Linguagem de Programação I - ESP201

Profª Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal de Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

April 1, 2025

Modularização

Funções

Funções

Definição de Função: Uma função nada mais é do que uma sub-rotina usada em um programa. Na linguagem C, denominamos função como um conjunto de comandos que realiza uma tarefa específica em um módulo dependente de código. A função é referenciada pelo programa principal através do nome atribuído a ela. A utilização de funções visa modularizar um programa, o que é muito comum em programação estruturada. Desta forma podemos dividir um programa em várias partes, no qual cada função realiza uma tarefa bem definida.

Funções

Esqueleto de uma função:

```
tipo_de_retorno nome_da_função(listagem de parâmetros){
    instruções;
    retorno_da_função;
}
```

Funções

Parâmetros de uma função: Os parâmetros de uma função são as variáveis declaradas diretamente no cabeçalho da função. A finalidade dos parâmetros é fazer a comunicação entre as funções e a função principal. Chamamos de passagem de parâmetros ou passagem de valores entre as funções.

Funções

```
1  #include <stdio.h>
2
3  int multiplica(int N1, int N2) //multiplica recebe N1,N2 e retorna um int
4  {
5    int resultado;
6    resultado = N1 * N2;
7    return(resultado); //retornando o valor para main
8  }
9  /******************* função principal (main) **********************/
10 void main(void){
11   int V1, V2, resultado;
12   printf("Digite o primeiro valor:");
13   scanf("%d", &V1);
14   printf("Digite o segundo valor:");
15   scanf("%d", &V2);
16   resultado = multiplica(V1,V2);
17
18   printf("Resultado = %d\n", resultado);
19 }
```

Protótipo de Função

Protótipo de Função

O protótipo de uma função é basicamente, uma declaração da interface da função, ou seja, deve especificar:

- Tipo da função;
- Nome da função;
- Lista de parâmetros que a função necessita;

Exemplo:
int multiplica(int N1, int N2);

Protótipo de Função

```c
#include <stdio.h>

int multiplica(int N1, int N2); //protótipo da função multiplica
/******************* função principal (main) **********************/
void main(void){
  int V1, V2, resultado;
  printf("Digite o primeiro valor:");
  scanf("%d", &V1);
  printf("Digite o segundo valor:");
  scanf("%d", &V2);
  resultado = multiplica(V1,V2);

  printf("Resultado = %d\n", resultado);
}

int multiplica(int N1, int N2) //multiplica recebe N1,N2 e retorna um int
{
  int resultado;
  resultado = N1 * N2;
  return(resultado); //retornando o valor para main
}
```

Função SEM Retorno

Função SEM Retorno

Em C, é possível criar funções que não retornam nenhum valor. Normalmente, isto é feito quando queremos executar um bloco de comandos, mas estes comandos não precisam retornar nada. Neste caso, devemos usar **void** no tipo de retorno do cabeçalho da função. Se a função não recebe nenhum parâmetro, também colocamos **void** no local da listagem dos parâmetros. exemplo:

```c
#include <stdio.h>

void imprime_cabec(void){
  printf("*******************************\n");
  printf("*       LINGUAGEM C           *\n");
  printf("*******************************\n");

  return; /* retorno de uma função void  */
}

void main(){
    imprime_cabec();
}
```

Recursividade

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

Dizemos que esses problemas têm estrutura recursiva. Para resolver um problema desse tipo, podemos aplicar o seguinte método:

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

Dizemos que esses problemas têm estrutura recursiva. Para resolver um problema desse tipo, podemos aplicar o seguinte método:

- se a instância for pequena, resolva-a diretamente;
- senão, reduza-a a uma instância menor $\xrightarrow{entao}$ aplique o método à instância menor $\xrightarrow{entao}$ volte à instância original.

Exercícios:

1. Faça um algoritmo recursivo que encontre qual o maior valor de um vetor.
2. Faça um algoritmo recursivo que calcule o fatorial de um número e mostre suas parcelas. Ex:
   $6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$
3. A função de Fibonacci é definida assim: $F(0) = 0$, $F(1) = 1$ e $F(n) = F(n-1) + F(n-2)$ para $n > 1$. Descreva a função F em linguagem C. Faça uma versão recursiva e uma iterativa.

Exercícios:

1.

```c
#include <stdio.h>
#include <stdlib.h>

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam] > maior)
        return v[tam];
    else
        return maior;
}
```

Exercícios:

1.

```c
#include <stdio.h>
#include <stdlib.h>

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam-1] > maior)
        return v[tam-1];
    else
        return maior;
}

int main (void){
int v1[]={134,3,234,7,567,5,678,2,899,0};
int v[100];
int tam, d=0;

for(tam=0;d!=-1;tam++){
    printf("Digite os elementos do vetor, digite -1 para sair\n");
    scanf("%d",&d);
    if(d!=-1)
     v[tam] = d;
}

  int i = maior_r(tam,v);
  printf("Maior item do Vetor= %d\n", i);
}
```

Exercícios:

2.

```c
#include <stdio.h>
#include <stdlib.h>

int fat (int n){
    if (n == 1){
        printf("1 = ");
        return 1;
    }
    else{
        printf("%d*",n);
        return (n * fat(n-1));
    }
}

int main (void){
int n;

  printf("Informe o número que você deseja saber o fatorial:");
  scanf("%d",&n);
  printf("%d! = ",n);
  printf("%d\n", fat(n));
}
```

Exercícios:

3.

```c
void fibonacci (int n){
    int fib, fib1, fib2, i;

    if (n == 0) {
        printf("0\n");
    } else if (n == 1){
        printf("0, 1\n");
    } else{
        printf("0, 1, ");
        fib1 = 1;
        fib2 = 0;
        i = 2;
        while (i <= n){
            fib = fib1+fib2;
            if (i==n) printf("%d\n", fib);
            else printf("%d, ", fib);
            fib2 = fib1;
            fib1 = fib;
            i++;
        }
    }

}
```

Exercícios:

3.

```c
/**A função de Fibonacci é definida assim:   F(0) = 0,   F(1) = 1   e   F(n) = F(n-1) + F(n-2)
para n > 1. Descreva a função F em linguagem C. Faça uma versão recursiva e uma iterativa.**/
int fibonacci (int n){
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibonacci(n-1) + fibonacci(n-2);
}
int main (void){
int n, fib, i;

  printf("Informe o número de fatores das sequências de Fibonacci:");
  scanf("%d",&n);
  for (i = 0; i <= n; i++)
     printf("Parcela %d da sequência = %d\n",i,fibonacci(i));
}
```

Referências

Referências