

Técnicas de Programação II

Revisão TP1

Parte2

Profa.: Leila Andrade

e-mail: leila@uniriotec.br

Material inspirado no curso TPII
da Profa. Adriana Alvim

Definição de funções

- Um programa bem estruturado deve ser pensado em termos de funções
 - e estas, por sua vez, podem (e devem, se possível) esconder do corpo principal do programa detalhes ou particularidades de implementação
- Em **C**, tudo é feito através de funções
- Os exemplos anteriores utilizam as funções da biblioteca padrão para realizar entrada e saída **printf()**, **scanf()**, **pow()**, **sqrt()**, etc
 - as funções necessárias a um programa, que não constam dessa biblioteca, têm de ser programadas

Definição de funções

- A forma geral para definir uma função é

```
tipo_retornado nome_da_função (lista de parâmetros...){  
  corpo da função  
}
```

- tudo antes da chave aberta { faz parte do cabeçalho da função
- e tudo entre as chaves '{ }' faz parte do corpo da função
- a lista de parâmetros é uma lista de declarações separadas por vírgula
- o corpo da função é um bloco de comandos que também pode conter declarações

Definição de funções

- Para ilustrar a criação de funções
 - será considerado o cálculo do fatorial de um número inteiro
- Pode-se escrever uma função que
 - dado um determinado número inteiro não negativo n
 - imprime o valor de seu fatorial
- Um programa que utiliza esta função seria

Definição de funções

```
/* programa que le um
   numero e imprime seu
   fatorial */
```

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

```
void fat (int n);
```

```
/* função principal */
```

```
int main (void){
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    fat(n);
```

```
    return 0;
```

```
}
```

```
/* função para imprimir o
   valor do fatorial */
```

```
void fat ( int n ){
```

```
    int i;
```

```
    int f = 1;
```

```
    for (i = 1; i <= n; i++)
```

```
        f *= i;
```

```
    printf("Fatorial =
```

```
    %d\n", f);
```

```
}
```

Definição de funções

- Nesse exemplo, a função `fat` recebe como parâmetro o número cujo fatorial deve ser impresso
- Os parâmetros de uma função devem ser listados
 - com seus respectivos tipos
 - entre os parênteses que seguem o nome da função
- Quando uma função não tem parâmetros
 - colocamos a palavra reservada `void` entre os parênteses

Definição de funções

- Devemos notar que `main` também é uma função
 - sua única particularidade consiste em ser a função automaticamente executada após o programa ser carregado
 - como as funções `main` apresentadas até agora não recebem parâmetros, usamos a palavra `void` na lista de parâmetros

Definição de funções

- Além de receber parâmetros
 - uma função pode ter um valor de retorno associado
- No exemplo do cálculo do fatorial
 - a função `fat` não tem nenhum valor de retorno associado
 - portanto colocamos a palavra `void` antes do nome da função
 - indicando a ausência de um valor de retorno

```
void fat (int n) {  
    ...  
}
```

Definição de funções

- A função **main** deve ter obrigatoriamente um valor inteiro como retorno
- Esse valor pode ser usado pelo sistema operacional para testar a execução do programa
- A convenção geralmente utilizada faz com que a função **main** retorne
 - zero no caso da execução ser bem sucedida ou
 - diferente de zero no caso de problemas durante a execução

Definição de funções

- Salientamos que **C** exige que se coloque o **protótipo** da função antes dela ser chamada
- O **protótipo** de uma função consiste na repetição da linha de sua definição seguida do caractere (;)
- Temos então

```
void fat (int n); /* obs: existe ; no protótipo */
int main (void) {
    . . .
}
void fat (int n) { /* obs: nao existe ; na definição */
    . . .
}
```

Definição de funções

- A rigor, no protótipo não há necessidade de se indicar os nomes dos parâmetros
 - apenas os seus tipos
 - portanto seria válido escrever
`void fat (int) ;`
- Entretanto, geralmente mantemos os nomes dos parâmetros
 - pois servem como documentação do significado de cada parâmetro
 - desde que se utilize nomes coerentes

Definição de funções

- O protótipo da função informa ao compilador
 - o nome de uma função
 - o tipo de dados retornado pela função
 - o número de parâmetros que a função espera receber
 - a ordem em que eles são esperados
- Por exemplo, considere a chamada `fat(4.5);`
 - o compilador provavelmente indicaria o erro
 - pois estaríamos passando um **valor real** enquanto a função espera um **valor inteiro**

