



Linguagem de Programação I

FUNÇÕES E PONTEIROS

FUNÇÕES

Escopo: Um código de uma função é privativo àquela função e não pode ser acessado por nenhum comando em outra função, exceto por meio de uma chamada à função.

Variável local: são definidas internamente a uma função. É criada quando ocorre a entrada da função e é destruída a sair.

Variável Global: variáveis visíveis pelo programa todo. Elas são declaradas fora da função `main()` e fora de qualquer outra função.

float n1, n2, media; —————> Variáveis globais

```
void media_notas();
```

```
int main()
```

```
{
```

```
    printf("Cálculo de media de duas notas\n");
```

```
    printf("\nInforme a primeira nota");
```

```
    scanf("%f", &n1);
```

```
    printf("\nInforme a segunda nota");
```

```
    scanf("%f", &n2);
```

```
    media_notas();
```

```
    printf(" \nA media das notas: %.2f ", media);
```

```
}
```

```
void media_notas()
```

```
{
```

```
    media=(n1+n2)/2;
```

```
}
```

ARGUMENTOS PARA FUNÇÕES

Tipo de
retorno

```
int  acha(char *s, char c)
{
    while (*s)
        if (*s==c)
            return 1;
        else
            s++;
    return 0;
}
```

Parâmetro formal: se comportam como variáveis locais dentro da função e são criadas na entrada e destruídas na saída.

Você deve assegurar que os argumentos usados na chamada da função sejam compatíveis com os tipos de seus parâmetros.

CHAMADA POR VALOR

Este método de passagem de argumentos copia o valor de um argumento no parâmetro formal da sub-rotina. Assim, alterações feitas nos parâmetros da função não tem efeito nenhum nas variáveis usadas para chamá-la.

```
int sqr(int x)
{
    x=x*x;
    return x;
}
int main()
{
    int t=10;
    printf("%d  %d",sqr(t), t);
}
```

CHAMADA POR REFERÊNCIA

Neste método, o endereço de um argumento é copiado no parâmetro. Dentro da função, o endereço é usado para acessar o argumento real utilizado na chamada.

Então, pode-se criar uma chamada por referência passando um ponteiro como argumento.

Ponteiros são passados para as funções como qualquer outra variável. É necessário declarar os parâmetros como do tipo ponteiro.

CHAMADA POR REFERÊNCIA

```
void troca(int *x, int *y)
```

```
{
```

```
    int aux;
```

```
    aux=*x;
```

```
    *x=*y;
```

```
    *y=aux;
```

```
}
```

```
i=10
```

```
j=20
```

```
Depois da troca:
```

```
i=20
```

```
j=10
```

```
int main()
```

```
{
```

```
    int i=10,j=20;
```

```
    printf("i=%d      j=%d",i,j);
```

```
    troca(&i,&j);
```

```
    printf("\nDepois      da      troca:      \n\ni=%d
```

```
j=%d",i,j);
```

```
    return 0;
```

```
}
```

POR VALOR

```
#include <stdio.h>

int quadradonum1(int num1)
{   return num1*num1; }

int quadradonum2(int num2)
{   return num2*num2; }

int main()
{
    int num1, num2;

    printf("Entre com o primeiro número ");
    scanf("%i",&num1);
    printf("Entre com o segundo número ");
    scanf("%i",&num2);

    printf("\nAntes da chamada da funcao Quadrado os numeros sao %i e %i
\n",num1,num2);
    printf("\nDepois da chamada da funcao Quadrado os numeros sao %i e %i
\n",num1,num2);
    printf("Quadrado dois números: %i e
%i",quadradonum1(num1),quadradonum2(num2));
    return 0;
}
```


POR REFERÊNCIA

```
#include <stdio.h>
void quadrado(int *num1, int *n);
int main()
{
    int num1, num2;
    printf("Entre com o primeiro número ");
    scanf("%i",&num1);
    printf("Entre com o segundo número ");
    scanf("%i",&num2);
    printf("\nAntes da chamada da funcao Quadrado os numeros sao %i e %i\n",num1,num2);

    quadrado(&num1,&num2);

    printf("\nDepois da chamada da funcao Quadrado os numeros sao %i e %i\n",num1,num2);
    return 0;
}

void quadrado(int *num1, int *n)
{
    *num1=  *num1 * *num1;
    *n=  *n * *n;

    return;
}
```

PONTEIROS E MATRIZES

Passar um vetor para uma função consiste em passar o endereço da primeira posição do vetor.

Se passarmos um valor de endereço, a função chamada deve ter um parâmetro do tipo ponteiro para armazenar este valor.
Ex.: $\text{int}^* \rightarrow$ Armazena endereços de inteiros.

Salientamos que a expressão “passar um vetor para uma função” deve ser interpretada como “passar o endereço inicial do vetor”.

