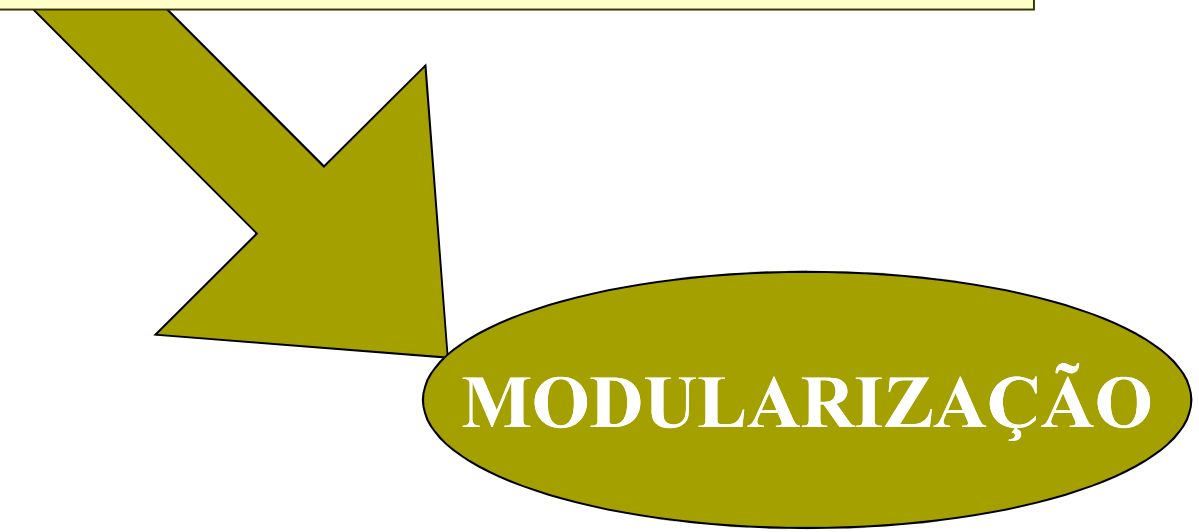
ICMC USP

Linguagem de Programação C

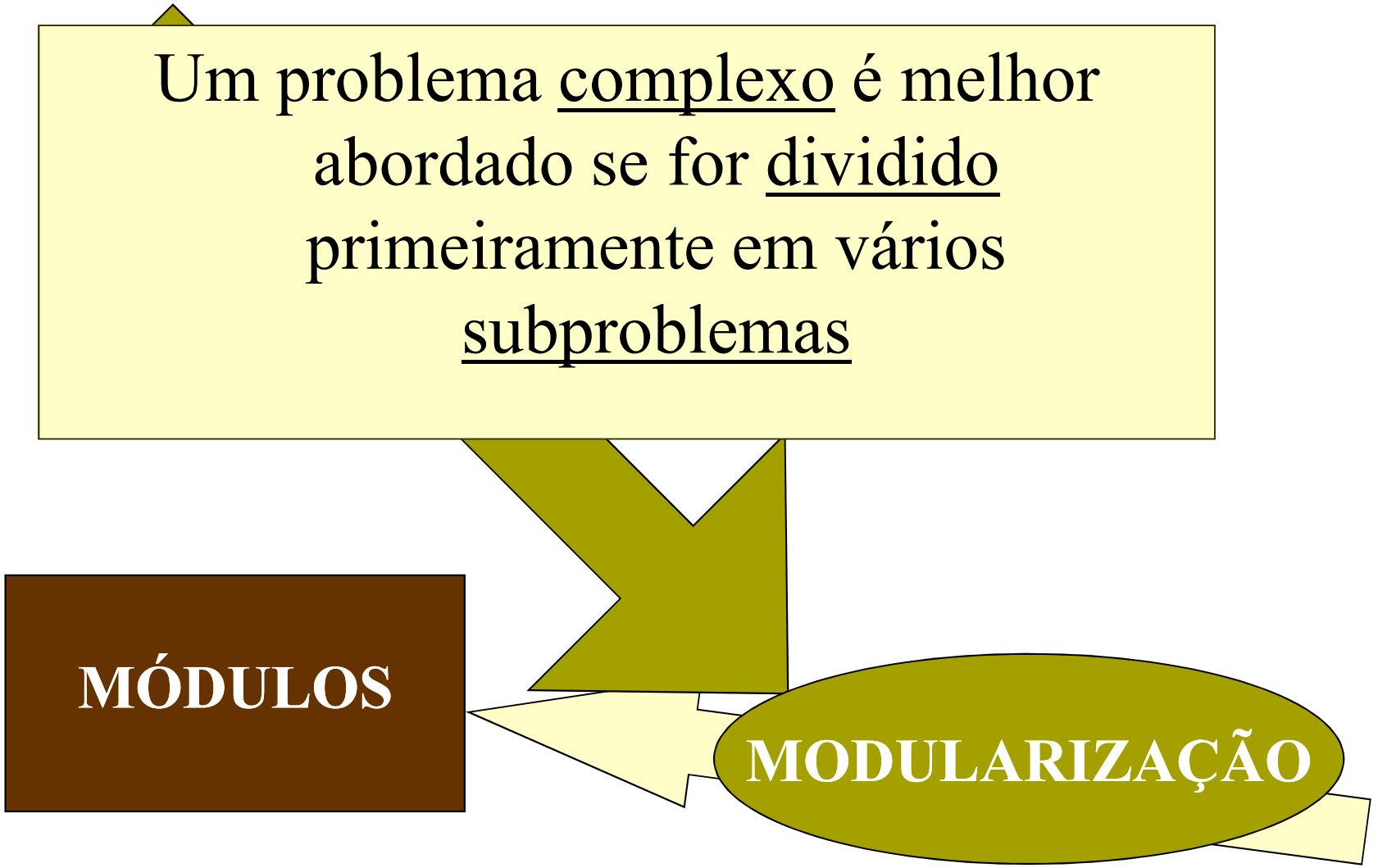
Aula: Funções em C
Prof Alneu de Andrade Lopes

MODULARIZAÇÃO

Um problema complexo é melhor abordado se for dividido primeiramente em vários subproblemas

