Técnicas de Programação II

Funções

Prof. José Luiz

Funções

- As funções permitem ao programador modularizar um programa.
- Todas as variáveis declaradas em definições de funções são **variáveis locais** – elas são conhecidas apenas na função onde são definidas.
- A maioria das funções tem uma lista de **parâmetros**.
- Os parâmetros fornecem os meios para a comunicação de informações entre funções. Os parâmetros de uma função também são variáveis locais.

Funções

Função	Descrição	Exemplo
sqrt(x)	Raiz quadrada de x	sqrt(900.0) = 30.0
exp(o)	Exponencial e°	exp(1.0) = 2.718282
log(x)	Logaritmo natural de x (base e)	log(2.7118282) = 1.0
log10(x)	Logaritmo de x (base 10)	log10(10.0) = 1.0
labs(x)	Valor absoluto de x	fabs(-9) = 9
ceil(x)	Arredonda x para menor inteiro maior que x	ceil(9.2) = 10.0
floor(x)	Arredonda x para maior inteiro menor que x	floor(9.2) = 9.0
pow(x,y)	X elevado à potência de y	pow(2,7) = 128
fmod(x,y)	Resto de x/y como número de ponto flutuante	fmod(13.657, 2.333) = 1.992
sin(x)	Seno trigonométrico de x (em radianos)	sin(0.0) = 0.0
cos(x)	Consseno trigonométrico de x (em radianos)	cos(0.0) = 1.0
tan(x)	Tangente trigonométrica de x (em radianos)	tan(0.0) = 0.0
Funções da biblioteca matemática usadas normalmente <math.h>		

Motivos para usar funções

- Dividir-para-conquistar torna o desenvolvimento do programa mais flexível.
- Capacidade de reutilização do código.
- Abstração
- Evita a repetição de código em um programa.

- Cada função deve se limitar a realizar uma tarefa simples e bem-definida, e o nome da função deve expressar efetivamente aquela tarefa. Isso facilita a abstração e favorece capacidade de reutilização do código.

Definições de funções

- Até agora apresentamos programas que consistiram em uma função denominada **main** que chamou as funções da biblioteca padrão para realizar suas tarefas.

- Doravante vamos analisar como os programadores escrevem suas próprias funções personalizadas.

Definições de funções

- Considere um programa que use a função **square** para calcular os quadrados dos números inteiros de 1 a 10.

```
#include <stdio.h>
int square(int);          /* protótipo da função */
int main()
(
        int x;
        for(x=1;x<=10;x++)
            printf("%d ", square(x));
        return 0;
}
int square(int y)      /* protótipo da função */
{
        return y * y ;
}
```

1	4	9	16	25	36	49	64	81	100

Definições de funções

- A linha:

  int square (int);

- É um protótipo da função. O **int** entre parênteses informa ao compilador que **square** espera receber um valor inteiro da função que faz a chamada.

- O **int** à esquerda do nome da função **square** informa ao compilador que **square** retorna um resultado inteiro à função que faz a chamada.

- O compilador consulta o protótipo da função para verificar se s chamadas de **square** contêm o tipo correto do valor de retorno, o número de argumentos, os tipos corretos de argumentos e se os argumentos estão na ordem correta.

Definições de funções

- O formato de uma definição de função é:

```
tipo-do-valor-de-retorno nome_da_função (lista_de_parâmetros)
{
        declarações
        instruções
}
```

- O nome_da_função é qualquer identificar válido.
- O tipo-do-valor-de-retorno é o tipo de dados do resultado devolvido à função que realiza a chamada.
- O tipo-do-valor-de-retorno **void** indica que a função não devolve um valor.
- A lista de parâmetros é uma lista separada por virgulas (caso mais de um), e caso não haja deve ser especificado como **void**.

Arquivos de cabeçalho

- Cada biblioteca padrão tem um arquivo de cabeçalho correspondente contendo os protótipos de todas as funções daquela biblioteca e definições dos vários tipos de dados e constantes necessários por elas.

Tipos de dados	Especificações de conversão de printf	Especificações de conversão de scanf
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
Char	%c	%c

Arquivos de cabeçalho

- Cada biblioteca padrão tem um arquivo de cabeçalho correspondente contendo os protótipos de todas as funções daquela biblioteca e definições dos vários tipos de dados e constantes necessários por elas.

Arquivo de cabeçalho da biblioteca padrão	Explicação
<assert.h>	Contem macros na ajuda por depuração
<ctype,h>	Converte letras em mai/min e vice-versa
<errno.h>	Define macros úteis para informações de erro
<float.h>	Contém limites do sistema p tamanho de PF
<limits.h>	Contém limites do sistema p tamanho de int.
<stdio.h>	Contém funções padrão de E/S e as informações utilizadas por elas.
<stdlib.h>	Contém funções para conversão de números em texto e texto em números, alocação de memória, números aleatórios e outras funções com várias finalidades.
<time.h>	Funções para manipular horários e datas

Arquivos de cabeçalho

- O programador pode criar arquivos de cabeçalho personalizados.
- Um arquivo de cabeçalho definido pelo programador pode ser incluído usando a diretiva de pré-processador **#include**.
- Por exemplo, o arquivo de cabeçalho **square.h** pode ser incluído da seguinte forma:

se estiver no diretório do fonte em questão:

```
#include “square.h”
```

ou com endereço absoluto:

```
#include <“C:\Users\José Luiz\Documents\Univás - José Luiz\Sistemas de Informação\LAB II\square.h”>
```

ou, se estiver no diretório padrão da IDE:

```
#include <square.h>
```

Cálculo do fatorial de um numero

```
#include <stdio.h>
int fat (int n);
int main (void)
{  int n, r;
   printf("Digite um número nao negativo:");
   scanf("%d", &n);
   r = fat(n);
   printf("Fatorial = %d\n", r);
   return 0;
}

/* função para calcular o valor do fatorial */
int fat (int n)
{  int i;
   int f = 1;
   for (i = 1; i <= n; i++)
      f *= i;
   return f;
}
```

