

Linguagem C: Funções

Programar em C

Funções

- Funções são blocos de código que podem ser nomeados e chamados de dentro de um programa
- Estrutura:

```
valor_retornado nome_função ( parâmetros )  
{  
    declarações  
    comandos  
}
```

Funções

- uma função pode retornar qualquer valor válido em C, sejam de tipos pré-definidos (*int*, *char*, *float*) ou de tipos definidos pelo usuário (*struct*, *typedef*)
- uma função que não retorna nada é definida colocando-se o tipo *void* como valor retornado (= procedure)
- Pode-se colocar *void* entre parênteses se a função não recebe nenhum parâmetro

Declaração de Funções

- Funções devem ser *definidas* ou *declaradas* antes de serem utilizadas
- A declaração apenas indica a *assinatura* ou *protótipo* da função:

valor_retornado nome_função(*declaração_parâmetros*);

- Menor função possível:

```
void faz_nada( void ) {}
```

Passagem de Parâmetros

- em C os argumentos para uma função são sempre passados por valor (*by value*), ou seja, *uma cópia* do argumento é feita e passada para a função

```
void loop_count( int i ) {  
    printf( "Em loop_count, i = " );  
    while( i < 10 ){  
        printf ( "%d ", i++);      \\ ==> i = 2 3 4 5 6 7 8 9  
    }  
}
```

```
void main( ) {  
    int i = 2;  
    loop_count( i );  
    printf( "\nEm main, i = %d.\n", i );  \\ ==> i = 2.  
}
```


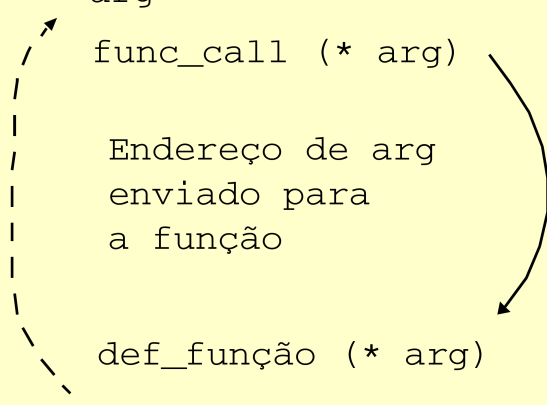
Passagem de Parâmetros

- como, então, mudar o valor de uma variável ?

passagem de parâmetro por referência

- enviar o *endereço* do argumento para a função

Passagem de Parâmetros

Passagem por valor	Passagem por referência
<p>arg</p> <p>func_call (arg)</p> <p>Cópia de arg enviado para a função</p> <p>def_função (arg)</p> <p>arg</p> 	<p>arg</p> <p>func_call (* arg)</p> <p>Endereço de arg enviado para a função</p> <p>def_função (* arg)</p> <p>(* arg)</p> 

Passagem de Parâmetros

- Passagem por referência:

```
void loop_count( int *i ) {  
    printf( "Em loop_count, i = " );  
    while( i < 10 ) {  
        printf ( "%d ", (*i)++);  \\      ==> i = 2 3 4 5 6 7 8 9  
    }  
}
```

```
void main( ) {  
    int i = 2;  
    loop_count( &i );  
    printf( "\nEm main, i = %d.\n", i );  \\      ==> i = 10.  
}
```

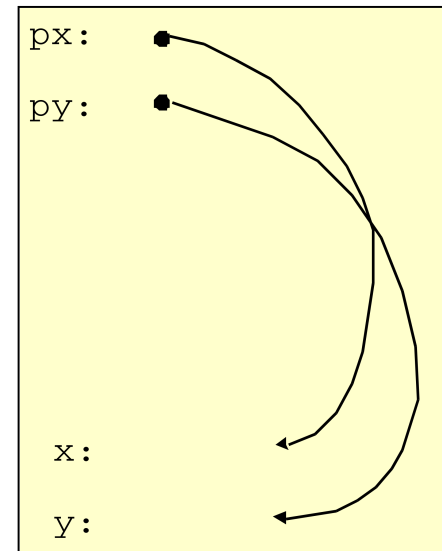

Prática: função *troca*

- Fazer uma função *troca(px, py)* que recebe como parâmetros 2 ponteiros para inteiros e troca o conteúdo deles
- ex:
 int x = 10, y = 20;
 troca(&x, &y);
 printf("x=%d y=%d", x, y) \\ => x=20 y=10

Prática: função *troca*

```
void troca (int *px, int *py)
{
    int temp;

    temp=*px;
    *px=*py;
    *py=temp;
}
```



Retornando Valores

- uma função retorna um valor através do comando *return*

- Ex:

```
int power (int base, int n) { // potência
    int i,p;

    p = 1;
    for (i = 1; i <= n; ++i)
        p *= base;
    return p;
}
```

Funções

- o valor retornado por uma função é sempre copiado para o contexto de chamada (retorno *by value*)

```
x = power(2, 5);           /* atribuição */
if (power(7, 2) > 12543)   /* comparação */
    printf("Numero grande!");
x = 10*power(2, 3);        /* expressão */
array[get_index()];        /* índice */
funcao( get_arg() );       /* argumento */
```

Prática: Localiza *char* em *string*

- Fazer uma função que procura um caracter em um *string* e retorna o seu endereço caso o encontre, senão retorna NULL (ponteiro nulo)

- Ex:

```
char *achachar (char *str, char c) {...}  
char str[] = "abcd5678";  
achachar(str, 'c');
```

//==> retorna endeço do terceiro caracter do *string*:
&str[2]

Achachar

```
char *achachar (char *str, char c) {  
    char *pc = str;  
  
    while (*pc != c && *pc != '\0') pc++;  
    return pc;  
}
```

```
char *achachar (char *str, char c) {  
    int i=0;  
  
    while (str[i] != c && str[i] != '\0') i++;  
    return &str[i];  
}
```

Parâmetros para *main()*

- ao executar programas a partir de linha de comando, é possível passar parâmetros diretamente para a função *main()*
- os argumentos são fornecidos na forma de um *array de strings*.
- *main()* é definida com dois parâmetros:
`main (int argc, char *argv[])`
argc é o número de argumentos
argv é o array de argumentos

Recursão em C

- uma função é dita recursiva quando dentro do seu código existe uma chamada para si mesma

Ex: cálculo do fatorial de um número:

$$n! = n * (n - 1)!$$

Exemplo Fatorial

```
#include <stdio.h>

int fatorial (int n)
{
    if (n == 0)    /* condição de parada da recursão */
        return 1;
    else if (n < 0) {
        printf ("Erro: fatorial de número negativo!\n");
        exit(0);
    }
    return n*fatorial(n-1);
}
```