Aula 16

MC 102 - Algoritmos e Programação de Computadores

Funções e Procedimentos II:
parâmetros por valor e por referência

Parâmetros por valor

Ao invocarmos uma função, obrigatoriamente temos que definir os valores que serão seus parâmetros, usando variáveis ou constantes.

Este processo é semelhante a uma atribuição. Assim, caso algum desses parâmetros tenha seu valor alterado dentro da função, como este é considerado uma variável local à função, nada acontece com a variável que definiu seu valor na chamada da função.

Chamamos estes parâmetros de parâmetros por valor

Parâmetros por valor

```
#include <stdio.h>

void func(int a) {
  a = 2;
  printf("2:a = %d\n", a);
}

int main(int argc, char **argv){
  int a = 1;

  printf("1:a = %d\n", a);
  func(a);
  printf("3:a = %d\n", a);

  return 0;
}
```

O valor da variável local **a** em **func** é alterado, mas a variável local **a** em **main** continua com seu valor antes da chamada da função

Resultado:

1:a = 1
2:a = 2
3:a = 1

Parâmetros por referência

Algumas vezes, precisamos que uma determinada função altere o valor da variável utilizada como parâmetro. Neste caso, no lugar do valor da variável, passamos como parâmetro o **endereço de memória** dessa variável.

```
func(&a);
```

A função func deverá alterar o conteúdo de a, de modo semelhante ao procedimento scanf.

A este tipo de parâmetros damos o nome de parâmetros por referência. Ao o utilizarmos, o que importa não é o conteúdo da variável, e sim seu endereço de memória.

Parâmetros por referência

Mas como declaramos o tipo deste parâmetro na função?

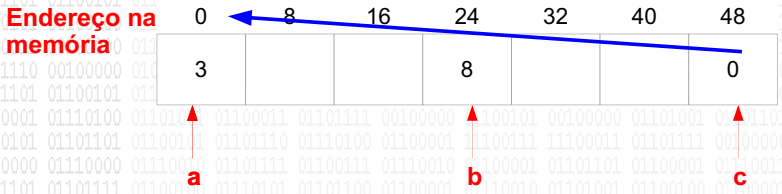
Em C, temos um tipo especial para variáveis que guardam endereço. Ele é representado pelo *

Exemplo:

```
char *a;  // a conterá o endereço de memória uma variável do tipo char
int *d; // d conterá o endereço de memória uma variável do tipo int
```


Parâmetros por referência

```
int a, b, *c;
a = 3;
b = a + 5;
c = &a;
```



Parâmetros por referência

Assim, na definição da função:

```
tipo  nome (tipo *parâmetro1, tipo *parâmetro2,
  ..., tipo *parâmetroN) {
      comandos;
}
```

Mas por que a passagem de parâmetros altera o valor da variável?

Parâmetros por referência

Assim, na definição da função:

```
tipo  nome (tipo *parâmetro1, tipo *parâmetro2,
  ..., tipo *parâmetroN) {
      comandos;
}
```

Mas por que a passagem de parâmetros altera o valor da variável?

Porque aqui o valor é alterado diretamente na posição da memória.

Parâmetros por referência

```
#include <stdio.h>

void func(int *a) {
  *a = 2;
  printf("2:a = %d\n", *a);
}

int main(int argc, char **argv){
  int a = 1;

  printf("1:a = %d\n", a);
  func(&a);
  printf("3:a = %d\n", a);

  return 0;
}
```

O valor da variável local ***a** em **func** é alterado, alterando também a variável local **a** em **main**

Resultado:

1:a = 1
2:a = 2
3:a = 2

Parâmetros por referência

Algumas considerações importantes

- Os parâmetros por referência devem sempre ser definidos usando um * antes do identificador

```
void func(int *a)
```

- A variável local definida pelo parâmetro deve ser referenciada usando * antes do identificador, da mesma forma que na sua definição

```
*a = 2;
```

- Para a chamada da função, é necessário que o parâmetro seja uma variável de mesmo tipo, e seu endereço deve ser informado

```
func(&a);
```


Exemplo – Troca de valores

Procedimento para troca de valores ente duas variáveis

```
#include <stdio.h>

void troca(int *a, int *b) {
  int aux;

  aux = *a;
  *a = *b;
  *b = aux;
}

int main(int argc, char **argv){
  int num1 = 1, num2 = 2;

  printf("%d %d\n", num1, num2);
  troca(&num1, &num2);
  printf("%d %d\n", num1, num2);

  return 0;
}
```

Resultado:

1 2
2 1

Vetores como parâmetros

Os vetores são sempre definidos como parâmetro por referência

```
#define TAM_MAX 30

void func(int vetor[TAM_MAX], int tamanho)
void func(int vetor[], int tamanho)
void func(int *vetor, int tamanho)
```

É possível não definir a dimensão do vetor (os dois últimos exemplos), mas deve-se tomar cuidado para não extrapolar a dimensão máxima permitida. Normalmente, o tamanho do vetor também é definido como parâmetro.

