Lógica de programação

Sub-rotinas

Walisson Pereira

walisson_pereira@uvanet.br
Universidade Estadual Vale do Acaraú

Roteiro

Introdução

Passagem de parâmetros

- Passagem de parâmetro por valor
- Passagem de parâmetro por referência

Variáveis locais e globais

Funções recursivas

Bibliotecas

Exercícios

Referências

Introdução

Introdução

Sub-rotinas (ou funções ou procedimento) são blocos de instruções que realizam tarefas específicas.

- O código de uma sub-rotina é carregado uma vez e pode ser executado quantas vezes forem necessárias.
- Como o problema pode ser subdividido em pequenas tarefas, os programas tendem a ficar menores e mais organizados.

Exemplo de uso:

```c
#include <stdio.h>

void soma (int a, int b) {
    int resultado = a + b;
    printf("%d\n", resultado);
}

int main () {
    soma(2, 9);
    soma(7, 8);
    soma(10, 15);
    return 0;
}
```

Introdução

Uma função também pode retornar um resultado para quem a chamou.

```
1  #include <stdio.h>
2
3  int epar (int x) {
4      return x % 2 == 0;
5  }
6
7  int main () {
8      printf("%d\n", epar(2));
9      printf("%d\n", epar(3));
10     printf("%d\n", epar(10));
11     return 0;
12 }
```

Introdução

Uma função pode chamar outra função.

```c
#include <stdio.h>

int epar (int x) {
    return x % 2 == 0;
}
void par_ou_impar(int valor) {
    if (epar(valor))
        printf("%d e par\n", valor);
    else
        printf("%d e impar\n", valor);
}
int main () {
    par_ou_impar(4);
    par_ou_impar(5);
    return 0;
}
```

Introdução

Observe que a função **par_ou_impar** deve ser declarada depois da função **epar**.

Se você inverter a ordem, dependendo do compilador, ele poderá indicar um alerta ou não compilar.

```
1  void par_ou_impar(int valor) {
2      if (epar(valor))
3          printf("%d e par\n", valor);
4      else
5          printf("%d e impar\n", valor);
6  }
7  int epar (int x) {
8      return x % 2 == 0;
9  }
```

Introdução

Quando o programa possui várias sub-rotinas, uma forma de organizar melhor o seu código, é declará seu cabeçalho antes da função **main** e depois implementar as demais funções.

```
1  #include <stdio.h>
2
3  int epar (int x);
4  void par_ou_impar(int valor);
5
6  int main () {
7      //codigo
8  }
9  int epar (int x) {
10     //codigo
11 }
12 void par_ou_impar(int valor) {
13     //codigo
14 }
```

Introdução

```c
#include <stdio.h>
int epar (int x);
void par_ou_impar(int valor);

int main () {
    par_ou_impar(4);
    par_ou_impar(5);
    return 0;
}
int epar (int x) {
    return x % 2 == 0;
}
void par_ou_impar(int valor) {
    if (epar(valor))
        printf("%d e par\n", valor);
    else
        printf("%d e impar\n", valor);
}
```

Introdução

Podemos usar alguns termos para especificar uma sub-rotina.

- **Função**: são sub-rotinas que retornam um valor.
- **Procedimento**: são sub-rotinas que não retorna um valor (não tem o return ao final e é do tipo void).

Introdução

Compare: um procedimento.

```c
#include <stdio.h>

void soma (int a, int b) {
    printf("%d\n, a + b");
}

int main () {
    soma(2, 3);
    return 0;
}
```

Compare: uma função.

```c
#include <stdio.h>

int soma (int a, int b) {
    return a + b;
}

int main () {
    int resultado = soma(2, 3);
    printf("%d\n", resultado);
    return 0;
}
```

Exercícios:

1. Escreva uma função que retorne o maior de dois números.
2. Escreva uma função que receba dois números e retorne **1** se o primeiro número for múltiplo do segundo. Caso contrário, retorne **0**.
3. Escreva uma função que receba o lado de um quadrado e retorne sua área.
4. Escreva uma função que receba a base e a altura de um triângulo e retorne sua área.

Passagem de parâmetros

Passagem de parâmetros

Existem duas formas de se passar parâmetros:

- por valor
- por referência

Passagem de parâmetro por valor

A passagem de parâmetro por valor significa que o valor da variável é enviado para a sub-rotina.

Em outras palavras, uma cópia do valor da variável é enviado para a sub-rotina.

