CURSO SUPERIOR DE ADS

Subprogramação

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Aula de Hoje

- Subprogramas ou funções.
- Parâmetro, retorno e argumento.
- Funções sem retorno e sem parâmetros.
- Funções sem retorno e com parâmetros.
- Funções sem retorno e com parâmetros.
- Passagem de parâmetros por valor e por referência.
- Funções com ponteiros.
- Exercícios.

Introdução

- Com o conteúdo visto até agora, percebemos que é possível construir programas de pequeno porte de forma que o código fique relativamente bem organizado.
- Entretanto em projetos maiores, os códigos acabam naturalmente se tornando confusos, mesmo fazendo todos os cuidados organizacionais.
- Por este motivo é importante dominar o conceito de subprogramação, implementado em linguagem C por meio de funções.

Introdução

- O programa principal poderá ser divido em programas menores, onde cada parte se preocupa somente em resolver um problema específico.
- A programação estruturada visa à decomposição do algoritmo total em **módulos**, chamados de subprogramas.

Conceitos Básicos

- Subprogramas são programas menores que resolvem determinado problema específico. Sua implementação é dada por funções, que agem de maneira similar às funções matemáticas.
- Por exemplo, considerando $f(x) = x^2$. sua saída vai depender do valor de x escolhido (entrada).
- Para qualquer valor de x colocado em seu domínio, uma saída correspondente é esperada no valor de $f(x)$.

Um exemplo de função é a printf

- Desde nosso primeiro programa utilizamos printf.
- Ela recebe como entrada uma string e uma série de variáveis, imprimindo no console seu resultado.
- Ela faz parte de um conjunto de funções pré-definidas disponibilizadas pela linguagem. Quando desejamos utilizá-las, podemos incorporá-las no programa pela diretiva **#include**.

Vantagens do uso de funções

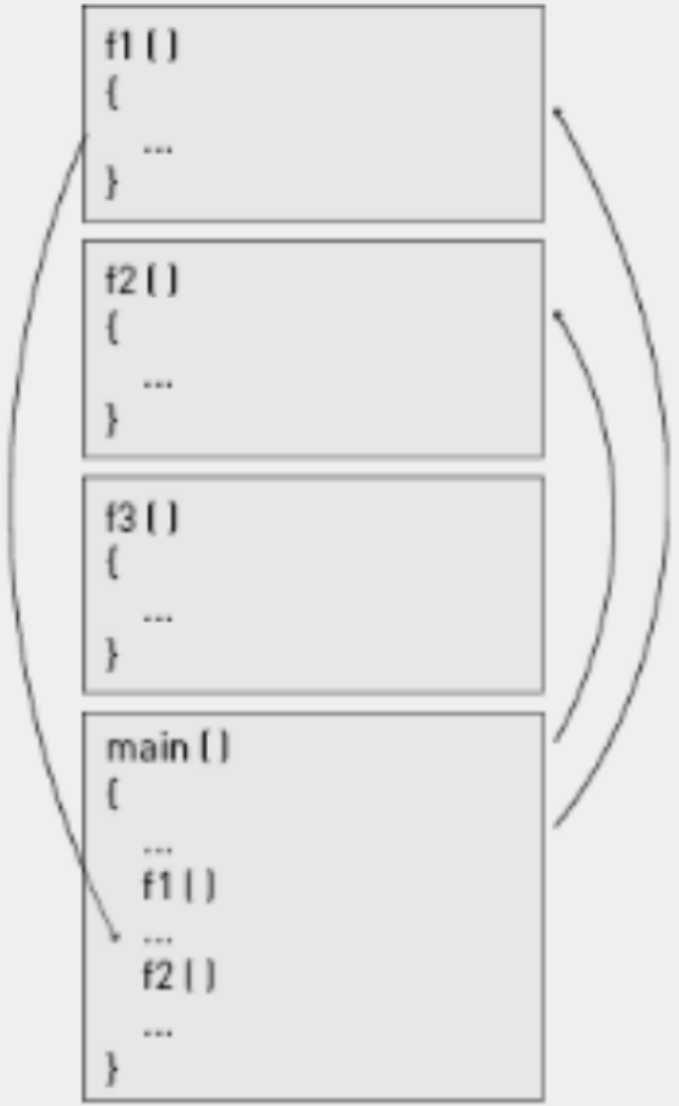
- **Organização e legibilidade** – cada parte do código tem seu próprio subcódigo.
- **Divisão de problemas** – cada função fica responsável por um subproblema.
- **Reaproveitamento** – a função pode ser reutilizada em outro contexto.
- **Manutenibilidade** – facilita o encontro e correção de erros, bem com alterações que serão refletidas em todo código.

