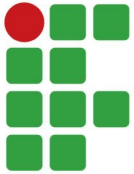


INSTITUTO FEDERAL
Bahia
Campus Santo Antônio de Jesus

http://www.portal.ifba.edu.br/santoantonio

LINGUAGEM DE PROGRAMAÇÃO

Prof. George Pacheco Pinto

Agenda

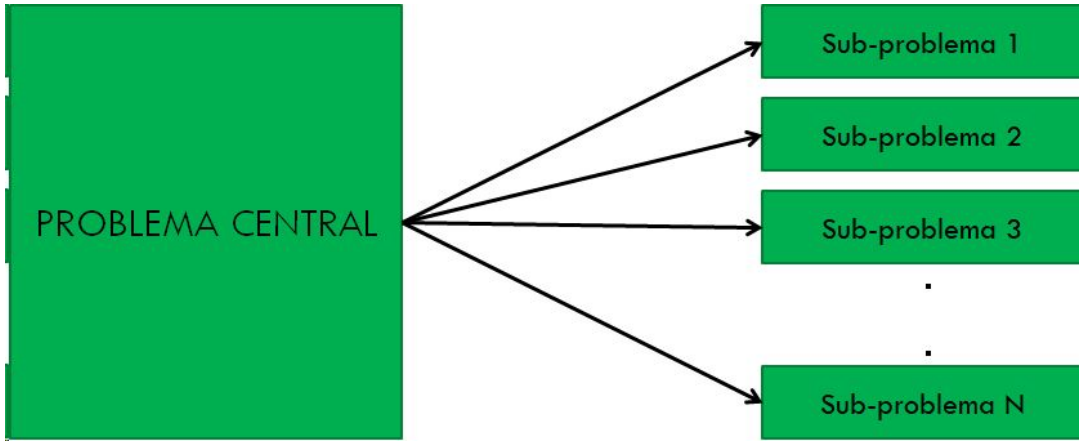
- Linguagem C
  - Modularização
    - Funções sem retorno (procedures)
    - Funções com retorno

Estruturação de código

- Problemas longos e complexos;
- Reaproveitamento de código existentes;
- Blocos grandes de código;
- Dificuldade no entendimento do programa;
- Dificuldade no desenvolvimento em grupo.

Método Top-Down

- Um formato estruturado de se resolver problemas.
- Dividir o problema em problemas menores.
- Resolver individualmente cada sub-problema.



Modularização

- Técnica para desenvolver algoritmo, por meio de refinamentos sucessivos;
- É a forma de dividir as tarefas em sub-algoritmos, e cada um desses módulos cuida de uma parte separada do problema;
- Módulos
  - Conjunto de comandos que constitui uma parte de um algoritmo principal, bem definida e independente em relação ao resto do algoritmo;
  - Cada módulo funciona como semelhantemente a um programa, tendo, em geral, entrada, processamento e saída.

Modularização

- Para resolver programas de forma estruturada utilizamos:
  - Programa Principal - main( )
  - Procedimento e Funções

- Programa Principal
  - É o bloco de código principal por onde o programa inicia e finaliza a execução.
  - O programa principal pode conter definições de variáveis, blocos de comandos e chamadas a sub-programas (Funções ou Procedimentos).
  - Todo programa em C deve possuir apenas 1 programa principal - main( )

Modularização

- Funções
  - Agrupa um conjunto de comandos e associa a ele um nome. O uso deste nome é uma chamada da função;
  - Após sua execução, programa volta ao ponto do programa situado imediatamente após a chamada. A volta ao programa que chamou a função é chamada de retorno;
  - Podem conter todos os elementos utilizados no programa principal (definições de variáveis, blocos de comandos e chamadas a outros sub-programas);
  - A principal diferença está no fato de que funções não são executadas automaticamente quando o programa inicia, elas precisam ser chamadas pelo programa principal;
  - Podem ou não receber parâmetros;
  - Uma grande vantagem da estrutura de funções é que as mesmas podem ser utilizadas repetidamente sem limitação o que torna o código reutilizável.

Funções sem retorno (Procedimentos)

- Funções sem retorno podem receber ou não dados (parâmetros) para desempenhar a sua tarefa;

- Quando chamados pelo programa principal, desviam o fluxo de execução do programa para o início do seu bloco de código e no final retornam para o fluxo principal do programa na instrução seguinte a que chamou o procedimento;

- **NÃO RETORNAM RESULTADO.**

Funções sem retorno – Sintaxe

```
void nome_funcao(parametros opcionais)
{
     //... bloco de código
}
```

** Por padrão quando o tipo da função não é explicitamente declarado, o valor de retorno é **int**.

- Parâmetros opcionais:
  - Valores que poderão ser enviados para a função que poderá ser utilizado na sua execução.
  - São “variáveis” e pode ser declarados com qualquer tipo(int, char, float,...).
  - Não é obrigatório a declaração de parâmetros nos procedimentos.

* A palavra void indica que o procedimento não retorna nenhum tipo de valor.

