

Técnica de Programação Modular

Técnicas de Programação para Sistemas Embarcados I



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Thiago Werlley Bandeira da Silva

Universidade Federal do Ceará

April 23, 2025



Funções

Características:

- » São trechos de códigos fora do programa principal
- » Implementam as subrotinas do programa
- » São blocos de instruções que realiza tarefas específicas
- » Sendo executados quantas vezes for necessários
- » Podem ser chamados em diversos pontos do programa

Funções

Permitem ao desenvolvedor separar seus programas em blocos, e com isso construir programas grandes e complexos através de implementação modular.

Funções

- Vantagens:
 - Organiza o código:
 - Cada função é um algoritmo
 - Dividir um programa complicado em várias funções simples
 - Evita repetições de código semelhante
 - Escrever o código apenas uma vez
 - Usar a mesma função várias vezes para variáveis diferentes
 - Simplifica e agiliza a programação



Definições de Funções

```
tipo nome(parâmetros) {  
    declarações de variáveis  
    ...  
    instruções  
    ...  
    return(valor);  
}
```

Funções

Nomes:

- Identifica o trecho de código da função
- O nome é usado na chamada
- Formação igual a nome de variável

```
tipo nome (parâmetros) {  
    declarações  
    instruções  
    return(valor);  
}
```

Funções

Parâmetros:

- » Defini dados de entrada
- » São variáveis locais, declaradas e atribuídas na chamada da função
- » Lista de tipos e nomes

```
tipo nome(parâmetros) {  
    declarações  
    instruções  
    return(valor);  
}
```

Funções

Parâmetros

- » Lista de 2 parâmetros:
- » Variáveis locais da função: float nota1 e nota2

```
... media(float nota1, float nota2) {
```

Declarações

Instruções

...

}

```
tipo nome(parâmetros) {  
    declarações  
    instruções  
    return(valor);  
}
```

Parâmetros na chamada da função

- Ao chamar a função, o valor de cada parâmetro deve ser informado
- Regras:
 - Um valor para cada parâmetro
 - Valores compatíveis com o tipo do parâmetro
 - Valores na mesma ordem que na declaração

Chamada de função:

`nome(parametro1, parametro2, ...)`

Funções

Parâmetros na chamada da função

```
float resultado;  
resultado = media (2.0, 5.0);
```

Quando a
função executa $\text{nota1} = 2.0$ $5.0 = \text{nota2}$

Declaração da função

```
... media (float nota1, float nota2) {  
    ...  
    // Corpo da função  
    ...  
}
```

Funções

Função sem parâmetros

```
...funcaoSemParametros (void)
{
    Declarações
    Instruções
    ...
}
```

Chamada de função:
funcaoSemParametros ()

```
tipo nome(void) {
    declarações
    instruções
    return(valor);
}
```

Funções

Corpo da função:

- » Código para o algoritmo da função
- » Declara variáveis locais, além dos parâmetros
- » Parâmetros são variáveis locais, já pré-declaradas

```
tipo nome(parâmetros) {  
    declarações  
    instruções  
    return(valor);  
}
```

Funções

Corpo da função

```
... media(float nota1, float nota2) {  
    float media;  
  
    media = (nota1 + nota2) / 2.0;  
  
    ...  
}
```

Funções

Valor de retorno

» tipo da função:

- Indica o domínio do resultado da função
- Qualquer tipo válido para variáveis

```
tipo nome(parâmetros) {  
    declarações  
    instruções  
    return valor;  
}
```

Funções

Valor de retorno

- Comando **return**
- Termina a execução da função
- Define o resultado (valor de retorno)
- Deve ser compatível com o tipo da função

```
tipo nome(parâmetros) {  
    declarações  
    instruções  
    return valor;  
}
```

Definição de Funções

```
...  
int funcaoA(int a) {  
    ...  
}  
float funcaoB(char c, double r) {  
    ...  
}  
char funcaoC(char c, int x) {  
    ...  
}  
//  
Int main( ... ) {  
    ...  
}
```

Definição de Funções

....

int funcaoA(int a) {

...

}

float funcaoB(char c, double r) {

...

}

char funcaoC(char c, int x) {

...


}

//

Int main(...) {

....