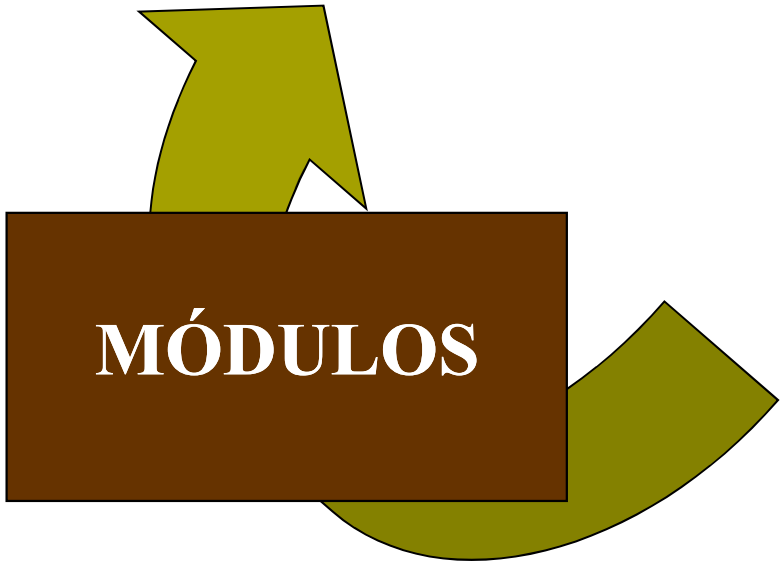


MODULARIZAÇÃO



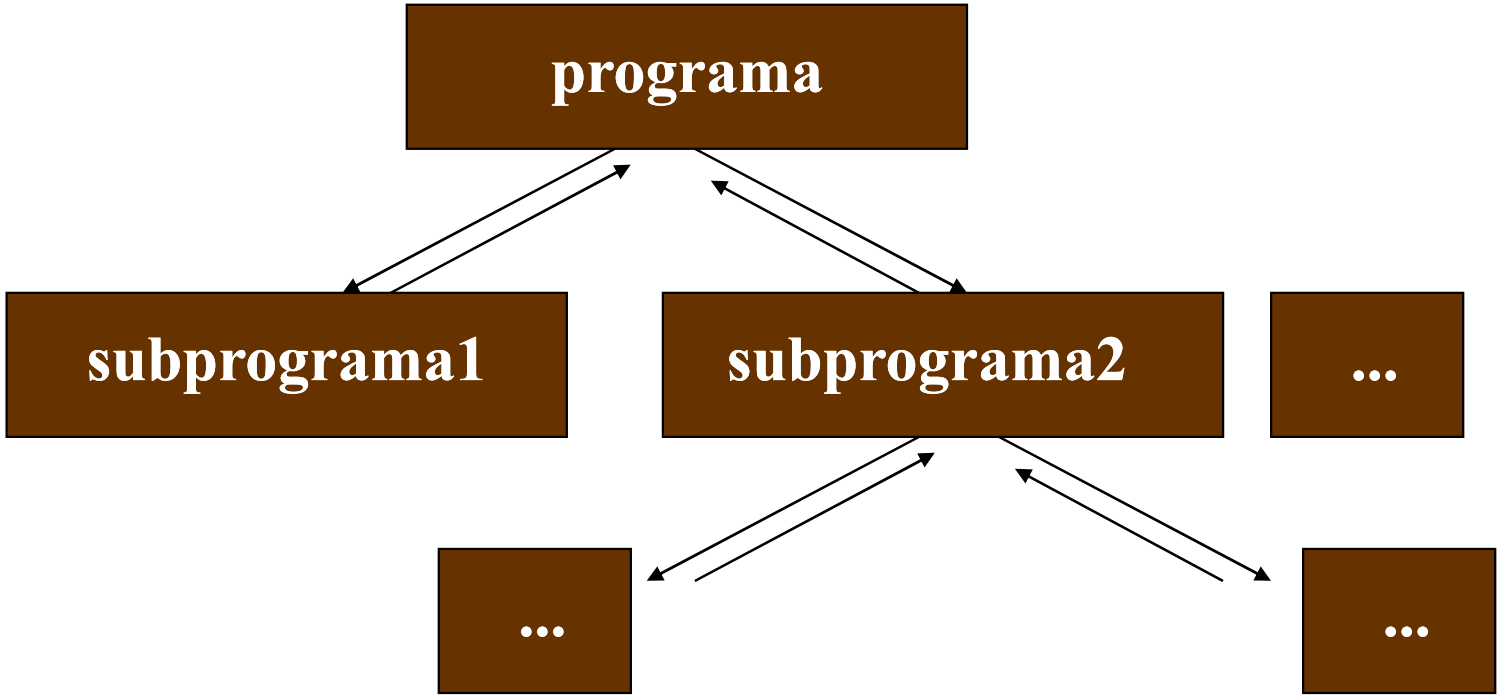
MODULARIZAÇÃO

A implementação desses Módulos é feita através de SUBPROGRAMAS



MODULARIZAÇÃO

A implementação desses Módulos é feita através de SUBPROGRAMAS



MODULARIZAÇÃO

A implementação desses Módulos é feita através de SUBPROGRAMAS

MÓDULOS

Os SUBPROGRAMAS em C são implementados por **FUNÇÕES**

FUNÇÕES

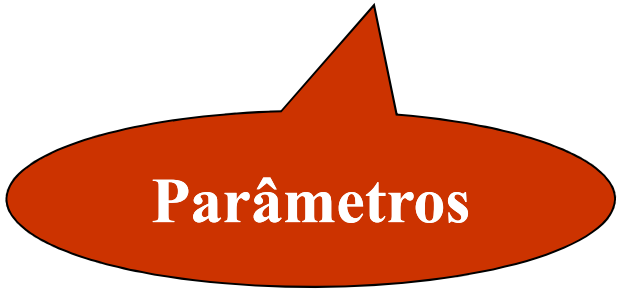
- As linguagens de programação têm à sua disposição várias funções pré-definidas:
- EXEMPLO (*C*)
  - pow (X,2) quadrado de X
  - printf(“%s” ,X) imprime string X

