Algoritmos - TADS

Algoritmos – Funções em C

Professor: Victor Hugo L Lopes

Agenda

- Definições
- Chamando uma Função
- Funções simples
- Protótipo de funções
- O tipo de uma função
- Parâmetros de funções
- Passagem de valores para funções

Programação em Linguagem C

- Definição:

Função é um conjunto de instruções agrupadas e nomeadas, desenhada para cumprir uma tarefa particular.

Funções dividem grandes tarefas de computação em tarefas menores, e permitem às pessoas trabalharem sobre o que outras já fizeram, em vez de partir do nada.

Funções podem esconder detalhes de operação de partes do programa que não precisam ser conhecidas pelos demais utilizadores.

Ex.: printf();

Programação em Linguagem C

- Chamando uma função:

Chamar uma função pode ser comparado à contratação de uma pessoa para a execução de um trabalho específico. As vezes a interação com essa pessoa é bem simples, outras vezes mais complexa.

Chamar uma função é solicitar que o programa desvie o controle e passe a executar as instruções da função, e que, ao término desta, volte o controle para a posição seguinte à chamada.

Ex.:

```
Int n;
Printf(“digite o valor:”);
Scanf(“%d”,&n);
Printf(“\nSeu valor foi:%d”,n);
```

Programação em Linguagem C

- Funções simples:

Um programa pode conter uma ou mais funções, das quais uma deve ser a main().

A execução do programa começa em main(), e quando o controle do programa encontra uma instrução que inclui o nome de uma função, esta é chamada.

Portanto, chamar uma função é transferir o controle da execução para um bloco de comandos no programa, que possui um nome definido.

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

Float celsius(float); //protótipo da função

Int main(){
        Float c, f;
        Printf(“Digite a temperatura em Fahrenheit: ”);
        Scanf(“%f”,&f);

        C = celsius(f); //chama a função

        Printf(“Celsius =  %.2f\n”,c):
        System(“PAUSE”);
        Return 0;

}

//=------
Float celsius(float fah){
        Float c;
        C = (fah  - 32.0) * 5.0/9.0;
        Return c;
}
```

Programação em Linguagem C

- Protótipo de funções:

Uma função não pode ser chamada sem antes ter sido declarada.

Esta declaração prévia é chamada de protótipo da função, que é uma instrução colocada, geralmente, no início do programa que estabelece o tipo da função e os argumentos que ela recebe.

O protótipo da função permite ao compilador verificar a sintaxe da chamada à função.

Programação em Linguagem C

- Protótipo de funções:

Protótipos de funções podem ser locais ou externos.

Os externos são como no exemplo da conversão de temperaturas, e estará visível à todo o programa. Já um protótipo local é aquele criado dentro de uma função, e que só será visível dentro dela:

```
Int main(){
      float celsius(float); //protótipo local
      float c, f;
      ...
}
Float celsius(float fah){
...
```

Programação em Linguagem C

- O tipo de uma função:

O tipo de uma função é definido pelo tipo de valor que ela retorna por meio do comando return.

Uma função pode ter qualquer tipo de dados que podemos utilizar em variáveis.

Caso uma função não retorne nada, ela é do tipo void.

Programação em Linguagem C

- O comando return:

O comando return termina a execução da função e retorna o controle para a instrução seguinte do código de chamada.

O comando return, caso tenha um valor à sua frente, devolve ao requisitante um dado que será convertido no tipo declarada na função.

Sintaxe:

Return;

Return expressão;

Return(expressão);

Programação em Linguagem C

- O comando return:

ex.:

```
Float celsius(float fah){
        Return (fah – 32.0) * 5.0/9.0;
}
```

Programação em Linguagem C

- O comando return:

Funções do tipo void podem utilizar o comando return desacompanhado de expressão.

Neste caso o return serve somente para encerrar a função.

Em funções void, o comando return não é obrigatório, e na sua falta, a função será encerrada ao se encontrar o fechamento das chaves.

Programação em Linguagem C

- O comando return:

Funções que retornam valores, ou seja, funções que não são do tipo void, devem ser chamadas esperando-se o seu valor, atribuindo este valor à uma variável do mesmo tipo, ou dentro de uma expressão:

C = celsius(f); //atribuindo o retorn da função à variável c

Printf("A temperatura em celsius: %.2f", celsius(f));

Programação em Linguagem C

Algumas funções podem ter vários comandos return, mas somente um poderá ser executado.