Exemplo Funções sem parâmetro

```
void imprimeBarra()
{
    printf(“********************”);
}


void pulaLinha()
{
    printf(“\n”);
}
```

Exemplo funções com parâmetro

```
void imprimePar(int numero)
{
    if(numero % 2 == 0){
     printf(“%d é par”mnumero);
    }
}

void imprimeTabuadaSoma(int numero)
{
    int i;
    for(i=0;i<10;i++){
     printf(“%d + %d = %d”,i,numero,i + numero);
    }
}
```

Protótipo/declaração de Funções

- Informa o compilador sobre o nome e como chamar a função;

```
#include<stdio.h>
//Prototipo da função
void imprimeLinha(void);
main(){
    imprimeLinha();
}
void imprimeLinha()
{
    printf(“**********”);
}
```

Utilizando Funções sem retorno (1)

```
#include<stdio.h>
//Prototipo da procedure
void imprimePar(int numero);

main(){
    imprimePar(1);
    imprimePar(2);
}
void imprimePar(int numero)
{
    if(numero % 2 == 0){
     printf(“%d é par”, numero);
    }
}
```

Utilizando Funções sem retorno (2)

```
#include<stdio.h>
//Protótipo da procedure
void imprimePar(int a);
void pulaLinha();
void imprimeLinha();

main(){
imprimeLinha();
pulaLinha()
imprimePar(2);
}
```

Utilizando Funções sem retorno (3)

```
void imprimePar(int numero)
{
    if(numero % 2 == 0){
     printf(“%d é par”,numero);
    }
}
void pulaLinha()
{
    printf(“\n”);
}
void imprimeLinha()
{
    printf(“**********”);
}
```

Funções com retorno

- Funções com retorno podem receber ou não dados (parâmetros) para desempenhar a sua tarefa.

- Quando chamados pelo programa principal, desviam o fluxo de execução do programa para o início do seu bloco de código e no final retornam para o fluxo principal do programa na instrução seguinte a que chamou a função retornando sempre um valor.

- Sempre retornam um resultado
  - **return();**

Sintaxe Funções com retorno

```
tipo nome_function(parametros opcionais)
{
 //... bloco de código
    return(valor);
}
```

- Parâmetros opcionais
  - Valores que poderão ser enviados para a função que poderá ser utilizado na sua execução.
  - São “variáveis” e pode ser declarados com qualquer tipo(int, char, float,...).
  - Não é obrigatório a declaração de parâmetros nas funções.
- Tipo
  - Tipo de dado de retorno (int, char, float,...)

Return

- A palavra reservada **return** é utilizada para retornar o resultado de uma função.

- Ex:

```
return(3); //A função retorna o valor inteiro 3.
return(a); //A função retorna o valor armazenado na variável a
```

Funções sem parâmetros

```
//Função que retorna o ano corrente.
int anoCorrente()
{
    return(2019);
}

//Função que retorna o valor da constante PI
float PI()
{
    return(3.1416);
}
```

Função com parâmetros

```
//Função que efetua a divisão e retorna o resultado.
float divisao(float dividendo,float divisor)
{
    if(divisor == 0){
     return(0);
    }
    else{
     return(dividendo/divisor);
    }
}
```

* No momento em que o comando return é executado, a linha de execução é interrompida.

Protótipo de Funções com retorno

```
#include<stdio.h>
//Prototipo da funcao
int anoCorrente();

main(){
    int ano = anoCorrente();
    printf(“%d”,ano);
}
//Implementação da funcao
int anoCorrente()
{
    return(2019);
}
```

EXEMPLO 1

```
int max(int num1, int num2);

int main () {
   int a = 100;
   int b = 200;
   int ret;
   ret = max(a, b);
   printf( "Maior valor é : %d\n",
ret );
   return 0;
}
```

```
int max(int num1, int num2) {

   int result;

   if (num1 > num2)
    result = num1;
   else
    result = num2;

   return result;
}
```

Exemplo 2

```
#include<stdio.h>
float divide(float dividendo, float divisor);

/*Função que efetua uma operação divisão e retorna o resultado da mesma. A função retorna 0 caso a divisão seja “por zero”*/
float divide(float dividendo, float divisor)
{
    if(divisor == 0){
       return(0);
    } else{
       return(dividendo/divisor);
    }
}
```

Exercícios

1. Acrescente ao programa anterior as funções:
   Soma
   Subtração
   Multiplicação

2. Escreva uma função que recebe como parâmetro um numero e imprime a tabuada da soma desse número.
   (Utilize as funções que você criou na questão 1 para realizar as operações básicas)

3. Com a mesma lógica da questão 2, escreva as funções abaixo:
   tabuadaSubtracao(int numero)
   tabuadaMultiplicação(int numero)

EXERCÍCIOS

4. Através das funções criadas nas questões anteriores, faça o programa abaixo funcionar.

```
main(){
    int numero = 5;

    tabuadaSoma(numero);
    pulaLinha();
    tabuadaSubtracao(numero);
    pulaLinha();
    tabuadaMultiplicacao(numero);
}
```

Referências

Consultar ementário.