Um vetor não pode ser utilizado como resultado de uma função. Isso será resolvido utilizando ponteiros.

Exemplo – Imprimindo vetor

Procedimento para imprimir os valores de um vetor

```
#include <stdio.h>

#define TAM 10

void imp_vet(int *vetor, int tam) {
  int i;

  for(i=0; i < tam; i++)
    printf("%d ", vetor[i]);
  printf("\n");
}

int main(int argc, char **argv){
  int primos[TAM] = {1,2,3,5,7,11,13,17,19,23};

  imp_vet(primos, 5);
  imp_vet(primos, TAM);

  return 0;
}
```

Resultado:

```
1 2 3 5 7
1 2 3 5 7 11 13 17 19 23
```


Matrizes como parâmetros

Como os vetores, as matrizes também definidas como parâmetro por referência

```
#define TMAX 30

void func(int mat[TMAX][TMAX], int lins, int cols)
void func(int mat[][TMAX], int lins, int cols)
void func(int *mat[TMAX], int lins, int cols)
```

Só é permitido não definir a primeira dimensão da matriz (os dois últimos exemplos), mas as demais devem ser definidas. Em geral, o número de linhas e colunas, ou o tamanho de cada dimensão para vetores multidimensionais, são definidos como parâmetro.

Uma matriz, tal como um vetor, não pode ser utilizada como resultado de uma função pelo mesmo motivo.

Exemplo – Imprimindo matriz

Procedimento para imprimir uma matriz

```
#include <stdio.h>

#define TAM 3

void imp_mat(int mat[TAM][TAM], int nl, int nc) {
  int i, j;

  for(i=0; i < nl; i++) {
    for(j=0; j < nc; j++)
      printf("%d ", mat[i][j]);
    printf("\n");
  }
  printf("\n");
}

int main(int argc, char **argv){
  int ident[TAM][TAM] = {{1,0,0},{0,1,0},{0,0,1}};

  imp_mat(ident, TAM, TAM);
  return 0;
}
```

Resultado:

```
1 0 0
0 1 0
0 0 1
```


Strings como parâmetros

Strings são definidas de maneira semelhante a vetores. No entanto, não há a necessidade de se definir o tamanho da string, uma vez que o caracter '\0' marca o final da cadeia de caracteres.

```
void func(char *str)
void func(char str[])
```

Da mesma forma que vetores, strings não devem ser utilizadas como resultado de uma função.

Exemplo – Inicio de uma string

Procedimento para copiar parte inicial de uma string

```
#include <stdio.h>

void inicio(char *dest, char *orig, int tam) {
  int i;
  for(i=0; i < tam; i++)
    dest[i] = orig[i];
  dest[i] = '\0';
}

int main(int argc, char **argv){
  char str1[] = "Primeiro Segundo";
  char str2[9];
  inicio(str2, str1, 8);
  printf("%s\n", str1);
  printf("%s\n", str2);
  return 0;
}
```

Resultado:

Primeiro Segundo
Primeiro

Problema 1

O escritor Pedro Bandeira utilizou em alguns de seus livros um código para criptografar mensagens. Implemente uma função que receba duas strings como parâmetro (origem, destino) e retorne em destino a string de origem criptografada com o código tenis-polar, efetuando as trocas múltuas dos seguintes caracteres:

T <-> P E <-> O N <-> L I <-> A S <-> R

Problema 2

Implemente uma função para retornar o valor aproximado de π, de acordo com a fórmula de Leibniz:

$$\pi \cong 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} - \frac{4}{9} + \ldots \Longleftrightarrow \pi \cong 4 \sum_{i=0}^{n} \frac{(-1)^i}{2i+1}$$

Problema 3

Escreva uma função que verifique se uma string representa um número inteiro. A sua função deve retornar 1 caso a string represente um número inteiro, e 0 caso contrário: