Funções

Marina Andretta

ICMC-USP

18 de abril de 2016

Funções

Uma função é um subprograma que realiza uma tarefa específica. Ela tem suas próprias variáveis, que não são "vistas" pelas outras funções.

Em linguagem C, uma função sempre deve ser declarada antes de ser usada. Você pode defini-la completamente ou apenas definir seu cabeçalho.

Funções

Para definir uma função em linguagem C, fazemos:

```
<tipo> <nome>(<tipo 1> <par 1>, ..., <tipo n> <par n>){
    <declarações de variáveis>;
    <comando 1>;
    ⋮
    <comando k>;
    return(<valor>);
}
```

Funções

Na definição da função:

- `<nome>` é o nome dado à função. Ele é usado quando se quer "chamar" a função para que ela seja executada.

- `<par 1>, ..., <par n>` são os parâmetros da função. Eles definem os valores que a função recebe e que são usados durante sua execução. Cada um destes parâmetros tem um nome e um tipo, dados por `<tipo 1>, ..., <tipo n>`.

Funções

- No corpo da função podem ser declaradas variáveis e executados comandos, como em um programa comum. Um ponto importante é que uma função não tem acesso aos nomes ou valores de variáveis declaradas em outras funções. A comunicação entre duas funções só pode ser feita através de seus parâmetros e dos valores que elas devolvem.

- `<valor>` é um valor que a função devolve depois de sua execução. Este valor deve respeitar o `<tipo>` definido no início da função.

Funções

Um exemplo de definição de uma função em linguagem C é:

```
float media(int a, int b) {
    float m;
    m = (a + b)/2.0;
    return(m);
}
```

Outra maneira de definir a mesma função:

```
float media(int a, int b) {
    return((a + b)/2.0);
}
```

Funções

Depois de definida a função, uma forma de usá-la é:

```
<nome variável> = <nome função>(<param 1>, ..., <param n>);
```

Aqui:

- `<nome variável>` é o nome da variável que deve armazenar o valor devolvido pela função, após sua execução.
- `<nome função>` é o nome da função a ser executada.
- `<param 1>, ..., <param n>` são os valores a serem usados na função, na ordem em que foram definidos.

Funções

No caso do exemplo, podemos fazer

```
x = media(2,3);
```

Depois desta chamada, a variável `x` (que deve ser do tipo `float`) recebe o valor `2.5`.

Os valores dos parâmetros podem ser passados diretamente, podem estar armazenados em variáveis, ser resultados de operações ou valores devolvidos por outras funções. A restrição é que os tipos sejam compatíveis.

Funções

Em alguns casos, a função não precisa devolver nenhum valor. Nestes casos, definimos o `<tipo>` como **void** e `<valor>` como vazio (ou **void**).

Se a função não precisa de nenhum parâmetro, basta não escrever nada entre os parênteses.

Funções

Um exemplo é:

```
void imprime_par() {
    int i;
    for(i = 0; i <= 10; i = i + 2) {
       printf(``%d '', i);
    }
    printf(``\n'');
    return;
}
```

Funções

Neste caso, como a função não devolve nenhum valor, podemos executá-la da sequinte maneira:

```
imprime_par();
```

Funções

É importante saber que os valores passados como parâmetros a uma função, quando armazenados em variáveis antes da chamada à mesma, não são alterados pela execução da função.

Por exemplo, se a função é definida como

```
void funcao(int a) {
    a = a + 1;
    return;
}
```

Funções

E a chamamos:

```
int x;
x = 1;
printf(''%d\n'', x);
funcao(x);
printf(''%d\n'', x);
```

Após a execução deste código, aparecem na tela os seguintes valores:

```
1
1
```

Exemplo de um algoritmo completo

```
int fatorial(int n) {
    int f;
    f = 1;
    while(n > 1) {
       f = f * n;
       n = n - 1;
    }
    return(f);
}

int main () {
    int n,k,fn,fk,fnk,c;
    printf(``Digite os valores de n e k:  '');
    scanf(``%d %d:  '',&n,&k);
    fn = fatorial(n);
    fk = fatorial(k);
    fnk = fatorial(n-k);
    c = fn/(fk*fnk);
    printf(``O numero de combinacoes de %d, %d a %d, eh %d.\n'',n,k,k,c);
    return(0);
}
```

Teste de mesa

Vamos simular a execução do algoritmo, supondo que o usuário digitou os números 4 e 2.