}



funcaoA conhecida aqui

Definição de Funções

....

```
int funcaoA(int a) {
```

....

```
}
```

```
float funcaoB(char c, double r) {
```

....

```
}
```

```
char funcaoC(char c, int x) {
```

....

```
}
```

```
//
```

```
Int main( ... ) {
```

....

```
}
```

funcaoA conhecida aqui

funcaoA, funcaoB conhecidas aqui

Definição de Funções

```
....  
int funcaoA(int a) {
```

```
....  
}
```

```
float funcaoB(char c, double r) {
```

```
....  
}
```

```
char funcaoC(char c, int x) {
```

```
....  
}
```

```
//  
Int main( ... ) {
```

```
....  
}
```

funcaoA conhecida aqui

funcaoA, funcaoB conhecidas aqui

funcaoA, funcaoB, funcaoC conhecidas aqui

Protótipos de funções

- Declaração de funções
- Conhecimento do formato da função para o compilador
- Protótipo é semelhante as declarações de variáveis
- Funções podem ser codificadas fora de ordem ou em outro arquivo

tipo nome_da_função (declaração_de_parâmetros);

Protótipos de Funções

```
...  
int funcaoA(int a);  
float funcaoB(char c, double r);  
char funcaoC(char c, int x);  
//  
Int main( ... ) {  
    ... }  
//  
int funcaoA(int a) {  
    ... }  
float funcaoB(char c, double r) {  
    ... }  
char funcaoC(char c, int x) {  
    ... }
```

Protótipos de Funções

....

int funcaoA(int a);

float funcaoB(char c, double r);

char funcaoC(char c, int x);

//

Int main(...) {

... }

//

int funcaoA(int a) {

... }

float funcaoB(char c, double r) {

... }

char funcaoC(char c, int x) {

... }



Protótipo de funções

Protótipos de Funções

....

int funcaoA(int a);

float funcaoB(char c, double r);

char funcaoC(char c, int x);

//

Int main(...) {

... }

//

int funcaoA(int a) {

... }

float funcaoB(char c, double r) {

... }

char funcaoC(char c, int x) {

... }

Protótipo de funções

funcaoA, funcaoB, funcaoC conhecidas aqui

Protótipos de Funções

....

int funcaoA(int a);

float funcaoB(char c, double r);

char funcaoC(char c, int x);

//

Int main(...) {

... }

//

int funcaoA(int a) {

... }

float funcaoB(char c, double r) {

... }

char funcaoC(char c, int x) {

... }

Protótipo de funções

funcaoA, funcaoB, funcaoC conhecidas aqui

Corpo das funções

Exemplo de Funções

```
1  #include <stdio.h>
2
3  float media(float nota1, float nota2){
4      float media=0;
5
6      media = (nota1+nota2)/2;
7
8      return(media);
9  }
10
11 int main(){
12     float nota1, nota2;
13
14     printf("Entre com as notas:");
15     scanf("%f%f",&nota1, &nota2);
16
17     printf("media:%f", media(nota1, nota2));
18
19     return(0);
20 }
```


Exemplo de Funções

```
1  #include <stdio.h>
2
3  float media(float , float);
4
5  int main(){
6      float nota1, nota2;
7
8      printf("Entre com as notas:");
9      scanf("%f%f",&nota1, &nota2);
10
11     printf("media:%f", media(nota1, nota2));
12
13     return(0);
14 }
15
16 float media(float nota1, float nota2){
17     float media=0;
18
19     media = (nota1+nota2)/2;
20
21     return(media);
```

Funções

Função main:

- » Onde se encontra o programa principal
- » Várias declarações possíveis:

```
void main(void) { ... }
```

```
int main(void) { ... }
```

```
int main(int argc, char *argv[]) {...}
```

Funções

Função main:

- » Os parâmetros `argc` e `argv` da acesso à linha de comando
- » O `argc` (argument count) é um inteiro e possui o número de argumentos, sendo no mínimo 1, pois o nome do programa é contado como sendo o primeiro argumento
- » O `argv` (argument values) é um ponteiro para uma matriz de strings. Cada string desta matriz é um dos parâmetros da linha de comando. O `argv[0]` sempre aponta para o nome do programa

```
$ ./program 10 2 1
```

```
argc = 4
```

```
argv[0] = program
```

