Universidade Positivo Disciplina: Programação de Computadores Professor: Marcelo Fassina	aula **05**

Conteúdo
- Modularização.

Competências
- (6) Aplicar os conceitos de modularização na organização do código de um projeto de software.

1 Modularização

Muitas vezes um problema grande pode ser resolvido mais facilmente se for dividido em pequenas partes. Tratar partes menores e mais simples em separado é muito mais fácil do que tratar o problema grande e difícil de uma vez. É comum em programação decompor programas complexos em programas menores e depois juntá-los para compor o programa final. Essa técnica de programação é denominada **programação modular.** A modularização, em linguagem C começa através do uso adequado de **funções.** Funções são algumas das formas usadas para agruparmos código de forma organizada e modularizada. Tipicamente, usamos funções para realizar tarefas que se repetem várias vezes na execução de um mesmo programa.

1.1 Funções

Nas linguagens de programação também temos funções e elas são bem parecidas com as funções da matemática. Uma função é um tipo especial de sub-rotina que retorna um resultado de volta ao "ponto" onde foi chamada.

A sintaxe de uma função é mostrada a seguir.

```
tipo_da_função nome_função (lista_de_parâmetos)
{
   /* corpo_da_função */
}
```

No exemplo a seguir, temos uma função que retorna a soma de dois números.

```
#include <stdio.h>

/* Declara a função */
int soma (int num1, int num2){

       int s;

       s = num1 + num2;

       return (s);
}

int main(){

       int resultado;

       /* Usa a função soma */
       resultado = soma (3 , 5);

       /* Mostra o resultado */
       printf ("Resultado: %d", resultado);

       return 0;
}
```

1.2 Localização das funções no programa fonte

A princípio podemos tomar com regra a seguinte afirmativa toda função deve ser declarada antes de ser usada. Alguns programadores preferem que o início do programa seja a primeira parte de seu programa. Para isto a linguagem C permite que se declare uma função, antes de defini-la. Esta declaração é feita através do **protótipo** da função. O protótipo da função nada mais é do que o trecho de código que especifica o nome e os parâmetros da função.

```
#include <stdio.h>

/* Protótipo da função soma */
int soma (int num1, int num2);

int main(){

	int resultado;

	/* Usa a função soma */
	resultado = soma (3 , 5);

	/* Mostra o resultado */
	printf ("Resultado: %d", resultado);

	return 0;
}

int soma (int num1, int num2)
{
	int s;

	s = num1 + num2;

	return (s);
}
```

1.3 Escopo das variáveis

Uma variável definida dentro de uma função só pode ser usada dentro desta função (não pode ser usada fora do escopo da função). Portanto, podemos definir variáveis com o mesmo nome em funções diferentes sem nenhum problema. Dizemos que as variáveis que são definidas dentro de uma função são locais a essa função, ou seja, elas só existem enquanto a função está sendo executada (elas passam a existir quando ocorre a entrada da função e são destruídas ao sair).

```
#include <stdio.h>

/* Variáveis globais */
int s, a;

int soma (int i){

	int v;
	v = s + i;
	s = v;

	return (v);
}

int main(){

	s = 1;
	a = soma (5);
	printf ("A: %d, S: %d\n", a, s);

	return 0;
}
```

1.4 Procedimentos

Chamamos de **procedimentos** as funções que não retornam nenhum valor (void). No exemplo a seguir, temos um procedimento apresenta a uma mensagem simples na tela.

```
#include <stdio.h>

void mensagem (){

	printf("Um programa em linguagem C.");

	return;
}

int main(){

	mensagem();

	return 0;
}
```

No próximo exemplo, a mensagem "Um programa em Linguagem C" é apresentada o número de vezes solicitado pelo usuário.

```
#include <stdio.h>

void mensagem (int num){

	int i;

	for(i = 1; i <= num; i++){
		printf("Um programa em linguagem C.\n");
	}

	return;
}

int main(){

	int num;

	printf("Informe o número de vezes de exibição da mensagem: ");
	scanf("%d", &num);

	mensagem(num);

	return 0;
}
```

1.5 Passagem de parâmetros

A passagem de parâmetros permite que uma função receba um valor vindo da função principal ou de outra função. Ou seja, permite a entrada de informações que serão trabalhadas na função, sendo que estas informações não são geradas na própria função.

Há dois modelos de passagem de parâmetros: por valor e por referência.

- **Passagem de parâmetros por valor**

A variável original é passada como parâmetro para uma função, mas quando o parâmetro é manipulado na função, o valor da variável original não é alterado. É como se o parâmetro fosse uma cópia da variável original e, desta forma, qualquer alteração afeta unicamente o parâmetro, sem afetar a variável original.