```
int classifica(int a, int b){
    if (a > b)
        return 1;
    else
        return 0;
}
int main(int argc, char** argv) {
    int n1, n2;
    scanf("%d%d",&n1,&n2);
    if (classifica(n1,n2))
        printf("%d maior que %d",n1, n2);
    else
        printf("%d menor ou igua a %d",n1,n2);
    return (EXIT_SUCCESS);
}
```

Programação em Linguagem C

- O comando return:

Uma limitação do comando return:

Enquanto vários valores podem ser passados à uma função, o return somente pode retornar um único valor.

Programação em Linguagem C

- Parâmetros de funções:

As variáveis que receberão as informações enviadas à uma função são chamadas de parâmetros.

A função deve declarar essas variáveis entre parênteses, no cabeçalho de sua definição.

ex.:

```
int classifica(int a, int b){
    ...
}
```

Programação em Linguagem C

- Passagem de argumentos por valor:

No exemplo da conversão de temperaturas, há na função uma nova variável, do mesmo tipo, que vai receber o valor passado pela função main().

Sua declaração indica que o valor enviado na chamada da função será armazenado na variável float fah.

A função copia o valor enviado pela main() na variável fah, criada por ela. Isso se chama passagem de argumento por valor.

Programação em Linguagem C

- **Exercitando**

Altere o programa da calculadora sequencial para que cada operação seja calculada em uma função, lembrando-se dos conceitos já aprendidos de escopo local e global.

Programação em Linguagem C

Para melhoria do entendimento dos conceitos de funções já abordados, vejamos o desenvolvimento de um programa para ler 3 números e devolver seu valor absoluto:

Passos:

- **Criar um novo projeto chamado valorAbsoluto;**
- **Na função principal do programa, crie as variáveis de entrada e leia os valores digitados pelo usuário;**

Programação em Linguagem C

Para melhoria do entendimento dos conceitos de funções já abordados, vejamos o desenvolvimento de um programa para ler 3 números e devolver seu valor absoluto:

Passos:

- **Crie um protótipo de função, do mesmo tipo de dados das variáveis que você criou, isto é, se suas variáveis de entrada são do tipo float, a função deve ser do tipo float, pois irá retornar o mesmo valor digitado pelo usuário, mas sem sinal. Chame-a de absoluto. Lembre-se que o protótipo deve ser declarado fora da função principal, e antes desta começar.**
- **Note que a função deverá receber o valor a ser testado, para poder retornar o valor absoluto. Então, altere o protótipo para que ele receba um parâmetro do mesmo tipo dos dados de entrada. Lembre-se da sintaxe do protótipo:**

tipo nomeDaFuncao(tipoDeDadosDoParametro);

Programação em Linguagem C

Para melhoria do entendimento dos conceitos de funções já abordados, vejamos o desenvolvimento de um programa para ler 3 números e devolver seu valor absoluto:

Passos:

- **Agora que temos o protótipo da função, volte à função principal do programa, e construa a chamada à função, que vai enviar um dos valores à função, como parâmetro, e vai atribuir o retorno à uma variável, no formato:**

```
variavel = nomeDaFuncao(parametro);
```

- **Repita esta chamada mais duas vezes, sendo que teremos 3 chamadas à função, uma para cada entrada;**

- **Construa a saída das informações na tela;**

Programação em Linguagem C

Para melhoria do entendimento dos conceitos de funções já abordados, vejamos o desenvolvimento de um programa para ler 3 números e devolver seu valor absoluto:

Passos:

- **Agora, resta construir a função de fato. Ela deve respeitar o formato descrito no protótipo, sendo construída posterior ao protótipo, de preferência após a função principal do programa.**

Se o protótipo foi escrito como:

```
float absoluto(float);
```

A. A função deve ser escrita como:

```
B. Float absoluto(float n){
C.     //corpo da função
D. }
```

E. Por fim, construa o corpo da função.

Programação em Linguagem C

Função do tipo void: é uma função que não retorna nada;

```
void linha(int); //protótipo
int main() {
    linha(20);
    printf("\xDB Um Programa em C\xDB\n");
    Linha(20);
     system(“PAUSE”);
    Return 0;
}
void linha(int n){
    int j;
    for (j=1; j <= n; j++){
        printf("\xDB");
    }
    printf("\n");
}
```

Programação em Linguagem C

Omitindo o protótipo de uma função:

- **Se declararmos a função antes dela ser chamada, retiramos a obrigação do protótipo:**

Cenário tradicional:

- Includes;

- Protótipos;

- Função principal;

- Funções;

Cenário sem protótipos:

A. - Includes;

B. - Funções

C. - Função principal;

Obs.: Funções que retornam int não requerem declaração de protótipo, tampouco declaração prévia.

- **Se o tipo da função for omitido, será assumido que é int.**

Programação em Linguagem C

Escopo:

Agora com a visão de modularização proporcionado pelo uso de funções, faz-se necessário rever a questão de variáveis de escopo local e global.

Definições:

Uma variável de escopo local é a variável declarada dentro de uma função, que só existe dentro desta função.

Uma variável de escopo global é uma variável declarada fora de qualquer função, e que existirá para toda e qualquer função do programa.

Programação em Linguagem C

Escopo:

Por padrão, iremos utilizar o formato:

#includes

//variáveis globais

//protótipos

Função principal

funções

Programação em Linguagem C

Exercitando:

Construa um programa que, com auxílio de função(ões) receba o raio de uma circunferência, calcule e apresente a área e o perímetro da circunferência.

Considere π = 3.14159

- **E a fórmula da área como: $S = \pi * r^2$**
- **Do perímetro: $S = 2 * \pi * r$**

Programação em Linguagem C

Passando vários argumentos em uma função:

É permitido se passar diversos argumentos a uma função, utilizando-se a sintaxe:

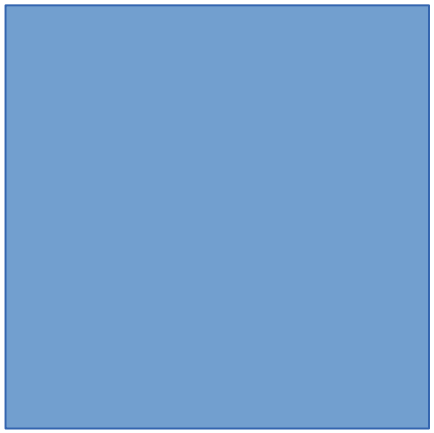
```
tipo nome(tipo, tipo, ... , tipo); //protótipo