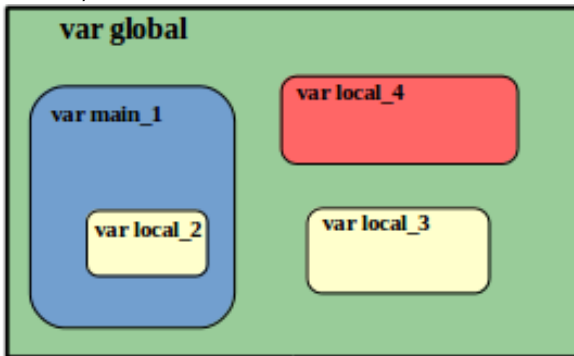
```
argv[1] = 10
```

```
argv[2] = 2
```

```
argv[3] = 1
```

Declaração de variável

- Tempo que o valor da variável permanece na memória:
 - Início até o final do programa
 - Dentro de um bloco com chaves
 - Chamada até final da função



Funções

- Variáveis declaradas localmente em uma função:
 - Tempo de vida: durante a execução da função
 - Durante a execução a variável é visível, acessível
 - Durante a execução está associada à posições de memória
 - Em duas execuções, ocupam posições de memória diferentes

```
1 {  
2     int var = 0;  
3     ...  
4 }
```

Funções

- Declaração dentro do corpo da função
 - Visibilidade apenas na função

```
1  int funcao(void);  
2  
3  int main(void) {  
4      ...  
5      funcao();  
6  }  
7  
8  int funcao(void) {  
9      int var = 0;  
10     ...  
11 }
```

Funções

- Variáveis declaradas fora do corpo das funções
 - Sobrevivem por toda a execução do programa
 - Permanecem ocupando a mesma posição de memória

```
1  int v = 0;
2  int funcao(void);
3
4  int main(void) {
5      ...
6      funcao();
7  }
8
9  int funcao(void) {
10     ...
11 }
```

Funções

```
1  int x = 0;
2  int funcA(void);
3  int funcB(void);
4
5  int main(void) {
6      ...
7      funcA();
8      funcB();
9  }
10
11 int funcA(void) {
12     int v = 0;
13     ...
14 }
15
16 int funcB(void) {
17     int w = 0;
18     ...
19 }
```


Redefinição de nomes

- Condições:
 - Variáveis com mesmo nome
 - Sobreposição de escopo
- Consequência:
 - Declaração mais recente torna inacessível outras declarações

```
1  int var = 0;
2  int func(void);
3
4  int main(void) {
5      ...
6      func();
7  }
8
9  int func(void) {
10     int var = 0;
11     ...
12 }
```

Variável global

- Declarada fora das funções
- Tempo vida:
 - até fim do programa
 - acessível a todas funções do programa, exceto quando há sobreposição
- Inicialização:
 - ao iniciar o programa

```
1  int var = 0;  
2  
3  int func1(void) {  
4      ...  
5  }  
6  
7  int func2(void) {  
8      ...  
9  }
```

Funções

```
1  int v = 0;
2  void func1(void);
3  void func2(void);
4
5  int main(int argc, char argv[]) {
6      func1();
7      func2();
8      func1();
9
10     return(0);
11 }
12
13 void func1(void) {
14     v++;
15     printf("f1: v=%d\n", v);
16 }
17
18 void func2(void) {
19     v+=2;
20     printf("f2: v=%d\n", v);
21 }
```

Resultado:
func1: v =
func2: v =
func1: v =

Variável local

- Declarada dentro das funções
- Tempo vida:
 - do início até o fim da função
 - nova execução não lembra valor anterior
- Inicialização:
 - no início da função

```
void func1(void){  
    int v = 0;  
    ...  
}  
void func2(void){  
    int w = 0;  
    ...  
}
```

Funções

```
1  int v;  
2  void func1();  
3  void func2();  
4  
5  void main(int argc, char argv[]) {  
6      func1();  
7      func2();  
8      func1();  
9  
10     return(0);  
11 }  
12  
13 void func1() {  
14     int v = 0; v++;  
15     printf("f1: v=%d\n", v);  
16 }  
17  
18 void func2() {  
19     int v = 0; v+=2;  
20     printf("f2: v=%d\n", v);  
21 }
```

Resultado:
func1: v =
func2: v =
func1: v =