FUNÇÕES

- As linguagens de programação têm à sua disposição várias funções pré-definidas:
- EXEMPLO (*C*)
  - pow (X,2) quadrado de X
  - printf("%s" ,X) imprime string X

FUNÇÕES

- As linguagens de programação têm à sua disposição várias funções pré-definidas:
- EXEMPLO (*C*)
  - pow (X,2) quadrado de X
  - printf("%s" ,X) imprime string X



FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:

H = pow ((pow(X,2) + pow (Y,2)), 0.5);

FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:

H = pow ((pow(X,2) + pow (Y,2)), 0.5);

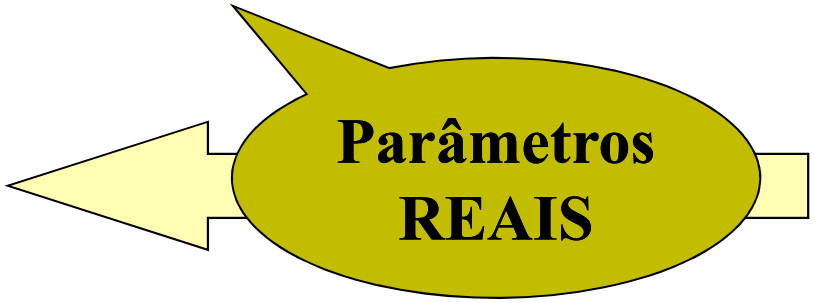
Parâmetros REAIS

FUNÇÕES

- As Funções Pré Definidas podem ser usadas diretamente em expressões:
- Exemplo:

H = pow ((pow(X,2) + pow (Y,2)), 0.5);

Estes parâmetros devem ser valores conhecidos



FUNÇÕES

Se houver necessidade o programador pode **definir** suas próprias **FUNÇÕES**

```
# include <stdio.h>

int sqr(float x){
    x = x*x;
    return(x);
}

main(){
    float t=10;
    printf("%f",sqr(t));
}
```

DEFINIÇÃO DE FUNÇÃO

Linguagem Algoritmica

função

declaração do tipo da saída da função NOME-DA-FUNÇÃO (declaração do tipo do parâmetro*1* parâmetro1, declaração do tipo do parâmetro*2* parâmetro2, ..., declaração do tipo do parâmetro*n* parâmetro*n*)

início

| declaração das variáveis utilizadas localmente

corpo da função

fim

DEFINIÇÃO DE FUNÇÃO

Em C

Tipo do retorno NOME-DA-FUNÇÃO (tipo_p1 p1, tipo_p2 p2,..., tipo_pn pn){

declaração de variáveis utilizadas no *corpo da função;*

corpo da função;

};

CARACTERÍSTICAS DAS FUNÇÕES

- Retorna um único valor
- Na definição da função devem ser declarados:
  - o tipo de todos os parâmetros
  - o tipo do valor que a função retorna
  - todas as variáveis utilizadas internamente no subprograma (variáveis locais)
- Para utilizar a função no programa principal basta colocar seu nome (identificador) e os parâmetros.