"protótipo" da função: deve ser incluído antes da função ser chamada

chamada da função

"main" retorna um inteiro:
0 : execução OK
≠ 0 : execução ¬OK

declaração da função: indica o tipo da saída e o tipo e nome das entradas

retorna o valor da função

Variável Global

- Declarada fora do corpo das funções:
  - Visível por todas as funções subsequentes.
- Não é armazenada na pilha de execução:
  - Não deixa de existir quando a execução de uma função termina.
  - Existe enquanto o programa estiver executando.
- Utilização de variáveis globais:
  - Deve ser feito com critério.
  - Pode-se criar um alto grau de interdependência entre as funções.
  - Dificulta o entendimento e o reuso do código.

Variável Global

```
#include <stdio.h>

int s, p;  /* variáveis globais */

void somaprod (int a, int b)
{
   s = a + b;
   p = a * b;
}

int main (void)
{
   int x, y;
   scanf("%d %d", &x, &y);
   somaprod(x,y);
   printf("Soma = %d  produto = %d\n", s, p);
   return 0;
}
```

Variáveis Estáticas

- Declarada no corpo de uma função:
  - Visível apenas na função em que foi declarada
- Não é armazenada na pilha de execução:
  - Armazena em uma área de memória estática
  - Continua existindo antes ou depois da função ser executada
- Utilização de variáveis estáticas:
  - Quando for necessário recuperar o valor de uma variável atribuída na última vez que a função foi executada.

Variáveis Estáticas

- Exemplo: função para imprimir números reais:
  - Imprime um número por vez, separando-os por espaços em branco e colocando, no máximo, cinco números por linha.

```
void imprime ( float a )
{
   static int n = 1;
   printf(" %f   ", a);
   if ((n % 5) == 0) printf(" \n ");
   n++;
}
```

Comentários

- Variáveis estáticas e variáveis globais:
  - São inicializadas com zero, se não forem explicitamente inicializadas.
- Variáveis globais estáticas:
  - São visíveis para todas as funções subsequentes
  - Não podem ser acessadas por função definidas em outros arquivos
- Funções estáticas:
  - Não podem ser chamadas por funções definidas em outros arquivos

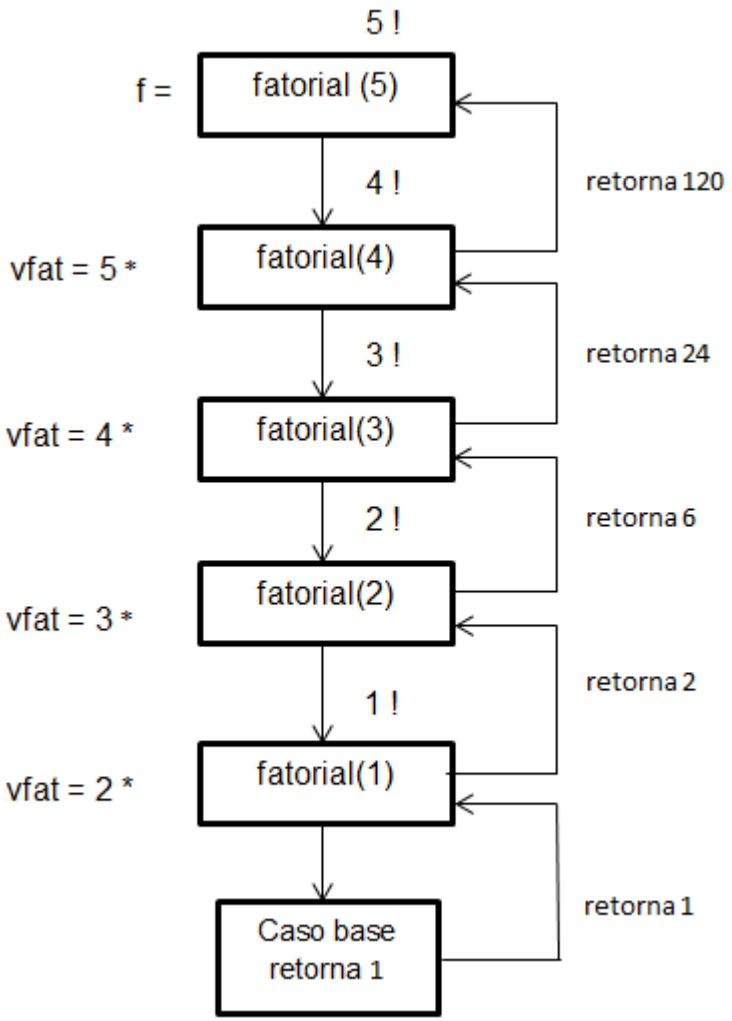
Funções Recursivas (recursão)

- Tipos de recursão:
  - Direta:
    - Uma função A chama a ela própria
  - Indireta:
    - Uma função A chama uma função B que por sua vez, chama A
- Comportamento:
  - Quando uma função é chamada recursivamente, cria-se um ambiente local pra cada chamada
  - As variáveis locais de chamadas recursivas são independentes entre si, somo se estivéssemos funções diferentes.

Funções Recursiv

- Exemplo: função recursiva para cá

```
int fat(int n)
{
        if (n <= 1)
                return 1;
        else
            return n * fat(n – 1);
}
```



Funções Recursivas (recursão)