Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

Funções em C

Material preparado pela profa
Silvana Maria Affonso de Lara

2º semestre de 2010

Roteiro da Aula

- Definição de Função
- Argumentos, retornos e protótipos
- Funcionamento de uma chamada
- Passagem de Informações
- Passagem de parâmetros por valor e por referência
- Pilha de inteiros em C
- Número de parâmetros variáveis
- Acesso aos parâmetros
- Parâmetros para main()

Função - Definição

- Agrupa um conjunto de comandos e associa a ele um nome
  - O uso deste nome é uma chamada da função
- Após sua execução, programa volta ao ponto do programa situado imediatamente após a chamada
  - A volta ao programa que chamou a função é chamada de retorno

FUNÇÃO

- A chamada de uma função pode passar informações (argumentos) para o processamento da função
  - Argumentos = lista de expressões
    - Lista pode ser vazia
    - Lista aparece entre parênteses após o nome da função
      - Ex.

```
int Soma (int x, int y) {
 }
```

O Retorno da Função

- No seu retorno, uma função pode retornar resultados ao programa que a chamou

```
return (resultados);
```

  - O valor da variável local *resultados* é passado de volta como o valor da função

- Valores de qualquer tipo podem ser retornados
  - Funções predicado: funções que retornam valores
  - Procedimentos: funções que não retornam valores

Exemplo:
```
void function (int x)
```

Funções

- Definições de funções
  - Funções são definidas de acordo com a seguinte sintaxe:

```
tipo_de resultado nome (lista de parâmetros)
{
        corpo de função
}
```

Funções - Exemplo

```
int MDC (int a, int b) {
    int aux;
    if (a < b) {
        aux = a;
        a = b;
        b = aux;
    }
    while (b != 0) {
        aux = b;
        b = a % b;
        a = aux;
    }
    return (a);
}
```

uma <u>função</u> C para calcular o máximo divisor comum entre dois números

Funções

- Definições de funções
  - Tipo de resultado
    - Quando a função é um procedimento, usa-se a palavra chave void
    - Procedimento não retorna valor
  - Lista de parâmetros
    - Funcionam como variáveis locais com valores iniciais
    - Quando função não recebe parâmetros, a lista de parâmetros é substituída pela palavra void

Funções

- Funcionamento de uma chamada:
  - Cada expressão na lista de argumentos é avaliada
  - O valor da expressão é convertido, se necessário, para o tipo de parâmetro formal
    - Este tipo é atribuído ao parâmetro formal correspondente no início do corpo da função
  - O corpo da função é executado

Funções

- Funcionamento de uma chamada:
  - Se um comando return é executado, o controle é passado de volta para o trecho que chamou a função
  - Se um comando return inclui uma expressão, o valor da expressão é convertido, se necessário, pelo tipo do valor que a função retorna
    - O valor então é retornado para o trecho que chamou a função
  - Se um comando return não inclui uma expressão nenhum valor é retornado ao trecho que chamou a função
  - Se não existir um comando return, o controle é passado de volta para o trecho que chamou a função após o corpo da função ser executado

Passagem de Informações

- Exemplo:

```
double mesada (double notas, int idade) {
      double total;

      if (idade > 10)
          return (idade * 20.0);
     else{
          total =  notas*idade*20;
          return total;
     }
}
```

Passagem de Informações

- Argumentos são passados **por valor**
  - Quando chamada, a função recebe o valor da variável passada
    - Quando argumento é do tipo atômico, a passagem por valor significa que a função não pode mudar seu valor
  - Os argumentos deixam de existir após a execução do método

FUNÇÕES

- Protótipos ou Declaração de funções
  - Antes de usar uma função em C, é aconselhável declará-la especificando seu protótipo
    - Tem a mesma forma que a função, só que substitui o corpo por um (;)
    - Nomes das variáveis de um parâmetro são opcionais
      - Fornecê-los ajuda a leitura do programa
  - A declaração apenas indica a *assinatura ou protótipo* da função:
      - ***valor_retornado* nome_função(*declaração_parâmetros*);**

Funções

- Mecanismo do processo de chamada de função
  1. Valor dos argumentos é calculado pelo programa que está chamando a função
  2. Sistema cria novo espaço para todas as variáveis locais da função (estrutura de pilha)
  3. Valor de cada argumento é copiado na variável parâmetro correspondente na ordem em que aparecem
     3.1 Realiza conversões de tipo necessárias

Funções

- Mecanismo do processo de chamada de função
  4. Comandos do corpo da função são executados até:
     4.1 Encontrar comando return
     4.2 Não existirem mais comandos para serem executados
  5. O valor da expressão return, se ele existe, é avaliado e retornado como valor da função
  6. Pilha criada é liberada
  7. Programa que chamou continua sua execução

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

FUNÇÕES

- Projeto *top-down*
  - Procedimentos e funções permitem dividir um programa em pedaços menores
    - Facilita sua leitura
  - É chamado de processo de decomposição
    - Estratégia de programação fundamental
    - Encontrar a decomposição certa não é fácil
      - Requer experiência

Funções

- Projeto *top-down*
  - Melhor estratégia para escrever programas é começar com o programa principal
  - Pensar no programa como um todo
  - Identificar as principais partes da tarefa completa:
    - Maiores pedaços são candidatos a funções
    - Mesmo essas funções podem ser decompostas em funções menores
    - Continuar até cada pedaço ser simples o suficiente para ser resolvido por si só

Exercício

```
int MDC (int a, b) {
    int aux;
    if (a < b) {
            aux = a;
            a = b;
            b = aux;
    }
    while (b !=  0){
            aux = b
            b = a % b;
            a = aux
    }
    return (a);
}
```

- Escrever um programa completo em C para calcular o máximo divisor comum entre dois números desmembrando o programa em pelo menos <u>três funções</u>

