Universidade de Brasília

Departamento de Ciência da Computação

Curso C: Funções e Macros

Prof. Ricardo Pezzuol Jacobi
rjacobi@cic.unb.br

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.
}
```

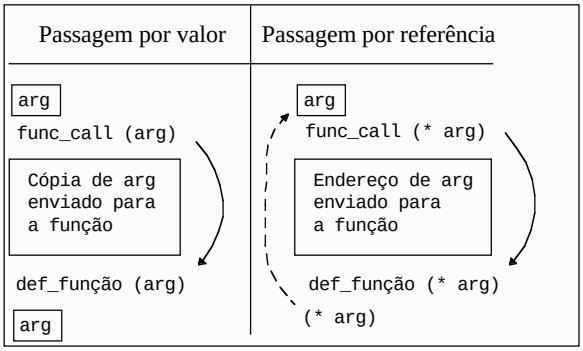

Passagem de Parâmetros

- 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 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*

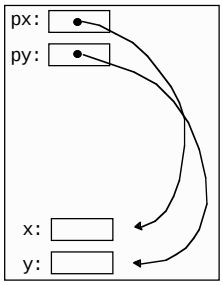
- Fazer uma função *troca(px, py)* que recebe como parâmetros 2 ponteiros para inteiros e troca o conteúdo deles
- ex:

```
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]

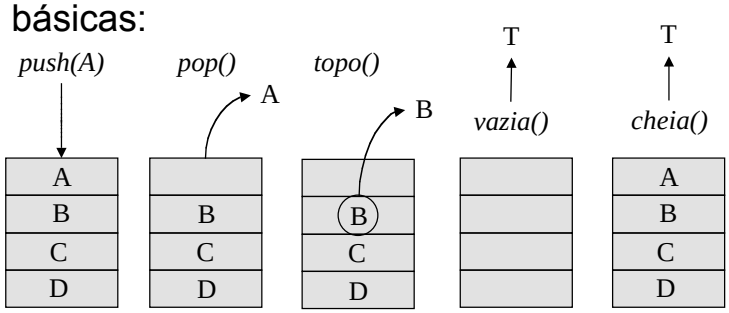
Achachar

```
char *achachar (char *str, char c) {
char *pc = str;

  while (*pc != c && *pc != '\0') pc++;
  return *pc ? pc : NULL;
}
```

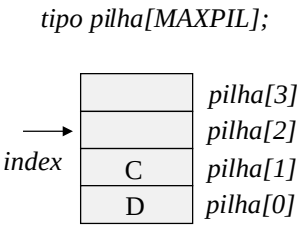
Exemplo: Pilha

- Um pilha é definida como uma estrutura onde os dados são inseridos em uma ordem e retirados em ordem inversa. Operações básicas:



Pilha com Array

- Uma pilha pode ser implementada como um array onde o topo é indicado com um índice




```
Topo():  index aponta para a posição
         vazia:
         retorna pilha[index-1];
Vazia:   retorna index == 0
Push(d): pilha[index] = d;
         incrementa index;
Pop():   decrementa index;
         retorna pilha[index];
Cheia(): returna index == MAXPIL+1
```

Pilha de Inteiros em C

```
#define MAXPIL 10
int pilha[MAXPIL];
int index = 0;

void push(int dado) { pilha[index++] = dado; }
int pop()   { return pilha[--index]; }
int topo()  { return pilha[index-1]; }
int vazia() { return (index == 0); }
int cheia() { return (index == MAXPIL+1; }
```


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 o 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()*

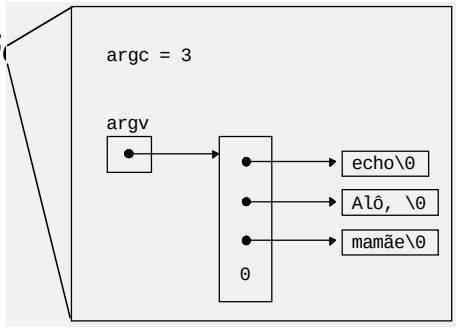
- ao executar programas a partir de linha de comando, é possível passar parâmetros diretamente para a função *main()*
- os argumentos são fornecidos na forma de um *array de strings*.
- *main()* é definida com dois parâmetros:

```
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");
}
```

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);
}
```

Exemplo Fatorial

```
fatorial(5)
    => (5 ≠ 0)
       return 5 • fatorial(4)
       => (4 ≠ 0)
          return 4 • fatorial(3)
          => (3 ≠ 0)
             return 3 • fatorial(2)
             => (2 ≠ 0)
                return 2 • fatorial(1)
                => (1 ≠ 0)
                   return 1 • fatorial(0)
                   => (0 == 0)
                   <=  return 1
                <= return 1 • 1   (1)
             <= return 2 • 1      (2)
          <=  return 3 • 2        (6)
       <= return 4 • 6            (24)
    <= return 5 • 24              (120)
120
```


Estrutura de uma Recursão

- uma recursão obedece a uma estrutura que deve conter os seguintes elementos:
- *função (par)*
  - *teste de término de recursão utilizando par*
    - *se teste ok, retorna aqui*
  - *processamento*
    - *aqui a função processa as informações em par*
  - *chamada recursiva em par'*
    - *par deve ser modificado de forma que a recursão chegue a um término*

Exemplo: *printd(int)*

- *printd(int)* imprime um inteiro usando recursão