É importante ter em mente que:

- Ao projetar as funções do programa, seja garantido que elas **resolvam apenas um problema específico**.
- Todo programa em C deve conter a função **main**, sendo que a sua execução sempre começará por ela. Todas as funções devem ser especificadas antes da função principal.
- A partir do momento em que uma função é especificada, sua chamada pode ocorrer quantas vezes o programador quiser dentro de qualquer outra função do programa.

Fluxo de execução sem funções

Fluxo de execução com funções



Parâmetros, Retorno e Argumento

- Funções podem possuir **parâmetros** e **retorno**.
- **Parâmetros** são valores de entrada recebidos por funções ao serem chamadas.
- **Retorno** é o valor de saída da função ao término de sua execução.
- Quando uma função é chamada, existe uma nomenclatura específica para o nome dos dados que serão atribuídos aos parâmetros, chamados de **argumentos**.

- Funções podem existir em 3 formatos: **protótipo**, **chamada** e **implementação** (de declaração).

Protótipo, Implementação e Chamada

```
tipo nome(tipo parametro1, ... tipo parametroN);  // prototipo

tipo nome(tipo parametro1, ... tipo parametroN)  // implementacao

{

...

return valor;

}

nome(argumento1 , ... argumentoN); // chamada
```

Protótipo

- Se situa no início do programa, antes da implementação das funções e contém apenas a sua definição, **servindo como assinatura, sem a sua implementação propriamente dita**.
- Serve para indicar ao programa quantas e quais funções ele terá, para que as funções possam chamar umas às outras de qualquer ponto do código.
- Caso ele não seja colocado, o acesso às funções só ocorrerá se a chamada estiver depois da sua implementação. A sintaxe do protótipo é formada pelo tipo de dado de retorno da função, nome e argumentos entre parênteses.

Implementação

- Expande o protótipo com o **corpo da função**, contendo todo seu código.
- Para isso, ao invés de encerrar a função com **;** (ponto e vírgula), abre-se um bloco similar aos comandos condicionais e iterativos, que deve conter o código propriamente dito.
- Ao final da função, caso ela contenha retorno, deve haver o comando **return** seguido do valor a ser retornado.
- O valor pode ser literal, variável ou expressão, desde que seja do tipo de retorno especificado na função.

Chamada

- Pode ocorrer dentro da função main ou de qualquer outra função do programa. Quando uma função é chamada, seu fluxo de execução é transferido para a função e após terminar, retorna para o ponto logo após de onde foi chamado.
- Pode haver quantas chamadas o programador desejar de cada função, inclusive nenhuma.

Exemplo

```
1 /*
2 Comentario inicial descrevendo o programa
3 */
4
5 #include <stdio.h>
6 // outras bibliotecas necessarias
7
8 // constantes com define
9
10 // prototipos de funcoes
11
12 // implementacao de funcoes
13
```

```
14 int main ( ) // programa principal
15 {
16     // declaracoes de variaveis
17
18     // instrucoes
19
20     return 0; // encerramento do programa
21 }
```

Funções sem retorno e sem parâmetro

```
void nome ( ); // prototipo

void nome ( ) // implementacao

{

...

}

nome( ); // chamada
```

Funções sem retorno e sem parâmetro

```c
#include <stdio.h>

// prototipos de funcoes
void imprimeSeparador ( );

// declaracoes de funcoes
void imprimeSeparador ( )
{
    printf("\n- - - - - \n");
}
```

```c
int main( )
{
    char mensagem[30];

    // chamada da funcao
    imprimeSeparador();

    printf("Digite uma mensagem:");
    fgets(nome, 30, stdin);

    // chamada da funcao
    imprimeSeparador();

    printf("Voce digitou: %s", mensagem);

    // chamada da funcao
    imprimeSeparador();

    return 0;
}
```

Funções sem retorno e com parâmetros

```
void nome(tipo parametro1, ... tipo parametroN); // prototipo

void nome(tipo parametro1, ... tipo parametroN) // implementacao

{

...

}

nome(argumento1, ... argumentoN); // chamada
```

