

Fundamentos de Programação

Passagem de Parâmetros por Referência

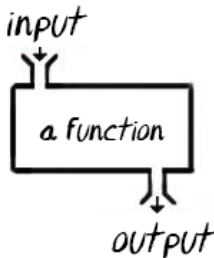
Dainf - UTFPR

Profa. Leyza B. Dorini
Prof. Bogdan T. Nassu

Funções: uma breve revisão

Descomposição de problemas

Como vimos, funções são utilizadas para implementar partes menores dos problemas, mais fáceis de entender e administrar. Este processo é conhecido como **modularização**! Seu princípio básico é:




Lembre-se que as funções podem não ter retorno (procedimentos).

Passagem de parâmetros por valor

```
1 void troca(int x, int y){  
2  
3  
4  
5  
6 }  
7  
8 int main(){  
9     int x, y;  
10  
11     x = 5;  
12     y = 10;  
13     troca(x, y);  
14  
15     printf("%d %d", x, y);  
16  
17     return 0;  
18 }
```

ao invocar a função troca(),
são passados como argumento
os conteúdos das variáveis x e y.



Passagem de parâmetros por valor

```
1 void troca(int x, int y){
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6 }
```

```
7
```

```
8 int main(){
```

```
9     int x, y;
```

```
10
```

```
11     x = 5;
```

```
12     y = 10;
```

```
13     troca(x, y);
```

```
14
```

```
15     printf("%d %d", x, y);
```

```
16
```

```
17     return 0;
```

```
18 }
```

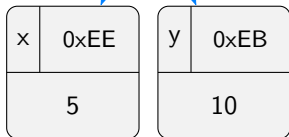
tais conteúdos são copiados para os parâmetros formais, os quais devem ser declarados com tipo compatível com o valor recebido e se comportam como variáveis locais.

Passagem de parâmetros por valor

```
1 void troca(int x, int y){
2
3
4
5
6 }
7
8 int main(){
9     int x, y;
10
11     x = 5;
12     y = 10;
13     troca(x, y);
14
15     printf("%d %d", x, y);
16
17     return 0;
18 }
```



no caso deste exemplo, o conteúdo de da variável x local à main() (ou seja, 5) é copiado para a variável x da função troca() e o conteúdo de y (10) é copiado para y.

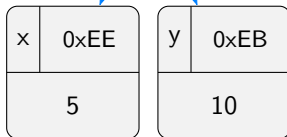


Passagem de parâmetros por valor

```
1 void troca(int x, int y){
2     int aux;
3     aux = x;
4     x = y;
5     y = aux;
6 }
7
8 int main(){
9     int x, y;
10
11     x = 5;
12     y = 10;
13     troca(x, y);
14
15     printf("%d %d", x, y);
16
17     return 0;
18 }
```

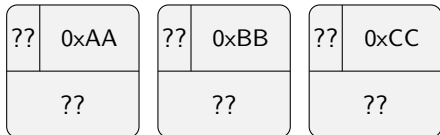


como as variáveis tem **escopo local**,
as alterações realizadas dentro da função
não tem efeito na variáveis da main()



Passagem de parâmetros por valor

```
1 void troca(int x, int y){
2
3
4
5
6 }
7
8 int main(){
9     int x, y;
10
11     x = 5;
12     y = 10;
13     troca(x, y);
14
15     printf("%d %d", x, y);
16
17     return 0;
18 }
```

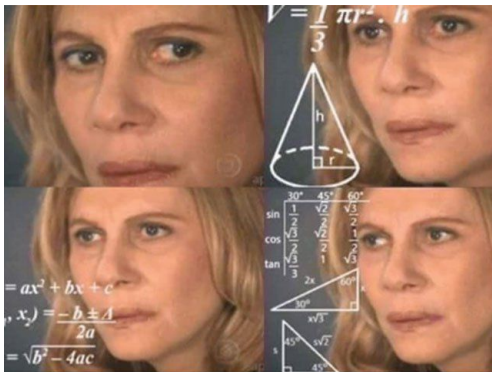


quando a função termina, as variáveis locais são dealocadas e o valor de aux é retornado e armazenado pela main()



Desafio

Vamos analisar um exemplo: usando passagem de parâmetros por valor, é possível fazer uma função que recebe dois inteiros e troca seu conteúdo de tal forma que as variáveis originais/iniciais também fiquem com os valores trocados?