Passagem de parâmetro por valor

```c
#include <stdio.h>

void soma (int x, int y)
{
    int s;
    s = x + y;
    printf("Soma = %d\n", s);
}

int main ()
{
    int a, b;
    scanf ("%d %d", &a, &b);
    soma (a, b);
}
```

Passagem de parâmetros

- A passagem de parâmetro por referência significa que a referência da variável é enviado para a sub-rotina.
- Diferente da passagem de parâmetro por valor que envia o valor, o passagem de parâmetro por referência envia "a própria variável".
- Em outras palavras, na passagem de parâmetros por valor, caso um valor seja passado, esse valor é apenas modificado dentro do escopo da sub-rotina (local). Na passagem de parâmetro por referência, se a variável passada for modificada, ela modifica globalmente.

Passagem de parâmetro por referência

```c
#include <stdio.h>

void soma (int x, int y, int *s)
{
    *s = x + y;
}
int main ()
{
    int a, b, c;
    scanf ("%d %d", &a, &b);
    soma (a, b, &c);
    printf("Soma = %d\n", c);
}
```

Observem que o procedimento **soma** recebeu as referências das variáveis **a**, **b** e **s** e modificou **s** e essa mudança refletiu na variável **c**.

Passagem de parâmetro por referência

A linguagem C não permite que vetores e matrizes sejam passados na íntegra como parâmetro para uma sub-rotina. Para resolver esse problema, deve-se passar apenas o endereço da posição inicial do vetor ou da matriz. Esse endereço é obtido utilizando-se o nome do vetor sem o índice entre os colchetes. Isso quer dizer que é possível passar um vetor para uma sub-rotina somente se essa passagem for por referência.

Passagem de parâmetro por referência

```c
#include <stdio.h>
#include <stdlib.h>
void gera_vetor (int v[], int n) {
    int i;
    for (i = 0; i < n; i++)
      v[i] = rand() % 10;
}
void imprime_vetor(int v[], int n) {
    int i;
    for (i = 0; i < n; i++)
      printf("%d ", v[i]);
    printf("\n");
}
int main () {
    int vet1[5];
    gera_vetor (vet1, 5);
    imprime_vetor(vet1, 5);
}
```

Variáveis locais e globais

Variáveis locais e globais

Quando usamos funções, começamos a trabalhar com variáveis internas ou locais e com variáveis externas ou globais. A diferença entre elas é a visibilidade ou escopo.

- Uma **variável local** a uma função existe apenas dentro dela, sendo normalmente inicializada a cada chamada. Assim, não podemos acessar o valor de uma variável local fora da função que a criou e, por isso, passamos parâmetros e retornamos valores nas funções, de forma a possibilitar a troca de dados no programa.
- Uma **variável global** é definida fora de uma função, podendo ser vista por todas as funções do módulo (programa) e por todos os módulos que importam o módulo que a definiu.

Variáveis locais e globais

As variáveis **a**, **b** e **s** são variáveis globais.

```
1  #include <stdio.h>
2
3  int a, b, s;
4
5  void soma () {
6      s = a + b;
7  }
8
9  int main () {
10     scanf("%d %d", &a, &b);
11     soma();
12     printf("Soma = %d\n", s);
13 }
```

Variáveis locais e globais

Cuidado ao usar variáveis globais com o mesmo nome das variáveis locais.

```
1  #include <stdio.h>
2
3  int a, b, s = 0;
4
5  void soma () {
6      int s;
7      s = a + b;
8  }
9
10 int main () {
11     scanf("%d %d", &a, &b);
12     soma();
13     printf("Soma = %d\n", s);
14 }
```

Funções recursivas

Funções recursivas

Já vimos que uma função pode chamar uma outra função.

Além disso, uma função pode chamar a si mesma.

Faça o teste: cálculo do fatorial

```c
#include <stdio.h>

int fatorial (int n) {
    if (n == 0 || n == 1)
      return 1;
    else
      return n * fatorial (n - 1);
}

int main () {
    int valor;
    scanf("%d", &valor);
    printf("Fatorial de %d = %d\n", valor, fatorial(valor));
}
```

Funções recursivas

Exemplo: fatorial modificado para facilitar o rastreamento

```c
#include <stdio.h>
int fatorial (int n) {
    printf("Calculando o fatorial de %d\n", n);
    if (n == 0 || n == 1) {
      printf("Retornando o fatorial de %d = 1\n", n);
      return 1;
    }
    else {
      int fat = n * fatorial (n - 1);
      printf("Retornando o fatorial de %d = %d\n", n, fat);
      return fat;
    }
}
int main () {
    int valor; scanf("%d", &valor);
    printf("Fatorial de %d = %d\n", valor, fatorial(valor));
}
```

Bibliotecas

Bibliotecas

Em geral, usamos sub-rotinas para evitar reescrever trechos de código que são utilizados em lugares diferentes do programa principal.