Funções sem retorno e com parâmetros

```c
#include <stdio.h>

// prototipos de funcoes
void imprimeSeparador(int quantidade);

// declaracoes de funcoes
void imprimeSeparador(int quantidade)
{
    int i;

    printf("\n");

    for(i =0; i <quantidade; i++)
    printf("-");

    printf("\n");
    }
```

```c
int main ()
{
    char mensagem[30];

    // chamada da funcao
    imprimeSeparador(10);

    printf("Digite uma mensagem:");
    fgets(mensagem, 30, stdin);

    // chamada da funcao
    imprimeSeparador(5);

    printf("Voce digitou: %s", mensagem);

    // chamada da funcao
    imprimeSeparador(20);

    return 0;
}
```

Funções com retorno

```
tipo nome(tipo parametro1, ... tipo parametroN);  // prototipo

tipo nome(tipo parametro1, ... tipo parametroN)  // implementacao

{

...

return  valor;

}

valor = nome(argumento1, ... argumentoN);  // chamada
```

Funções com retorno

```c
#include <stdio.h>

// prototipos de funcoes
float soma(float valor1, float valor2);
void imprimeMaior(float valor1, float valor2);


// declaracoes de funcoes
float soma(float valor1, float valor2)
{
    return valor1 + valor2;
}
```

```c
void imprimeMaior(float valor1, float valor2)
{
    if(valor1 > valor2)
      printf("Valor %f maior que %f .\n", valor1, valor2);
    else if(valor2 > valor1)
      printf("Valor %f maior que %f .\n", valor2, valor1);
    else
      printf("Valores iguais! \ n");
}
```

```c
int main ()
{
    float v1, v2, res;

    printf("Digite o primeiro valor:");
    scanf("%f", &v1);

    printf("Digite o segundo valor:");
    scanf("%f", &v2);

    res = soma (v1, v2);

    printf("Resultado da soma: %f\n", res);

    imprimeMaior(v1, v2);

    return 0;
}
```

Escopo de variáveis

Escopo de Variáveis

- Quando variáveis estão localizadas dentro de um escopo em específico, como em alguma função ou até mesmo na função main, elas são chamadas de locais.
- Variáveis locais só podem ser referenciadas por comandos que estão dentro do bloco em que foram declaradas e existem apenas quando o seu respectivo bloco de código está sendo executado.
- Variáveis globais são declaradas no início do programa, fora de qualquer função, permitindo que qualquer parte do código possa utilizá-las.

Exemplo

```c
#include <stdio.h>

int cont = 5; // variavel global

void funcao1 ()
{
    int cont; // cont local da funcao 1

    cont = 15;

     printf("\n cont funcao 1 = %d\n", cont);
}
```

```c
int main ()
{
    int cont; // cont local do programa principal

    cont = 10;

    printf("\n cont main = %d \n", cont);

    funcao1();

    printf("\n cont main apos funcao 1 = %d \n", cont);

    return 0;
}
```

```
cont main = 10

cont funcao 1 = 15

cont main apos funcao1 = 10
```

Atenção

- Note que a variável global **cont**, no exemplo do slide anterior, não será modificada nem impressa, pois **variáveis locais que possuem o mesmo nome das globais em escopo próprio, possuem prioridade**.

- Deve-se evitar ao máximo o uso de variáveis globais, pois elas **prejudicam a legibilidade do código**.
- Variáveis só devem se comunicar com o programa principal com parâmetros e retorno de funções

Passagem de parâmetros e funções com ponteiros

Existem duas formas principais de passagens de parâmetros de funções na linguagem C: por valor ou por referência

Passagem por valor

- É o tipo padrão, ocorrendo normalmente ao pedir um parâmetro que não seja do tipo ponteiro.
- Quando os argumentos são passados como parâmetros para as funções durante sua chamada, **uma cópia desses valores é feita para variáveis locais com os mesmos nomes dos parâmetros**, agindo como se fossem variáveis locais da função.
- Por este motivo, qualquer alteração feita pela função nesses valores não será refletida fora do escopo da função.

Exemplo

```
Na funcao main:
a=5; b=10
&a=0x7f1b88; &b=0x7f1b8c
Na funcao troca:
x=10; y=5
&x=0x7f1b5c; &y=0x7f1b58
Na funcao main:
a=5; b=10
&a=0x7f1b88; &b=0x7f1b8c
```

```c
#include <stdio.h>

void trocaIncorreta(int x , int y);

void trocaIncorreta(int x , int y)
{
    int temp;

    temp = x;
     x = y;
     y = temp;

     printf("Na funcao troca: \ n");
     printf("x=%d; y=%d\n", x , y);
     printf("&x=%p; &y=%p\n",&x,&y);
}
```

```c
int main ()
{
    int a=5 , b=10;

    printf("Na funcao main: \ n");
    printf("a=%d; b=%d\n", a , b);
    printf("&a=%p; &b=%p\n",&a,&b);

    trocaIncorreta(a, b);

    printf("Na funcao main: \ n");
    printf("a=%d; b=%d\n", a , b);
    printf("&a=%p; &b=%p\n",&a,&b);

    return 0;
}
```