Resumo

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

valor_retornado nome_função (*parâmetros*)
{
  declarações
  comandos
}

Detalhes

- 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 ) {}
```

DEFINIÇÃO DE FUNÇÃO

Exemplo - Função Fatorial

```
função
real FAT(declaração do tipo do parâmetro X)
início da função
    //declaração das variáveis utilizadas no subprograma
    P← 1
    para I de 1 até X
        faça  P ← P * I
    fim para
    return(P);
fim da função
```

DEFINIÇÃO DE FUNÇÃO

- Função Fatorial

tipo do valor que a função retorna

função

real FAT(declaração do tipo do parâmetro X)

início d função

o das variáveis utiliz no subprograma

NOME DA FUNÇÃO

tipo do parâmetro

para I de 1 at

| faça P ←

fim para

return(P);

fim da função

PARÂMETRO FORMAL
quando a função
for utilizada, este parâmetro
recebe um valor real
lido ou atribuído no
programa principal

DEFINIÇÃO DE FUNÇÃO

Exemplo - Função Fatorial

```
função
real FAT(declaração do tipo do parâmetro X)
início da função
   //declaração das variáveis utilizadas no subprograma
   P← 1
   para I de 1 até X
   |  faça  P ← P * I
   fim para
   return(P);
fim da função
```

Declarações relativas às variáveis do algoritmo

DEFINIÇÃO DE FUNÇÃO

Exemplo - Função Fatorial

```
função
real FAT(declaração do tipo do parâmetro X)
início da função
   //declaração das variáveis utiliza
   P← 1
   para I de 1 até X
   |  faça  P ← P * I
   fim para
   return(P);
fim da função
```

O número para ona qual é calculado o fatorial entra como parâmetro

NÃO PRECISA SER LIDO DENTRO DA FUNÇÃO

DEFINIÇÃO DE FUNÇÃO

Exemplo - Função Fatorial

```
função
real FAT(declaração do tipo do pa
início da função
   declaração das variáveis utilizad
   P← 1
   para I de 1 até X
      faça  P ← P * I
   fim para
   return(P);
fim da função
```

- O resultado do cálculo do fatorial deve ser retornado para o programa que chamou a função.

Exercício

- Implemente a função fatorial.

Passagem de Parâmetros

em C os argumentos para uma função são, via de regra, 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.
}
```