Funções

- 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.
}
```

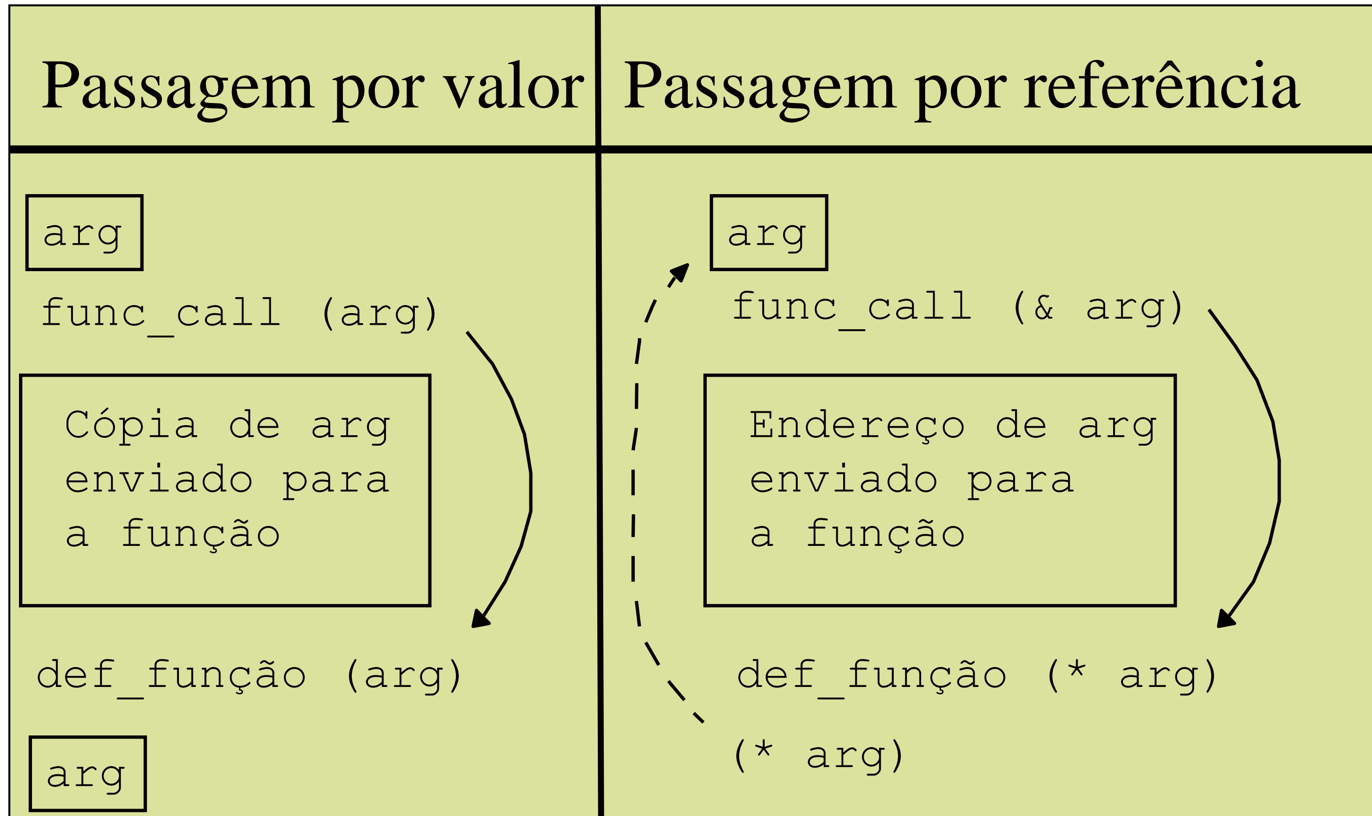
Passagem de Parâmetros

- 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*

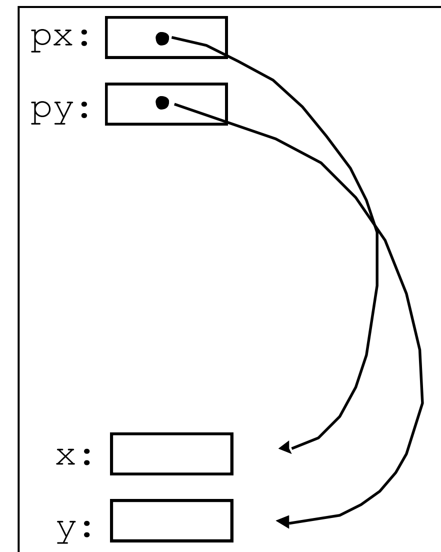
- Fazer uma função *troca(px, py)* que recebe como parâmetros 2 ponteiros para inteiros e troca o conteúdo deles
- ex:

```
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 */
```

Vetores como Argumentos de Funções

Quando vamos passar um vetor como argumento de uma função, podemos declarar a função de três maneiras equivalentes. Seja o vetor:

```
int matrx [50];
```

e que queiramos passá-la como argumento de uma função func(). Podemos declarar func() de três maneiras:

```
void func (int matrx[50]);
void func (int matrx[]);
void func (int *matrx);
```

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 endereço do terceiro caracter do *string*: &str[2]

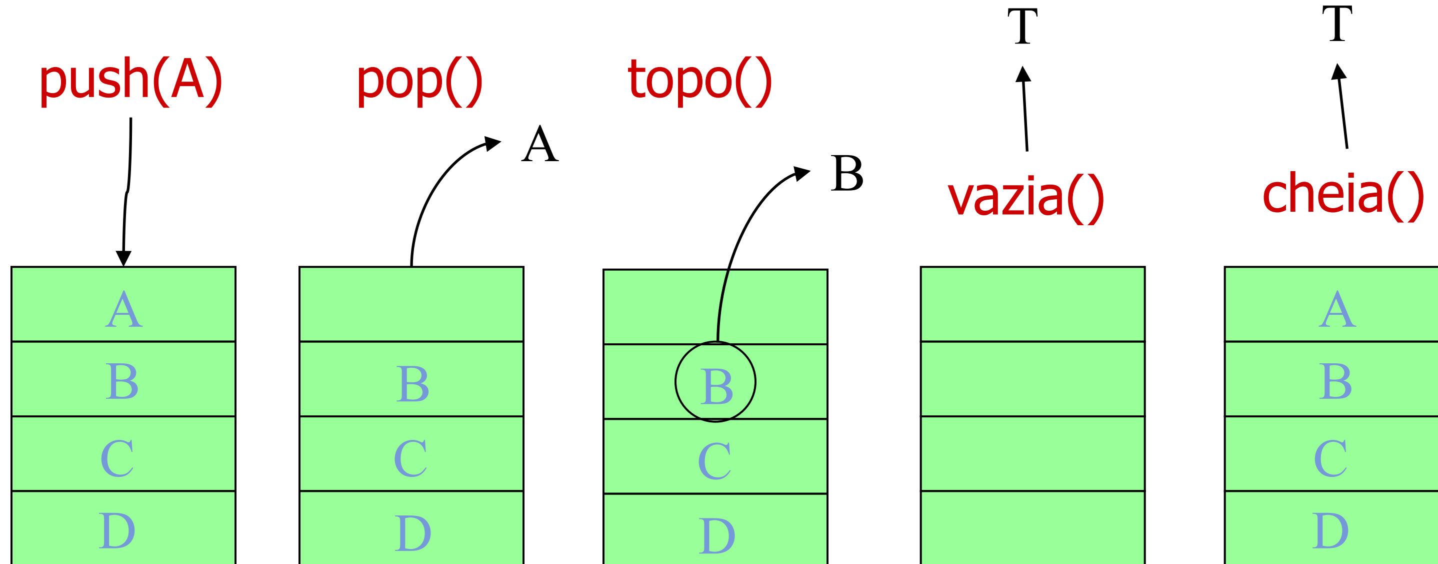
Achachar

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

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

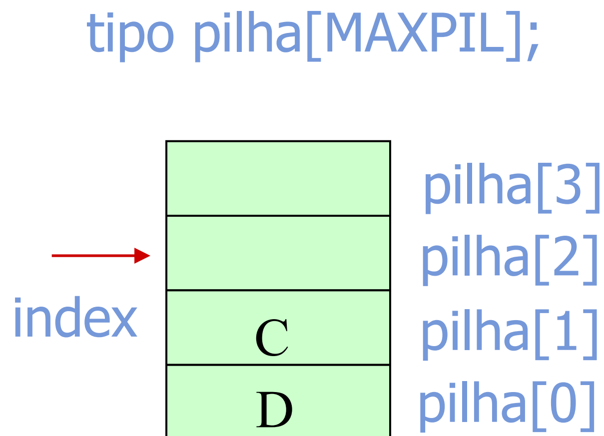
EXEMPLO: PILHA

- 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():	retorna 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, arg_def) inicia o ponteiro *ap* fazendo-o apontar para o primeiro argumento da lista, ou seja, o primeiro argumento depois de *arg_def*.

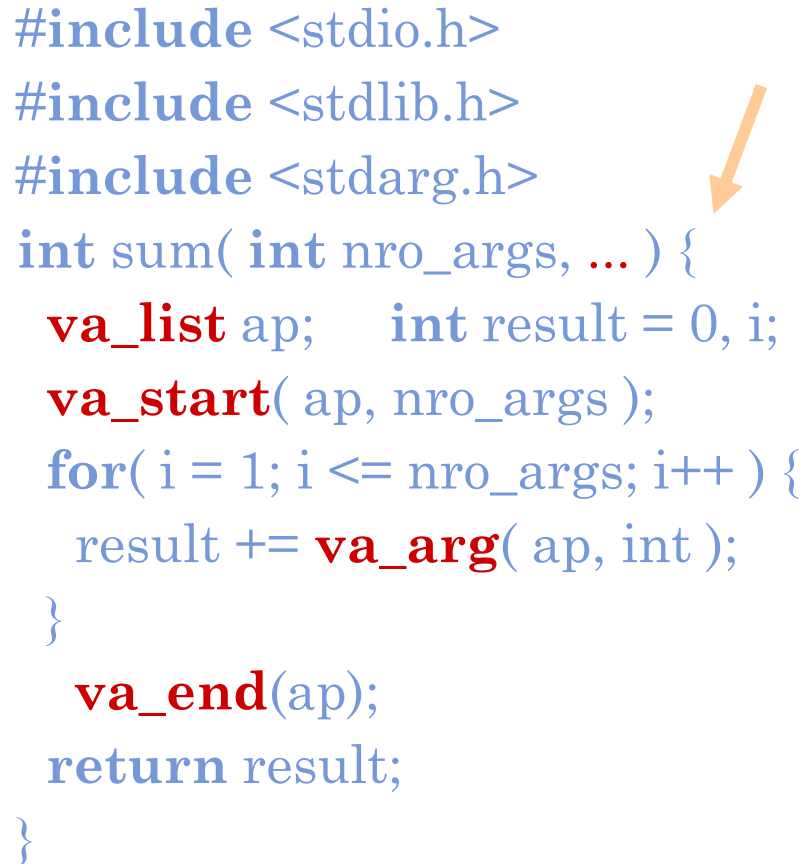