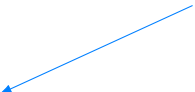
Funções: passagem de parâmetros por referência

Para alterar de fato o conteúdo da variável passada como argumento, podemos passar o **endereço** da variável (e não o seu valor). Desta forma, as alterações serão realizadas diretamente na variável, e não em uma cópia.

Passagem de parâmetros por referência

```
1 void troca(int *px, int *py){  
2  
3  
4  
5  
6 }  
7  
8 int main(){  
9     int x, y;  
10  
11     x = 5;  
12     y = 10;  
13     troca(&x, &y);  
14  
15     printf("%d %d", x, y);  
16  
17     return 0;  
18 }
```

note que, neste exemplo, ao invocar a função troca(), são passados os endereços das variáveis x e y.



Passagem de parâmetros por referência

```
1 void troca(int *px, int *py){
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6 }
```

```
7
```

```
8 int main(){
```

```
9     int x, y;
```

```
10
```

```
11     x = 5;
```

```
12     y = 10;
```

```
13     troca(&x, &y);
```

```
14
```

```
15     printf("%d %d", x, y);
```

```
16
```

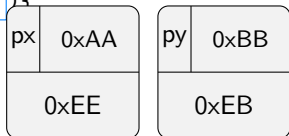
```
17     return 0;
```

```
18 }
```

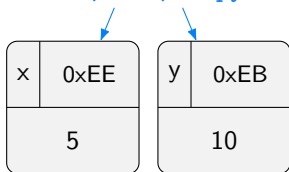
na passagem de parâmetros por referência,
tais endereços são copiados para os parâmetros formais,
os quais são declarados como ponteiros.

Passagem de parâmetros por referência

```
1 void troca(int *px, int *py){
2
3
4
5
6 }
7
8 int main(){
9     int x, y;
10
11     x = 5;
12     y = 10;
13     troca(&x, &y);
14
15     printf("%d %d", x, y);
16
17     return 0;
18 }
```



no caso deste exemplo,
o endereço de x (ou seja, 0xEE)
é copiado para a variável px
e o endereço de y (0xEB)
é copiado para py.



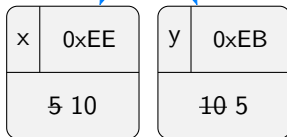
Passagem de parâmetros por referência

```
1 void troca(int *px, int *py){  
2     int aux  
3     aux = *px;  
4     *px = *py;  
5     *py = aux;  
6 }
```



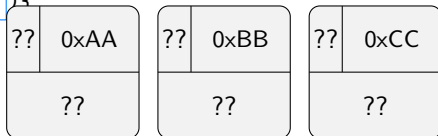
como os parâmetros formais da função
são ponteiros que apontam para as variáveis
na main(), alterações realizadas dentro da função
tem efeito nas variáveis referenciadas

```
7  
8 int main(){  
9     int x, y;  
10  
11     x = 5;  
12     y = 10;  
13     troca(&x, &y);  
14  
15     printf("%d %d", x, y);  
16  
17     return 0;  
18 }
```

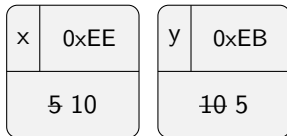


Passagem de parâmetros por referência

```
1 void troca(int *px, int *py){
2     int aux
3     aux = *px;
4     *px = *py;
5     *py = aux;
6 }
7
8 int main(){
9     int x, y;
10
11     x = 5;
12     y = 10;
13     troca(&x, &y);
14
15     printf("%d %d", x, y);
16
17     return 0;
18 }
```



quando a função termina, as variáveis locais são desalocadas e o valor de aux é retornado e armazenado pela main()

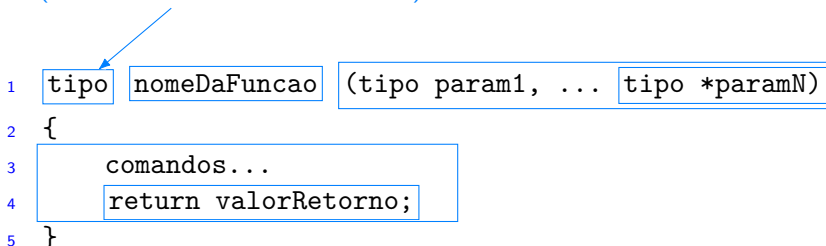


Definindo funções

Definindo uma função

A definição de funções continua igual.