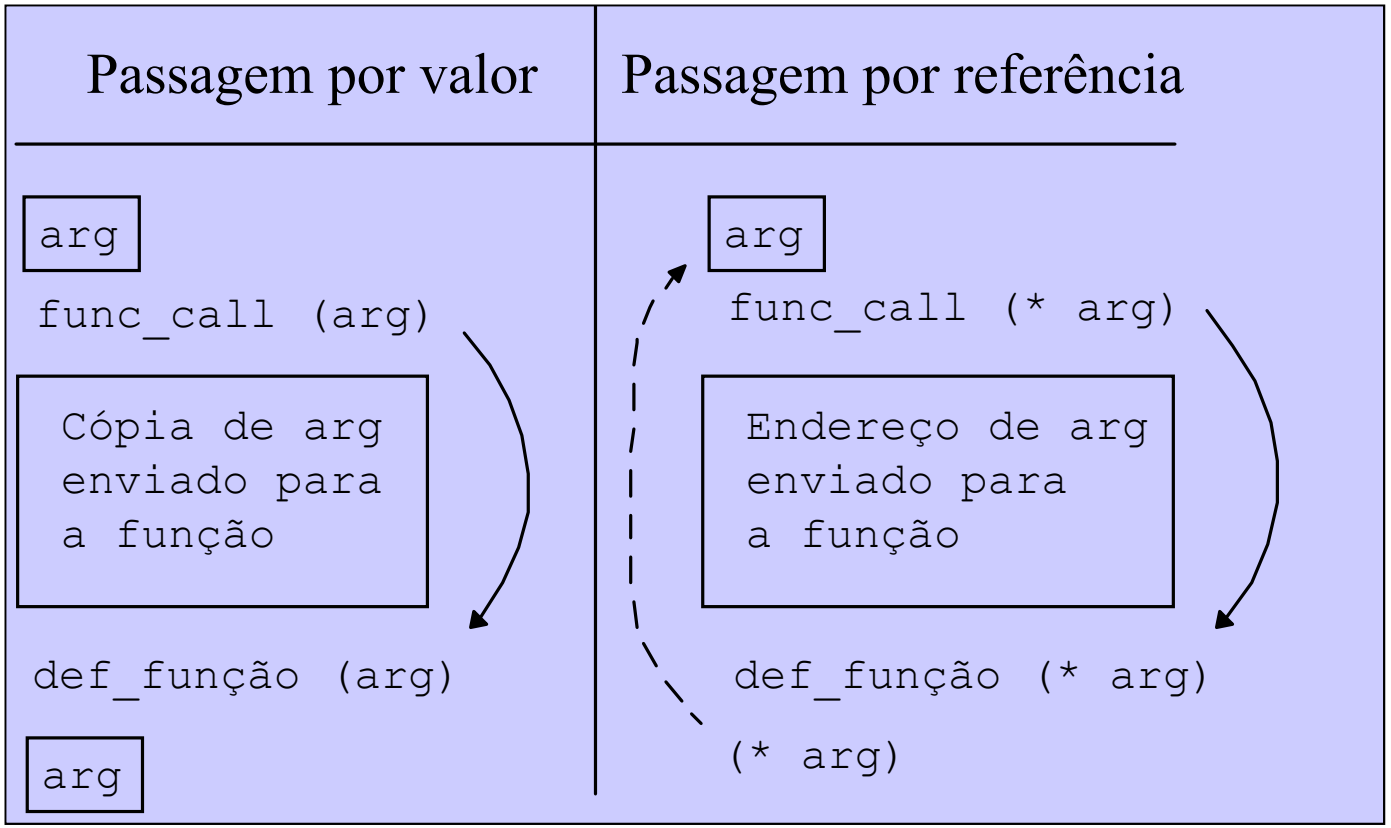
Passagem de Parâmetros

- 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 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*

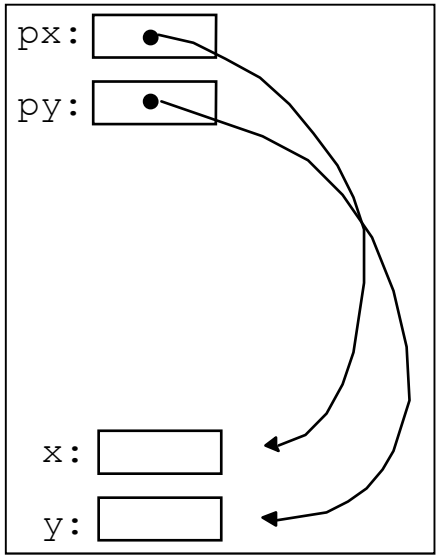
- Fazer uma função *troca(px, py)* que recebe como parâmetros 2 ponteiros para inteiros e troca o conteúdo deles
- ex:

```
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) {
  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 */
```

Ex: Concatena Strings

```
char *concatena( char cabeca[], char cauda[] )
{
    int i, j;

    for (i = 0; cabeca[i] != '\0'; i++);
    for (j = 0; (cabeca[i] = cauda[j]) != '\0'; i++, j++);
    cabeca[i] = '\0';
    return cabeca;
}
```

Exemplo (cont.)

```
int main( )
{
  char nome[80] = "Santos";
  char sobrenome[] = " Dumont";

  printf( "O nome é %s.\n",
          concatena( nome, sobrenome ) );

  return 0;
}
```