  - arg_def é o nome do último argumento especificado na declaração da função

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 nro_args, ... ) {
  va_list ap;     int result = 0, i;
  va_start( ap, nro_args );
  for( i = 1; i <= nro_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

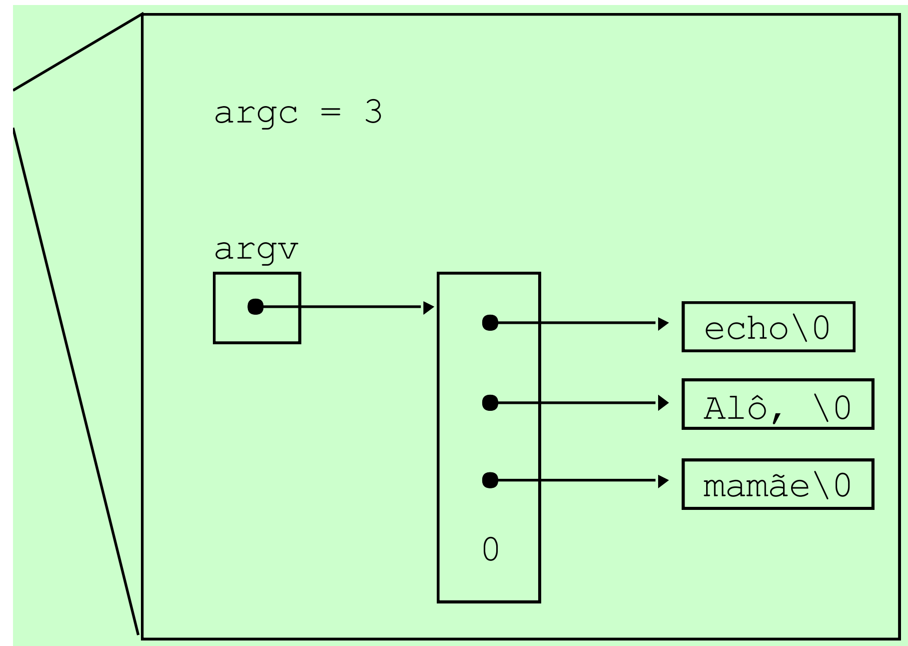
argv é o array de argumentos

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");
}
```

Exercício

- Fazer um programa em C que lê um array de caracteres e cria um outro array em que os caracteres estão totalmente invertidos, por exemplo, ler “MARCONI” e retornar “INOCRAM”
  - Usar ponteiros
  - Usar uma função que recebe um array e retorna o outro invertido