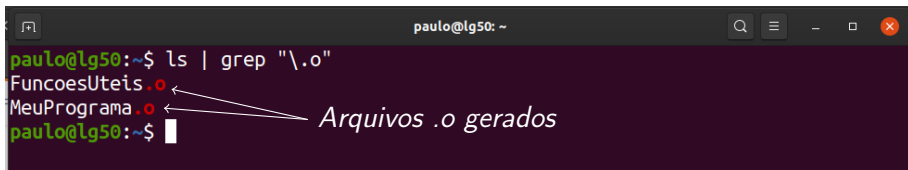
```
1  #include "FuncoesUteis.h"
2
3  int main() {
4      int n[] = {1,3,5,7,9};
5      float m[] = {2.0, 4.1, 6.001, 8.0, 10.9};
6
7      Int_printArray(n, 5);
8      Float_printArray(m, 5);
9
10     return 0;
11 }
```

- O header e uma implementação se conectam na etapa de **ligação** (*linking*) do processo de compilação.
- Para que possamos realizar essa ligação precisamos dos **arquivos objeto** do nosso programa e da implementação.



```
paulo@lg50: ~  
paulo@lg50:~$ gcc -c FuncoesUteis.c  
paulo@lg50:~$ gcc -c MeuPrograma.c  
paulo@lg50:~$
```

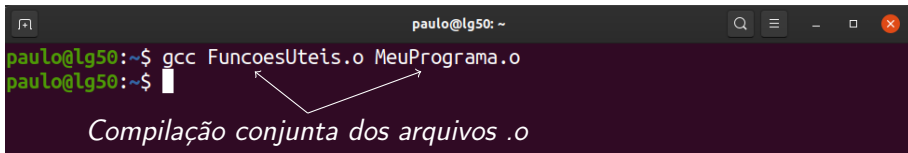
-c: GCC cria apenas os arquivos objeto



```
paulo@lg50: ~  
paulo@lg50:~$ ls | grep "\.o"  
FuncoesUteis.o  
MeuPrograma.o  
paulo@lg50:~$
```

Arquivos .o gerados

Após sua geração, compilamos os arquivos objeto juntos para formar o executável.



```
paulo@lg50: ~  
paulo@lg50:~$ gcc FuncoesUteis.o MeuPrograma.o  
paulo@lg50:~$
```

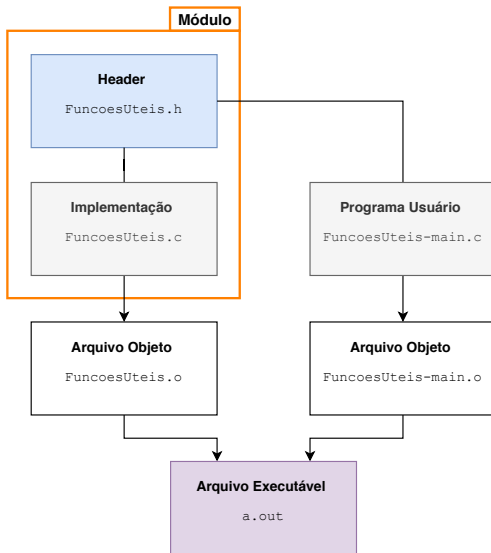
Compilação conjunta dos arquivos .o

- Etapas para a compilação e execução do programa que usando o módulo externo FuncoesUteis.

1. `gcc -c FuncoesUteis.c FuncoesUteis-main.c`

2. `gcc FuncoesUteis.o FuncoesUteis-main.o`

3. `./a.out`



```
1  for (x = 0; x < TAM-1; x++ ) {  
2      for(y = 0; y < TAM-1-x; y++) {  
3          if (vetor[y] > vetor[y+1]) {  
4              aux = vetor[y];  
5              vetor[y] = vetor[y+1];  
6              vetor[y+1] = aux;  
7          }  
8      }  
9  }
```

Exercício 14

O código acima é uma implementação do algoritmo de ordenação *Bubble Sort*, crie um módulo com uma função para este algoritmo e um programa de demonstração em que faz uso da sua biblioteca para ordenar os elementos de um vetor.