Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

FUNÇÕES

Funções

- As funções também são chamadas de sub-rotinas, subprogramas e modularização;

Funções

- As funções também são chamadas de sub-rotinas, subprogramas e modularização;
- São carregadas na memória apenas uma vez e, podem ser executadas quantas vezes forem necessárias;

Funções

- As funções também são chamadas de sub-rotinas, subprogramas e modularização;
- São carregadas na memória apenas uma vez e, podem ser executadas quantas vezes forem necessárias;
- Como o problema pode ser subdividido em pequenas tarefas, os programas tendem a ficar menores e mais organizados;

Funções

- Os programas são executados linearmente

Funções

- Os programas são executados linearmente
  - Uma linha após a outra;

Funções

- Os programas são executados linearmente
  - Uma linha após a outra;
- Com a utilização de funções é possível realizar desvios na execução do programa;

Funções

- Os programas são executados linearmente
  - Uma linha após a outra;
- Com a utilização de funções é possível realizar desvios na execução do programa;
- Estes desvios são efetuados quando uma função é chamada pelo programa principal.

Funções

Forma geral de uma função:

```
1 tipo_retornado nome_função ( parâmetros ) {
2    conjunto de declarações e
3    conjunto de comandos
4 }
5
```

Exemplo

```
1  float calculo(float salario)
2  {
3      float perc, valor;
4      scanf("%f",&perc);
5      valor = (salario*perc)/100;
6      return valor;
7  }
8
9  void main()
10 {
11     float sal, aum, novo_sal;
12     scanf("%f",&sal);
13     aum = calculo(sal);
14     novo_sal = sal + aum;
15     printf("novo salário é: %.2f", novo_sal);
16 }
```

Por que usar funções?

- Evita escrita repetida de código (uma certa sequência de comandos deve ser repetida em vários lugares de um programa).

Por que usar funções?

- Evita escrita repetida de código (uma certa sequência de comandos deve ser repetida em vários lugares de um programa).
  - Economiza o tempo gasto com o trabalho de copiar estas sequências;

Por que usar funções?

- Evita escrita repetida de código (uma certa sequência de comandos deve ser repetida em vários lugares de um programa).
  - Economiza o tempo gasto com o trabalho de copiar estas sequências;
  - Evita a necessidade de mudar em múltiplos lugares caso deseje alterar o seu funcionamento;

Por que usar funções?

- Evita escrita repetida de código (uma certa sequência de comandos deve ser repetida em vários lugares de um programa).
  - Economiza o tempo gasto com o trabalho de copiar estas sequências;
  - Evita a necessidade de mudar em múltiplos lugares caso deseje alterar o seu funcionamento;
- Dividir grandes tarefas de computação em tarefas menores:

Por que usar funções?

- Evita escrita repetida de código (uma certa sequência de comandos deve ser repetida em vários lugares de um programa).
  - Economiza o tempo gasto com o trabalho de copiar estas sequências;
  - Evita a necessidade de mudar em múltiplos lugares caso deseje alterar o seu funcionamento;
- Dividir grandes tarefas de computação em tarefas menores:
  - Facilita o gerenciamento de grandes sistemas e

Por que usar funções?

- Evita escrita repetida de código (uma certa sequência de comandos deve ser repetida em vários lugares de um programa).
  - Economiza o tempo gasto com o trabalho de copiar estas sequências;
  - Evita a necessidade de mudar em múltiplos lugares caso deseje alterar o seu funcionamento;
- Dividir grandes tarefas de computação em tarefas menores:
  - Facilita o gerenciamento de grandes sistemas e
  - Aumenta a confiabilidade dos mesmos.

Parâmetros

- A declaração de parâmetros é uma lista de variáveis juntamente com seus tipos:

```
tipo nome1, tipo nome2, ... , tipoN nomeN
```

Corpo

- De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função.

Corpo

- De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função.
  - Uma função é construída com o intuito de realizar uma tarefa específica e bem-definida.

Corpo

- De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função.
  - Uma função é construída com o intuito de realizar uma tarefa específica e bem-definida.
  - As operações de entrada e saída de dados (funções **scanf()** e **printf()**) devem ser feitas em quem chamou a função (por exemplo, na **main()**).