Depois da declaracao das variáveis, na função **main**, temos

```
main

n =
k =
fn =
fk =
fnk =
c =
```

Teste de mesa

Então, pede-se para o usuário digitar dois números, ele os digita (4 e 2) e estes valores são armazenados:

```
main

n = 2
k = 4
fn =
fk =
fnk =
c =
```

Teste de mesa

Depois disso, a função `fatorial` é chamada. O valor de seu parâmetro é definido (neste caso, 4) e sua variável é declarada:

`main`	`fatorial`
`n = 4`	`n = 4`
`k = 2`	`f =`
`fn =`	
`fk =`	
`fnk =`	
`c =`	

Teste de mesa

Então, o valor de `f` é definido como `1` e começa a execução do laço:

`main`	`fatorial`
`n = 4`	`n = 4`
`k = 2`	`f = 1`
`fn =`	
`fk =`	
`fnk =`	
`c =`	

Teste de mesa

main	fatorial
n = 4	n = ~~4~~ 3
k = 2	f = ~~1~~ 4
fn =	
fk =	
fnk =	
c =	

Teste de mesa

main	fatorial
n = 4	n = ~~4~~ ~~3~~ 2
k = 2	f = ~~1~~ ~~4~~ 12
fn =	
fk =	
fnk =	
c =	

Teste de mesa

main	fatorial
n = 4	n = ~~4~~ ~~3~~ ~~2~~ 1
k = 2	f = ~~1~~ ~~4~~ ~~12~~ 24
fn =	
fk =	
fnk =	
c =	

Teste de mesa

O laço é encerrado, a função devolve o valor de `f` (neste caso, 24) e este valor é armazenado na variável `fn`.

`main`	`fatorial`
`n = 4`	`n =` ~~`4`~~ ~~`3`~~ ~~`2`~~ `1`
`k = 2`	`f =` ~~`1`~~ ~~`4`~~ ~~`12`~~ `24`
`fn = 24`	
`fk =`	
`fnk =`	
`c =`	

Teste de mesa

Então, a função `fatorial` é chamada novamente. O valor de seu parâmetro é definido (neste caso, 2) e sua variável é declarada:

`main`	`fatorial`
`n = 4`	`n = 2`
`k = 2`	`f =`
`fn = 24`	
`fk =`	
`fnk =`	
`c =`	

Teste de mesa

Então, o valor de `f` é definido como `1` e começa a execução do laço:

main	fatorial
`n = 4`	`n = 2`
`k = 2`	`f = 1`
`fn = 24`	
`fk =`	
`fnk =`	
`c =`	

Teste de mesa

main	fatorial
n = 4	n = ~~2~~ 1
k = 2	f = ~~1~~ 2
fn = 24	
fk =	
fnk =	
c =	

Teste de mesa

O laço é encerrado, a função devolve o valor de `f` (neste caso, 2) e este valor é armazenado na variável `fk`.

`main`	`fatorial`
`n = 4`	`n = 2 1`
`k = 2`	`f = 1 2`
`fn = 24`	
`fk = 2`	
`fnk =`	
`c =`	

Teste de mesa

A função `fatorial` é chamada uma terceira vez. O valor de seu parâmetro é definido (neste caso, 2) e sua variável é declarada:

`main`	`fatorial`
`n = 4`	`n = 2`
`k = 2`	`f =`
`fn = 24`	
`fk = 2`	
`fnk =`	
`c =`	

Teste de mesa

Então, o valor de `f` é definido como `1` e começa a execução do laço:

`main`	`fatorial`
`n = 4`	`n = 2`
`k = 2`	`f = 1`
`fn = 24`	
`fk = 2`	
`fnk =`	
`c =`	

Teste de mesa

main	fatorial
n = 4	n = ~~2~~ 1
k = 2	f = ~~1~~ 2
fn = 24	
fk = 2	
fnk =	
c =	

Teste de mesa

O laço é encerrado, a função devolve o valor de `f` (neste caso, 2) e este valor é armazenado na variável `fnk`.

`main`	`fatorial`
`n = 4`	`n = 2 1`
`k = 2`	`f = 1 2`
`fn = 24`	
`fk = 2`	
`fnk = 2`	
`c =`	

Teste de mesa

Por fim, é calculado o valor da variável c

```
main

n = 4
k = 2
fn = 24
fk = 2
fnk = 2
c = 6
```

Na tela aparece a mensagem:

```
O numero de combinacoes de 4, 2 a 2, eh 6.
```