Passagem por referência

- Em vez de passar uma cópia do valor, é esperado como parâmetro o endereço de onde está situada a variável.
- Como o endereço será recebido, é possível alterar diretamente o campo valor na memória da variável apontada pelo endereço.
- Assim a passagem por referência ocorre com o uso de ponteiros, uma vez que os parâmetros precisam armazenar diretamente os endereços de memória das variáveis.
- Os argumentos, por sua vez, agora precisam necessariamente ser endereços de memória.

Exemplo

```
Na funcao main:
a=5; b=10
&a=0x7f1b88; &b=0x7f1b8c
Na funcao troca:
*x=10; *y=5 x=0x7f1b88; y=0x7f1b8c
&x=0x7f1b68; &y=0x7f1b60
Na funcao main:
a=10; b=5
&a=0x7f1b88; &b=0x7f1b8c
```

```c
#include <stdio.h>

void trocaCorreta(int *x , int *y);

void trocaCorreta(int *x , int *y)
{
    int temp;

    temp = *x;
     *x = *y;
     *y = temp;

     printf("Na funcao troca: \ n");
     printf("*x=%d; *y=%d\n", *x , *y);
     printf("x=%p; y=%p\n", x , y);
     printf("&x=%p; &y=%p\n",&x,&y);
}
```

```c
int main ()
{
    int a=5 , b=10;

    printf("Na funcao main: \n");
    printf("a=%d; b=%d\n", a, b);
    printf("&a=%p; &b=%p\n",&a,&b);

    trocaCorreta(&a,&b);

    printf("Na funcao main: \n");
    printf("a=%d; b=%d\n", a, b);
    printf("&a=%p; &b=%p\n",&a,&b);

    return 0;
}
```

Resumindo...

- A passagem de parâmetros por valor pode ser utilizada quando não for necessário alterar os valores dos parâmetros dentro da função.
- Nos casos que houver necessidade de alterar esses valores, deve-se utilizar a passagem de parâmetros por referência.

- O retorno da função também oferece uma alternativa viável para comunicação de variáveis da função com as demais partes do código, mas possibilita que apenas um valor seja retornado.

Outro ponto importante

- É a passagem de vetores como parâmetros.
- Como o vetor é um ponteiro para a primeira posição da memória, obrigatoriamente deve ser passado um ponteiro como parâmetro.
- Dessa forma, o argumento deve ser o endereço do primeiro elemento do vetor.
- Todas as operações e modificações que forem feitas no vetor dentro da função são necessariamente refletidas do lado de fora, não havendo necessidade de retornar um vetor, por exemplo.

Exercícios

Exercício 1

- Faça um programa que calcule o índice de massa corporal (IMC) de um usuário a partir da sua massa e sua altura.
- O IMC deve ser calculado por uma função específica para tal, que recebe os dois argumentos e retorna no valor.
- Peça os valores ao usuário e indique o número resultante do IMC no programa principal.

Exercício 2

- Faça um programa que permita ao usuário ao selecionar uma das quatro operações básicas (adição, subtração, multiplicação e divisão) para que o cálculo seja realizado.
- Cada uma das operações deve ser construída em uma função própria e devem ser chamadas na função principal com um menu próprio.

Exercício 3

- Faça um programa que desenvolva e teste uma função que calcule a área e o perímetro de um retângulo a partir dos valores de sua base e altura.
- Os valores da área e perímetro devem obedecer à passagem de parâmetros por referência e devem ser calculados na própria função.
- A função deve retornar um inteiro com o valor 0 se a base ou a altura forem negativos ou 1 caso contrário.
- Teste a função no programa principal.

Exercício 4

- Faça um programa que codifique uma função que recebe um vetor de números inteiros e seu número de elementos ocupados e retorne o maior número entre eles.
- Teste a função no menu principal.

Exercício 5

- Faça um programa que some o valor de dois vetores do tipo float com o mesmo número de elementos.
- A soma de vetores deve ser implementada por meio de uma função que deve armazenar seu resultado no segundo vetor passado como parâmetro, modificando seus valores também fora da função.
- A função não deve possuir retorno. Imprima o conteúdo dos vetores após a chamada da função no programa principal.