Corpo

- De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função.
  - Uma função é construída com o intuito de realizar uma tarefa específica e bem-definida.
  - As operações de entrada e saída de dados (funções **scanf()** e **printf()**) devem ser feitas em quem chamou a função (por exemplo, na **main()**).
  - Isso assegura que a função construída possa ser utilizada nas mais diversas aplicações, garantindo a sua generalidade.

Retorno

- Uma função pode retornar qualquer valor válido em C

Retorno

- Uma função pode retornar qualquer valor válido em C
  - tipos pré-definidos (int, char, float e double);

Retorno

- Uma função pode retornar qualquer valor válido em C
  - tipos pré-definidos (int, char, float e double);
  - tipos definidos pelo usuário (struct).

Retorno

- Uma função pode retornar qualquer valor válido em C
  - tipos pré-definidos (int, char, float e double);
  - tipos definidos pelo usuário (struct).
- Uma função que retorna nada é definida colocando-se o tipo **void** como valor retornado

Comando return

- O valor retornado pela função é dado pelo comando **return**. Forma geral:

  return *valor ou expressão*;

  ou

  return;

Comando return

- O valor retornado pela função é dado pelo comando **return**. Forma geral:

  return *valor ou expressão*;

  ou

  return;

- É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

Chamada de Função

- Para usar uma função, devemos chamá-la dentro da função principal (main) ou dentro de outra função

Chamada de Função

- Para usar uma função, devemos chamá-la dentro da função principal (main) ou dentro de outra função
- Para chamar a função, basta escrever seu nome e colocar os parâmetros necessários

Chamada de Função

- Para usar uma função, devemos chamá-la dentro da função principal (main) ou dentro de outra função
- Para chamar a função, basta escrever seu nome e colocar os parâmetros necessários
- Se a função retorna algum valor, pode-se copiar este valor para um variável ou usá-lo em alguma expressão

Exemplos

```
1  // função sem retorno e sem parâmetros de entrada
2  void MensagemBoasVindas() {
3      printf("\n");
4      printf("======================\n");
5      printf(" Seja Bem-Vindo \n");
6      printf("======================\n");
7      printf("\n");
8  }
9  int main()
10 {
11     // Chamando a função
12     MensagemBoasVindas();
13     ....
14
```

Exemplos

```
1  // função sem parâmetros de entrada, mas com retorno
2  char MenuPrincipal(){
3      char op;
4      printf("Escolha uma opção: \n\n");
5      printf("1 - Novo Jogo\n");
6      printf("2 - Carregar Jogo\n");
7      printf("3 - Sair\n");
8      setbuf(stdin,NULL);
9      scanf("%c",&op);
10 }
11 return op;
12 int main() {
13     char escolha;
14     MensagemBoasVindas();
15     escolha = MenuPrincipal();
16
```

```
1 // função com retorno e com parâmetros
2 int VerificaAprovacao(double nota, int faltas) {
3     int aprovado = 1;
4     if ( (faltas > 18)  (nota < 60.0) ) {
5         aprovado = 0;
6     }
7 }
8 return aprovado;
9 int    main(){
10    int ap;
11    ap = VerificaAprovacao(4.0, 5);
12    if (ap) {
13        printf("Aprovado!!!");
14    }
15
```

```
1 // supor 0 -> aprovado
2 // supor 1 -> reprovado
3 // supor 2 -> reprovado por falta
4
5 if (VerificaAprovacao(nota, faltas) == 1) {
6    printf("Aprovado!!!");
7 } else if (VerificaAprovacao(nota, faltas) == 0) {
8    printf("Reprovado!!!");
9 } else if (VerificaAprovacao(nota, faltas) == 2) {
10   printf("Reprovado por falta");
11 }
12
```

```
1 // supor 0 -> aprovado
2 // supor 1 -> reprovado
3 // supor 2 -> reprovado por falta
4
5 ap = VerificaAprovacao(nota, faltas);
6
7 if (ap == 1) {
8    printf("Aprovado!!!");
9 } else if (ap == 0) {
10   printf("Reprovado!!!");
11 } else {
12   printf("Reprovado por falta");
13 }
14
```

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

Prof. Thiago Pirola Ribeiro