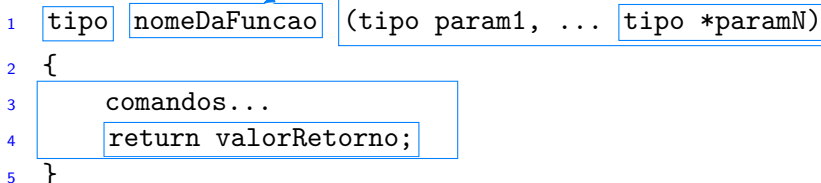
possui um tipo, o qual corresponde
ao tipo de seu valor de retorno
(ou void, caso não tenha retorno)



```
1 tipo nomeDaFuncao (tipo param1, ... tipo *paramN)
2 {
3     comandos...
4     return valorRetorno;
5 }
```

Definindo uma função

é por meio deste nome que
você irá invocar (chamar) a função -
portanto, escolha um que seja significativo



```
1 tipo nomeDaFuncao (tipo param1, ... tipo *paramN)
2 {
3     comandos...
4     return valorRetorno;
5 }
```

The diagram illustrates the syntax for defining a function in C. It shows five lines of code. Line 1: `tipo nomeDaFuncao (tipo param1, ... tipo *paramN)`. Line 2: `{`. Line 3: `comandos...`. Line 4: `return valorRetorno;`. Line 5: `}`. Blue boxes highlight the following parts: `tipo` on line 1; `nomeDaFuncao` on line 1; the entire parameter list `(tipo param1, ... tipo *paramN)` on line 1; the opening curly brace `{` on line 2; the closing curly brace `}` on line 5; the text `comandos...` on line 3; and the text `return valorRetorno;` on line 4. A blue arrow points from the text 'é por meio deste nome que...' to the `nomeDaFuncao` box.

Definindo uma função

```
1 tipo nomeDaFuncao (tipo param1, ... tipo *paramN)
2 {
3     comandos...
4     return valorRetorno;
5 }
```

bloco de comandos da
função (delimitado por chaves)

Definindo uma função

```
1 tipo nomeDaFuncao (tipo param1, ... tipo *paramN)
2 {
3     comandos...
4     return valorRetorno;
5 }
```

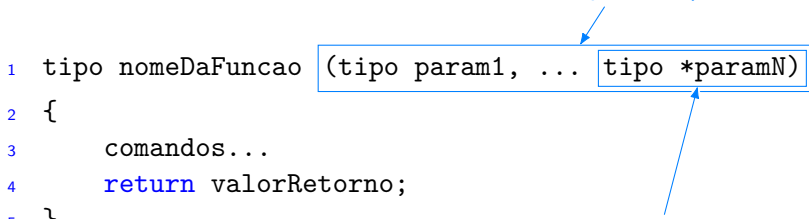
↑
retorna o conteúdo da variável
valorRetorno para quem
chamou a função (se o tipo da
função é void, não tem return)

Definindo uma função

Entretanto, preste atenção no seguinte aspecto...

parâmetros formais - cada um tem seu tipo e se comporta como uma variável local, a qual é inicializada com os valores passados no momento que a função é invocada.

```
1 tipo nomeDaFuncao (tipo param1, ... tipo *paramN)
2 {
3     comandos...
4     return valorRetorno;
5 }
```



a diferença agora é que, caso algum valor seja recebido por referência, o(s) parâmetro(s) formal(is) em questão precisa(m) ser um **ponteiro**.