```
void printd (int n) {
 if (n < 0) {          /* imprime sinal */
     putchar('-');
     n = -n;
 }
 if (n / 10)            /* termino recursao */
     printd(n/10);      /* recursao se n>10 */
 putchar(n % 10 + '0'); /* senao imprime char */
}
```


Exemplo *printd()*

```
printd(-1974)
==> (n < 0) –> putchar('-')                -
    ==>printd(197)                         -
        ==> printd(19)                     -
            ==> printd(1)                  -
             (1 / 10 == 0)                 -
            putchar(1 + '0')               -1
        putchar(19%10 + '0')               -19
    putchar(197%10 + '0')                  -197
putchar(1974 %10 + '0')                    -1974
```

Exemplo: *quicksort*

- *quicksort* é um exemplo de algoritmo recursivo para ordenar elementos em um *array*
- baseia-se em um processo de divisão e conquista:
  - ◊ divisão: o array A[*n*] é dividido em dois subarrays A_0 e A_1 de forma que para qualquer elemento $A_0[i]$, $A_1[j]$ temos:
    $$A_0[i] \leq A_1[j]$$
  - ◊ conquista: A_0 e A_1 são recursivamente ordenados

Quicksort

```
void quicksort (int *A, int inicio, int fim)
{
 int meio;

 if (inicio < fim) {
    meio = parte(A, inicio, fim);
    quicksort(A, inicio, meio);
    quicksort(A, meio+1, fim);
 }
}
```

Quicksort

```
void troca (int *A, int a, int b)
{
 int tmp;

 tmp = A[a];
 A[a] = A[b];
 A[b] = tmp;
}

int parte (int *A, int inicio, int fim)
{
 int x = A[inicio];
 int i = inicio - 1;
 int f = fim + 1;

 while(TRUE) {
   do --f; while(A[f] > x);
   do ++i; while(A[i] < x);
   if (i < f)
     troca(A, i, f);
   else
     return f;
 }
}
```


Prática

- 1. Escrever uma função recursiva que retorna o tamanho de um *string*, *rstrlen(char s[])*
- 2. Fazer uma função recursiva que conta o número de ocorrências de um determinado caracter, *caract(char c, char s[])*
- 3. Escreva uma função recursiva que produza o reverso de um *string*, *reverse(char s[])*

Macros

- definição de uma macro

```
#define nome_macro texto_equivalente
```

- a definição da macro vai até o fim da linha
- o caracter "\" permite continuar a definição da macro em outras linhas:

```
#define nome_macro texto_1 \
                   texto_2 \
                      ...
     /*até encontrar uma linha sem "\" */
```


Macros

- além de tornar o código mais legível, macros podem aumentar a velocidade de execução:

```
#define TAXA 0.3
...
imposto = preco * TAXA;
```

- é mais eficiente que:

```
float taxa = 0.3;
...
imposto = preco * taxa;
```

- pois o programa não precisa ler taxa da memória

Macros

```
#include <stdio.h>
#include <stdlib.h>
#define FIRST_CHAR 'O'
#define SECOND_CHAR 'i'
#define COMMA ','
#define STRING " este e um exemplo de macro."
void main() {
  putchar( FIRST_CHAR );
  putchar( SECOND_CHAR );
  putchar( COMMA );
  puts( STRING );
}
```

Oi, este e um exemplo de macro

Macros

- atenção as aspas:

```
#define STRING “Uma sequencia de caracteres.”
...
printf( “%s”, “STRING”);
```

==> ? STRING

Aninhamento de Macros

```
#define NUM_1 10
#define NUM_2 5
#define SOMA NUM_1 + NUM_2
...
printf ( “Resultado soma : %d”, SOMA);
```

- primeiro passo do pré-processador:

```
printf ( “Resultado soma : %d”, NUM_1 + NUM_2);
```

- segundo passo do pré-processador:

```
printf ( “Resultado soma : %d”, 10 + 5);
```


Macros em Formato de Função

```
#define nome(lista_parâmetros) texto

Ex:
#define MILES2KM(m) m*(8.0/5.0)
...
double milhas, kilometros;
milhas = 62.0;
kilometros = MILES2KM(milhas);
-- após expansão da macro:
kilometros = milhas*(8.0/5.0);
```

Macros em Formato de Função

- macros podem ter vários parâmetros separados por vírgulas:

```
#define SOMA(a, b) a + b
...
int i=10, j=20, k;
float f=5.5, g=2.3, h;

k = SOMA(i, j);
h = SOMA(f, g);
```


Prática

- Implemente a macro soma, testando a sua aplicação em dados tipo inteiro e float