==> Santos Dumont

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 ? pc : NULL;
}
```

Número de Parâmetros Variável

- C permite declarar funções com número variável de argumentos através da inserção de reticências "..."

  função (arg1, arg2, ...);

- Como determinar o número de parâmetros passados:
  - string de formato, como no comando printf:

    Ex: **printf ("%s %d %f\n", s, i, f);**
  - pela especificação do número de parâmetros

    Ex: **soma (3, 10, -1, 5);**
  - pela inclusão de um valor de terminação

    Ex: **media (1, 4, 6, 0, 3, -1);**

Acesso aos Parâmetros

- C oferece uma série de macros para acessar uma lista de argumentos:
- *va_list*: é um tipo pré-definido utilizado para declarar um ponteiro para os argumentos da lista
- *va_start(va_list ap, ultimo_arg_def):* inicia o ponteiro *ap* fazendo-o apontar para o primeiro argumento da lista, ou seja, o primeiro argumento depois de *ultimo_arg_def*.
  - *ultimo_arg_def* é o nome do último argumento especificado no declaração da função

Acesso aos Parâmetros

- *type va_arg(va_list ap, type):* retorna o valor do argumento apontado por ap e faz ap apontar para o próximo argumento da lista. Type indica o tipo de argumento lido (int, float, etc.)

- *void va_end (va_list ap)* : encerra o acesso à lista de parâmetros. Deve sempre ser chamada no final da função

Exemplo Parâmetros Variáveis

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
int sum( int no_args, ... ) {
  va_list ap;     int result = 0, i;
  va_start( ap, no_args );
  for( i = 1; i <= no_args; i++ ) {
    result += va_arg( ap, int );
  }
    va_end(ap);
  return result;
}
```

```
sum( 5, 3, 5, 18, 57, 66 ) ==> 149
sum( 2, 3, 5 ) ==> 8
```

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[])

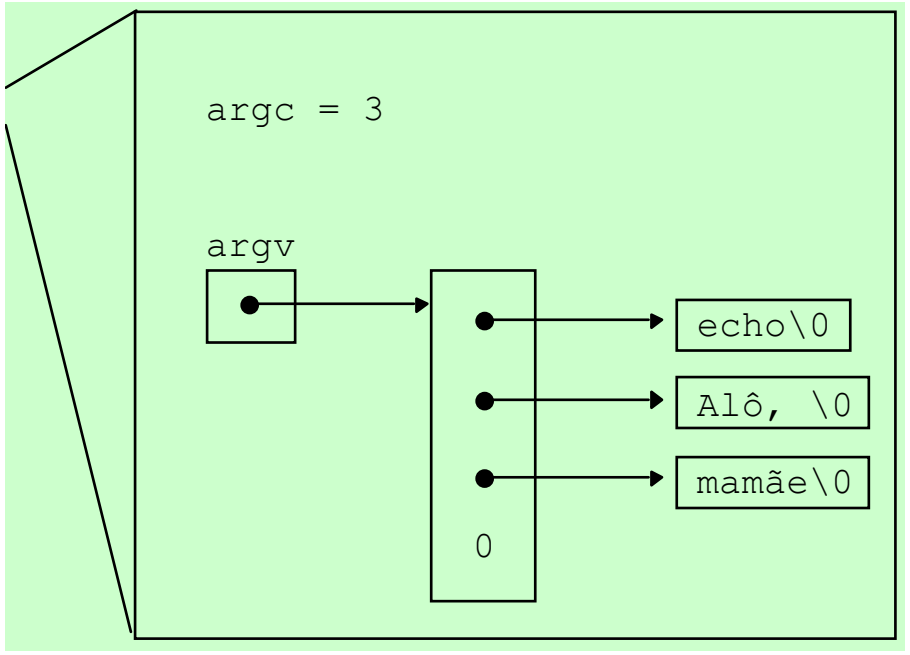
  argc é o número de argumentos

  argv é o array de argumentos

Parâmetros para main()

- por convenção, o primeiro argumento *argv[0]* é o nome do programa e o último é *0 (zero)*
- *ex:*

  echo Alô, mamãe



Parâmetros para main()

Uma possível implementação para echo:

```
#include <stdio.h>

void main( int argc, char *argv[] ) {
  int i;
   for ( i = 1; i < argc; i++ )
     printf("%s%s", argv[i], (i < argc-1)?" ":"");
   printf("\n");
}
```