Os elementos do vetor não são copiados para a função, o argumento copiado é apenas o endereço do primeiro elemento.

```
#include <stdio.h>
```

```
float media(float *,int tamanho);
```

```
float media1(float notas[50],int tamanho);
```

```
int main()
```

```
{
```

```
    float notas[50], m, m1;
```

```
    int i=-1;
```

```
    do
```

```
    {
```

```
        i++;
```

```
        printf("Digite a nota do aluno %d ", i+1);
```

```
        scanf("%f",notas+i);
```

```
    } while (*(notas + i) > 0.0);
```

```
    m=media(notas,i); // como o nome da matriz é um endereço  
                    // não usamos & nesta instrução
```

```
    m1=media1(notas,i);
```

```
    printf("Média das notas: %.3f \n",m);
```

```
    printf("Média(1) das notas: %.3f \n",m1);
```

```
    return 0;}
```

```
float media(float *lista,int tamanho)
{
    int i;
    float m=0.0;
    for (i=0;i<tamanho;i++)
        m+= *(lista++);
    return m/tamanho;
}
```

```
float media1(float notas[50],int tamanho)
{
    int i;
    float m=0.0;
    for (i=0;i<tamanho;i++)
        m+= notas[i];
    return m/tamanho;
}
```

PONTEIROS E STRINGS

Strings são matrizes do tipo char . Dessa forma a notação ponteiro pode ser aplicada.

Vamos escrever uma função que retorna o endereço da primeira ocorrência do character, se este existir, ou o endereço zero, caso o character não seja encontrado

```
#include <stdio.h>
```

```
char *procura(char *s, char ch)
```

```
{
```

```
    while (*s != ch && *s != '\0')  
        s++;
```

```
    if (*s != '\0')
```

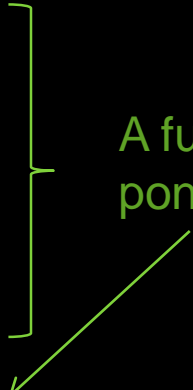
```
        return s;
```

```
    else
```

```
        return (char *)0;
```

```
}
```

A função retorna um
ponteiro char



Para retornar um ponteiro, deve-se declarar
uma função como tendo tipo de retorno
ponteiro.

```
int main()
{
    char str[81], *ptr;

    printf("Digite uma frase: \n");
    gets(str);

    ptr=procura(str, 'h');

    printf("\nA frase começa no endereco %p\n", ptr);

    if (ptr)
    {
        printf("\nPrimeira ocorrencia do caractere 'h': %p\n", ptr);
        printf("\nA sua posição eh: %d\n", ptr-str);
    }
    else
    {
        printf("O caracter 'h' nao existe nessa frase\n");
    }
    return 0;
}
```

MATRIZES E STRINGS

```
char *p = "alo mundo";
```

O comando acima funciona porque o compilador C cria o que é chamada de "tabela de string" usada para armazenar constantes strings usadas pelo programa.

Assim, o comando acima, coloca o endereço de "alo mundo" armazenado na tabela de string, no ponteiro p.

p pode ser usado no programa como qualquer outra string.

MATRIZES E STRINGS

Qual a saída:

```
int main()
```

```
{
```

```
    char *p = "lo mundo";
```

```
    char m[]={ 'O', 'I', 'E', 'E', 'E' };
```

```
    printf("\n\tEndereco de P:    %p ",p);
```

```
    printf("\n\t%s",p);
```

```
    printf("\n\n\tEndereco de M   : %p ",m);
```

```
    p++;
```

//m++; → atribuicao para constante não permitida

```
    printf("\n\n\tEndereco de P++: %p ",p);
```

```
    printf("\n\t%s",p);
```

```
    printf("\n\n\t%s",m);
```

```
}
```

MATRIZES E STRINGS

Endereco de P: 0x402020
alo mundo

Endereco de M : 0x22cd10

Endereco de P++: 0x402021
lo mundo

OIEEE

RETORNO DE MATRIZ EM FUNÇÃO

```
int main()
{
    int num,i;
    int A[10],*M; //matriz

    for (i = 0; i < 10; i++)
    {
        printf("Informe o elemento %i:  ", i + 1);
        scanf("%i", &A[i]);
    }
    printf("\nInforme um numero para multiplicacao: ");
    scanf("%i", &num);

    M=Multiplica(A,num);

    for (i=0;i<10;i++)
    {
        printf("\n A[%i] = %i X %i = M[%i] =  %i",
i,A[i],num,i,M[i]);
    }

    return 0;
}
```

RETORNO DE MATRIZ EM FUNÇÃO

```
int *Multiplica(int A[10],int num)
{
    int i;
    int *M = (int *) malloc(10 * sizeof
(int));

    for (i=0;i<10;i++)
    {
        M[i] = A[i]*num;
    }

    return M;
}
```

REFERÊNCIA BIBLIOGRÁFICA

Mizrahi, Victorine Viviane. Treinamento em Linguagem C. São Paulo: Editora Pearson, 2008, 2ª edição.

Schildt, Hebert. C completo e Total. São Paulo: Makron Books, 1996.

W. Celes e J. L. Rangel, Vetores e Alocação Dinâmica. Puc-Rio. Apostila.