Exemplo: nesse exemplo, é solicitado um número inteiro e uma função apresenta a sequência do número informado com decremento de 1. Ao final da execução, é mostrado o valor de número informado pelo usuário.

```
#include <stdio.h>

void mostraDec(int num){
	printf("Sequência decrescente de números:\n");

	while(num > 0){
		printf("%d\n", num);
		num -= 1;
	}

	return;
}

int main(){
	int num;

	printf("Informe número: ");
	scanf("%d", &num);

	mostraDec(num);

	printf("Número informado pelo usuário: %d", num);

	return 0;
}
```

- **Passagem de parâmetros por referência**

A variável original é passada como parâmetro para uma função e, quando o parâmetro é manipulado na função, o valor da variável original também é afetado. É como se o parâmetro fosse a própria variável original e, desta forma, qualquer alteração no parâmetro afeta a variável original. Para sinalizar a passagem de parâmetros por referência, deve-se utilizar * na frente o parâmetro determinado. Uma observação importante é que o parâmetro deve ser passado para a função juntamente com o endereço (&).

```
#include <stdio.h>

void mostraDec(int *num){
	printf("Sequência decrescente de números:\n");

	while(*num > 0){
		printf("%d\n", *num);
		*num -= 1;
	}

	return;
}

int main(){
	int num;

	printf("Informe número: ");
	scanf("%d", &num);

	mostraDec(&num);

	printf("Número informado pelo usuário: %d", num);

	return 0;
}
```

1.6 Funçoes recursivas

Uma função recursiva é uma função que chama a si mesma. Elas podem ser usadas para poder processar uma determinada operação e geralmente há condições internas para que a recursividades sejam aplicadas, uma vez que sem condições, ela chamaria a si mesmo eternamente, causando o que chamamos de loop infinito.

Um número fatorial pode ser calculado pelo fatorial de seus números anteriores. Por exemplo, o fatorial de 5 é:

Fatorial (5) = 5 x fatorial (4)
Fatorial (4) = 4 x fatorial (3)
Fatorial (3) = 3 x fatorial (2)
Fatorial (2) = 2 x fatorial (1)
Fatorial (1) = 1 x fatorial (0)
Fatorial (0) = 1

Exemplo de cálculo de fatorial usando uma função recursiva.

```
#include<stdio.h>

int fatorial (int num){

	if (num >= 1){
		return num * fatorial(num - 1);
	}
	else{
		return (1);
	}
}

int main(){

	int num;

	printf("Informe um número positivo: ");
	scanf("%d", &num);

	printf("O fatorial de %d é %d.", num, fatorial(num));

	return 0;
}
```

Nessa função, supondo que o número informado pelo usuário informou o número 5, inicialmente a função "fatorial" é chamada e recebe 5 como parâmetro. Em seguida, o número 5 é passado para a função "fatorial" a partir da mesma função (chamada recursiva). Em cada chamada recursiva, o valor do parâmetro "num" é diminuído em 1.

Quando o valor de "num" for menor que 1, não há chamada recursiva e o fatorial é retornado finalmente para a função principal.

EXERCÍCIOS

1. Escreva um programa que escreva na tela a seguinte saída, escrevendo uma linha com 20 asteriscos através de um laço for:

   ```
   ********************
   Números entre 1 e 5
   ********************
   1
   2
   3
   4
   5
   ********************
   ```

2. Reescreva o programa do exercício anterior, usando uma função de nome "linha" para escrever a linha com os asteriscos.

3. Modifique a função "linha" do exercício anterior de tal forma que a quantidade de asteriscos seja um argumento da mesma.

4. Escreva uma função que retorne o valor absoluto de um número.

5. Escreva uma função que receba dois números inteiros e retorne a soma dos números existentes entre eles.

6. Construa uma função que retorne o maior valor entre dois números inteiros.

7. Construa uma função que retorne o maior valor entre três números inteiros.

8. Escreva uma função que receba uma letra e 3 notas de um aluno. Se a letra for A, a função deve retornar a média aritmética das notas do aluno e se a letra for P, a função deve retornar a sua média ponderada com os pesos 5, 3 e 2 respectivamente.

9. Escreva uma função que receba um símbolo e dois números. O símbolo representa a operação que se deseja efetuar com os números (soma, subtração, multiplicação ou divisão).

10. Escreva uma função que receba como parâmetro um número inteiro e gere como saída n linhas com asteriscos, conforme o exemplo abaixo (para n = 5):

    ```
    *
    **
    ***
    ****
    *****
    ```

11. Escreva uma função recursiva que retorne a soma dos números naturais para um número inteiro positivo passado como parâmetro. Exemplo: para num = 6, retorne 21 (1 + 2 + 3 + 4 + 5 + 6).

12. Escreva uma função recursiva que, dados x e y como parâmetros, calcule a potência de x (x^y).