tipo  nome(tipo n1, tipo n2, ... , tipo nm){
        //corpo da função
}
```

Programação em Linguagem C

Passando vários argumentos em uma função:

Exercitando:

Construa agora um programa que leia os dados necessários, calcule e apresente a área de um retângulo. Utilize funções.



Programação em Linguagem C

Passando vários argumentos em uma função:

Exercitando:

Construa uma programa que leia a altura e a base de um retângulo e desenhe o relativo retângulo na tela.

Utilize o caractere “\xDB” para desenho.

Programação em Linguagem C

Passando vários argumentos em uma função:

Exercitando:

Construa um programa que possibilite o cálculo geométrico de diferentes formas geométricas(círculos, retângulos/quadrados, trapézios, triângulos), que permita ao usuário selecionar qual cálculo ele deseja fazer(área, perímetro, volume) e em qual forma.

Programação em Linguagem C

Exercitando:

Continuando o software da calculadora sequencial, agora inclua opções para calcular a raiz quadrada e porcentagem.

Melhore as funções para corrigir possíveis erros, como a divisão por zero.

Por que não construir uma interface visual para ela utilizando funções para desenho da tela?

Divirta-se sem moderação!!!

Programação em Linguagem C

Exercitando:

O programa a seguir demonstra uma nova forma de utilizar a função scanf, onde podemos utilizar elementos textuais como máscara de entrada.

No programa, note que nos formatadores de entrada, utilizamos o sinal de dois pontos, para que as horas e minutos digitados, no seu formato padrão, sejam separados pelos dois pontos e guardados separadamente nas respectivas variáveis.

```
#include <stdio.h>
#include <stdlib.h>
int minutos(int, int); //protótipos
int main(int argc, char** argv) { //-------------------------------------------------------------
    //variaveis locais
    int mins1, mins2;
    int hora, min;
    printf("Digite a primeira hora (HH:MM): ");
    scanf("%d:%d",&hora,&min);
    mins1 = minutos(hora, min);
    printf("Digite a segunda hora (HH:MM): ");
    scanf("%d:%d",&hora,&min);
    mins2 = minutos(hora, min);
    printf("A diferença e' de %d minutos.",mins2-mins1);

    return (EXIT_SUCCESS);
}//-----------------------------------------------------------------------------------------------
int minutos(int h, int m){
    return (h*60 + m);
}
```

Programação em Linguagem C

Chamadas a funções usadas como argumento de outras funções:

Uma chamada de uma função pode ser utilizada numa expressão qualquer, da mesma forma que utilizamos valores numéricos e variáveis.

ex.:

```
int funcao1(int x){
    return x * x;
}
...
int a;
a = funcao1(10);
a = funcao1(2) * 3; // o dobro de 2 vezes 3
```

note que funcao1() recebe um valor quando chamado e retorna o dobro deste valor, que é atribuído à a.

Programação em Linguagem C

Chamadas a funções usadas como argumento de outras funções:

Estendendo esta visão, podemos utilizar uma chama a uma função como argumento de uma outra função:

ex.:

```
Int dobro(int x){
        return x * x;
}
Int soma(int a, int b){
        return a + b;
}
...
Int total = soma (dobro(2),dobro(3));
```

total recebe a soma do dobro de 2 e do dobro de 3

Programação em Linguagem C

Chamadas a funções usadas como argumento de outras funções:

Por fim, importante também saber que a chamada de uma função pode ser diretamente utilizada em uma expressão aritmética ou lógica:

ex.:

```
Int calcula(int a, int b){
      return a * b;
}
...
If (calcula(10,2) > 20) ...
...
Int total = (2 + 3) * calcula(2,3);
```

Ou seja, se uma função retorna alto, entenda-a como algo que possua valor.

Programação em Linguagem C

Funções Recursivas

Uma função é dita recursiva quando dentro dela há uma chamada à ela mesma. Veja o exemplo da função de cálculo de fatorial:

```
long fatorial(int n){
   return ((n==0) ? (long)1 : (long)n*fatorial(n-1));
}
```

Note que dentro do corpo da função fatorial há uma chamada à ela mesma, até que o parâmetro n chegue à zero.

Neste caso, a função retorna à ela mesma antes de concluir sua execução.

Programação em Linguagem C

Funções Recursivas

Podemos enxergar uma função recursiva como uma chamada sucessiva à outra função diferente, mas com o mesmo corpo:

```
Long fatorial(int n){ // esta função será chamada com o valor 3
      return ((n ==0) ? (long)1 : 3 * f1(2));
}
Long f1(int n){
      return ((n ==0) ? (long)1 : 2 * f2(1));
}
Long f2(int n){
      return ((n ==0) ? (long)1 : 1 * f3(0));
}
Long f3(int n){
      return(1);
}
```

Programação em Linguagem C

Considerações sobre conflito de nomes de variáveis

Sempre que duas variáveis tiverem o mesmo nome mas diferentes endereços de memória elas não serão as mesmas variáveis.

Isto tem a ver com os escopos de funções, onde podemos ter duas variáveis com os mesmos nomes, desde que cada uma esteja em um escopo diferente.

Programação em Linguagem C

Considerações sobre conflito de nomes de variáveis

```
...
Int main(){
        int k = 5; //Primeira variável
        printf("Em main() - endereco de k = %p\n", &k);
        func1(k);
        printf("Em main() - o valor de k = %d\n ", k);
        system("PAUSE");
        return 0;
}
Void func1(int k){
        k = k + 2;
        printf("Em func1() - endereco de k = %p\n", &k);
        printf("Em func1() - o valor de k = %d\n", k);
}
```

Programação em Linguagem C

Considerações sobre conflito de nomes de variáveis

De acordo com a saída do programa, notamos facilmente que as duas variáveis não são as mesmas, pois o seu endereço é diferente.

Neste caso, não há conflito de variáveis em escopos diferentes.

Uma variável local só tem existência dentro do seu escopo, e se há uma outra variável de mesmo nome, global ou local em outro bloco, a variável local tem precedência sobre a que está declarada em outro bloco.

Programação em Linguagem C

Considerações sobre conflito de nomes de variáveis

De acordo com a saída do programa, notamos facilmente que as duas variáveis não são as mesmas, pois o seu endereço é diferente.

Neste caso, não há conflito de variáveis em escopos diferentes.

Uma variável local só tem existência dentro do seu escopo, e se há uma outra variável de mesmo nome, global ou local em outro bloco, a variável local tem precedência sobre a que está declarada em outro bloco.

Programação em Linguagem C

Considerações sobre conflito de nomes de variáveis

```
Int k = 5; //primeira variável
Int main(){
    int k = 10; //segunda variável
    printf("Em main() - o valor de k = %d\n ", k);
    func1( );
    system("PAUSE");
    return 0;
}
Void func1( ){
    printf("Em func1() - o valor de k = %d\n", k);
}
```

Programação em Linguagem C

Exercitando:

1. **Construa uma função que receba o ano como argumento e retorne 1 se for um ano bissexto e 0 se não for. Um ano é bissexto se for divisível por 4 e não for divisível por 100. Um ano também é bissexto se for divisível por 400.**

1620	1624	1628	1632	1636	1640	1644	1648	1652	1656	1660	1664	1668	1672	1676
1680	1684	1688	1692	1696	1704	1708	1712	1716	1720	1724	1728	1732	1736	1740
1744	1748	1752	1756	1760	1764	1768	1772	1776	1780	1784	1788	1792	1796	1804
1808	1812	1816	1820	1824	1828	1832	1836	1840	1844	1848	1852	1856	1860	1864
1868	1872	1876	1880	1884	1888	1892	1896	1904	1908	1912	1916	1920	1924	1928
1932	1936	1940	1944	1948	1952	1956	1960	1964	1968	1972	1976	1980	1984	1988
1992	1996	2000	2004	2008	2012	2016	2020							

Programação em Linguagem C

Exercitando:

2. Dia da semana: Escreva uma função que receba o dia, mês e ano e calcule o dia da semana que caiu esta data. Para isso, basta transformar a data gregoriana para juliana, e calcular o resto da divisão inteira por 7, que vai retornar um número entre 0 e 6, onde 0 é segunda-feira e 6 domingo.

A conversão de Gregoriana para Juliana

(1461*(a + 4800 + (m – 14)/12))/4

+ (367 * (m - 2 - 12 * ((m – 14)/12)))/12

- (3 * ((a+ 4900 + (m-14)/12)/100))/4+d – 32075;

3. Escreva um programa que leia o ano e escreva o calendário deste ano.