Bibliotecas

Entretanto, se mais de um programa que você for criar necessitar do mesmo conjunto de sub-rotina, ao invés de reescrevê-la em cada programa, o ideal é que elas pudessem ser escritas apenas uma vez e usada quando forem necessárias.

Uma forma de contornar esta limitação é criar bibliotecas próprias.

Bibliotecas

Na ciência da computação, biblioteca é uma coleção de subprogramas utilizados no desenvolvimento de software.

Na linguagem C, para criar uma biblioteca é necessário gerar um novo arquivo e dentro dele pôr o código de todas as funções que farão parte dessa biblioteca. Este arquivo deve ser salvo com a extensão .h e deve ser compilado normalmente.

Faça o teste (1/2): crie um arquivo com o nome biblioteca.h

```
1 int some (int a, int b) {
2   return a + b;
3 }
4 int subtraia (int a, int b) {
5   return a - b;
6 }
7 int multiplique (int a, int b) {
8   return a * b;
9 }
```

Bibliotecas

Faça o teste (2/2): crie um arquivo com o nome exemplo.c no mesmo diretório do biblioteca.h

```c
#include <stdio.h>
#include "biblioteca.h"

int main () {
  int n1, n2;
  scanf("%d %d", &n1, &n2);
  printf("Soma = %d\n", some (n1, n2));
  printf("Subtracao = %d\n", subtraia (n1, n2));
  printf("Multiplicacao = %d\n", multiplique (n1, n2));
}
```

No terminal:

```
gcc exemplo.c biblioteca.h -o executavel
./executavel
```

Bibliotecas

A forma de criação de biblioteca apresentada até aqui é a mesma apresentada no livro **Fundamentos da Programação de Computadores**. Para o que vamos trabalhar nesta disciplina, este formato é o suficiente.

Entretanto, para um código maior que é necessário manter uma melhor organização, há um outro modelo mais adequado.

Parte 1/4: Crie um arquivo chamado **biblioteca.h** contendo apenas o cabeçalho das funções que serão implementadas.

```
1 int some (int a, int b);
2 int subtraia (int a, int b);
3 int multiplique (int a, int b);
```

Parte 2/4: Crie um arquivo chamado **biblioteca.c** contendo a implementação das sub-rotinas indicadas em **biblioteca.h**.

```c
#include "biblioteca.h"

int some (int a, int b) {
  return a + b;
}
int subtraia (int a, int b) {
  return a - b;
}
int multiplique (int a, int b) {
  return a * b;
}
```

Parte 3/4: Crie um arquivo principal que irá usar a biblioteca criada.

```c
#include <stdio.h>
#include "biblioteca.h"

int main () {
  int n1, n2;
  scanf("%d %d", &n1, &n2);
  printf("Soma = %d\n", some (n1, n2));
  printf("Subtracao = %d\n", subtraia (n1, n2));
  printf("Multiplicacao = %d\n", multiplique (n1, n2));
}
```

Parte 4/4: No terminal, compile e execute

```
gcc codigo.c biblioteca.c -o executavel
./executavel
```

Exercícios

Exercícios

1. Faça uma sub-rotina que receba um número inteiro e positivo N como parâmetro e retorne a soma dos números inteiros existentes entre o número 1 e N (inclusive).
2. Crie uma sub-rotina que receba três números inteiros como parâmetros, representando horas, minutos e segundos, e os converta em segundos. Exemplo: 2h, 40 min e 10s corresponde a 9.610 segundos.
3. Crie uma sub-rotina que receba como parâmetro dois vetores de dez elementos inteiros positivos e retorne um vetor com a soma dos vetores.
4. Faça uma sub-rotina que receba como parâmetro um vetor com cinco números reais e retorne esses números ordenados de forma crescente.

Referências

Referências

1 ASCENCIO, A.F.G.; CAMPOS, E.A.V. de. Fundamentos da Programção de Computadores. Algoritmos, Pascal e C/C++. São Paulo: Pearson Prentice Hall, 2012.

2 VAREJÃO, F. M. V. Introdução à programação: uma nova abordagem usando C. Rio de Janeiro: Elsevier, 2015.

3 BACKES, A. Linguagem C: completa e descomplicada. Rio de Janeiro: Elsevier, 2013.