Definindo uma função

Fique atento ao acessar os parâmetros recebidos por referência dentro da função!

neste exemplo, dois parâmetros são passados por valor (soma e qtde) e um por referência (pt).

```
1 float teste (float soma, int qtde, int *aux) {  
2  
3     float media;  
4  
5     media = soma / qtde;  
6     *aux = qtde * 2;  
7  
8     return media;  
9 }
```

se você quiser acessar o conteúdo da variável referenciada, o parâmetro pt precisa ser acessado usando o operador *

Checkpoint (parte 1)

Complete o trecho de código abaixo de forma a atribuir para a variável soma o resultado de $a + b$ e para a variável referenciada por sub o resultado da subtração de $a - b$.

```
1 int soma (int a, int b, int *sub){  
2  
3     int soma;  
4  
5     ????? = a + b;  
6     ????? = a - b;  
7  
8     return soma;  
9 }
```

Checkpoint (parte 1)

```
1 int soma (int a, int b, int *sub){
```

```
2
```

```
3     int soma;
```

devemos acessar diretamente
a variável local soma...

```
4
```

```
5     soma = a + b;
```

```
6     *sub = a - b;
```

... e usar o operador * para
a variável sub, pois ela é um ponteiro.

```
7
```

```
8     return soma;
```

```
9 }
```

Portanto, quem vai receber o
resultado de a-b é a
variável apontada por sub

Chamando (invocando) funções

É a mesma coisa que já estudamos anteriormente: para que as funções definidas sejam de fato executadas, precisamos chamá-las (invocá-las)!

Chamando (invocando) uma função

quando houver, o valor retornado pela função precisa ser utilizado de forma adequada

os argumentos passados como entrada precisam ser compatíveis com o que está definido na função

```
1  resposta = soma(n1, n2, &aux);
2
3
4
5
6
7
8
9
10 int soma (int a, int b, int *sub)
```

Chamando (invocando) uma função

```
1 resposta = soma(n1, n2, &aux);
```

2
3
4
5 no caso do exemplo, a
6 variável resposta
7 deveria ser do
8 tipo int

```
9  
10 int soma (int a, int b, int *sub)
```

neste exemplo específico, como a função soma() espera receber os dois primeiros argumentos do tipo int, as variáveis n1 e n2 **devem** ser inteiras. Além disso, também precisamos passar como terceiro parâmetro o endereço de uma variável do tipo inteiro

Exercício de fixação

Exercício de fixação 1

Faça um procedimento que, dados dois números inteiros, `a` e `b`, recebidos por referência, decmente o primeiro e incremente o segundo. Depois, invoque a função criada a partir da `main()`.
Protótipo:

```
void questao01(int *a, int *b);
```

Exercício de fixação 1

```
1 void questao01(int *a, int*b);
2
3 int main(){
4     int v1, v2;
5
6     printf("Digite dois valores inteiros: ");
7     scanf("%d %d", &v1, &v2);
8
9     questao01(&v1, &v2);
10
11     printf("Os novos valores são: %d %d\n", v1,v2);
12
13     return 0;
14 }
15
16 void questao01(int *a, int*b)
17 {
18     *a = *a - 1; //ou (*a)--;
19     *b = *b + 1; //ou (*b)++;
20 }
```

Exercício de fixação 2

Faça uma função que leia n valores inteiros do teclado e retorne a média, a soma e o maior valor.



Também está com dúvida sobre como retornar três valores (dado que no return só podemos retornar um)?

Exercício de fixação 2

```
1 float min_max_media(int n, float *maior, float *menor){
2     int i;
3     float aux, soma=0;
4
5     *maior = 0;
6     *menor = 10; //como são notas entre 0 e 10
7
8     for(i=0; i<n; i++){
9         scanf("%f", &aux);
10        if(aux > *maior)
11            *maior = aux;
12        if(aux < *menor)
13            *menor = aux;
14        soma = soma + aux;
15    }
16    return soma/n;
17 }
```


Exercício de fixação 2

Para invocar a função criada...

```
1 float min_max_media(int n, float *maior, float *menor){
2     .... //slide anterior
3 }
4
5 int main(){
6
7     float media, max, min;
8     int n;
9
10    printf("Quantos valores deseja ler? ");
11    scanf("%d", &n);
12
13    media = min_max_media(n, &max, &min);
14
15    printf("Maior: %f \n", max);
16    printf("Menor: %f \n", min);
17    printf("Media: %f \n", media);
18
19    return 0;
20 }
```