Definição de funções

- Por isso, exige-se a inclusão do arquivo `stdio.h` para a utilização das funções de entrada e saída da biblioteca padrão
 - nesse arquivo, encontram-se, entre outras coisas, os protótipos das funções `printf` e `scanf`
- Se uma função for definida antes de ser invocada
 - então a definição da função também serve como o protótipo da função
- Se uma função é invocada antes de ser definida e essa função não tiver um protótipo de função
 - ocorre um erro de compilação

Definição de funções

- Forneça sempre protótipos de função, mesmo que seja possível omiti-los
 - quando as funções são definidas antes de serem utilizadas
- Fornecer os protótipos evita associar o código à ordem em que as funções são definidas
 - o que pode mudar a medida que o programa cresce

Definição de funções

- Uma função pode ter um valor de retorno associado
- Para ilustrar a discussão, vamos reescrever o código anterior
 - fazendo com que a função `fat` retorne o valor do fatorial
 - a função `main` fica então responsável pela impressão do valor

Definição de funções

```
/* programa que le um numero e imprime seu fatorial (versão
2) */
#include <stdio.h>
int fat (int n);
int main (void){
    int n, r;
    scanf("%d", &n);
    r = fat(n);
    printf("Fatorial = %d\n", r);
    return 0;
}
/* funcao 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;
}
```

Definição de funções

- De fato, esta segunda implementação da função `fat` é mais adequada
 - pois a tarefa executada pela função se limita a fazer o cálculo do fatorial
 - a decisão de imprimir ou não o resultado na tela deve ficar a cargo da função que chama a função
 - denominada **função cliente**
- Por exemplo, podemos usar a função `fat` para avaliar qualquer expressão que envolva o cálculo do fatorial

Definição de funções

- Para ilustrar
 - consideremos o cálculo do número de combinações de n elementos tomados k a k
 - no qual a ordem dos elementos é relevante
 - esse número é dado pela fórmula do arranjo
$$a = n!/(n-k)!$$
 - por meio da função `fat`, podemos facilmente implementar uma função para o cálculo do número de arranjos

Definição de funções

```
int arranjo (int n, int k) {  
    int a;  
    a = fat(n)/fat(n-k);  
    return a;  
}
```

- ou simplesmente

```
int arranjo (int n, int k) {  
    return fat(n)/fat(n-k);  
}
```

Comando `return`

- O comando `return` é usado para terminar uma função
 - provavelmente retornando um valor
- A execução do comando `return` faz com que a função corrente termine
- O controle é transferido para o próximo comando imediatamente depois da chamada da função
- Se o comando `return` não for seguido de nenhuma expressão então o tipo da função deve ser `void`
- Se o comando `return` for seguido de alguma expressão então o tipo da função não pode ser `void`

Retorno de uma função

- Existem duas maneiras pelas quais uma função termina e retorna ao código que a chamou
- Retorno natural
 - quando a chave de fechamento ‘}’ da função for encontrada
- Retorno explícito
 - execução do comando **return**
 - pode haver **mais de um** em uma função

Comando `return`

- O comando `return` pode ou não incluir uma expressão

```
return;
```

```
return ++a;
```

```
return (a * b); /* não é necessário  
parênteses */
```

Comando `return`

- Comando `return` com expressão
 - seu valor é enviado ao programa que chamou a função
 - além disso, este valor é convertido, **se necessário**, para o tipo da função como especificado na definição da função

```
float f(char a, char b, char c) {  
    int i;  
    ....  
    return i; /* o valor retornado vai ser convertido  
para float */  
}
```

Referências

- W. Celes, R. Cerqueira e J.L. Rangel, **Introdução a Estruturas de Dados - com técnicas de programação em C**, Ed. Campus, 2004
- Fábio Mocarzel, **Apostila: Introdução à computação**, ITA, S. José Campos, SP
- Samuel P. Harbison III e Guy L. Steel Jr, **C - A Reference Manual**, Prentice Hall, 2002
- A. Kelley, **A Book on C**, Benjamin Cummings, 1995
- D.D Salvetti e L.M. Barbosa, **Algoritmos**, Pearson Makron Books, 1998
- H. Schildt, **C Completo e Total**, Pearson Makron Books, 1997
- Deitel e Deitel. **C++ Como Programar**, 5ª Edição. Pearson Education do Brasil