



Introdução à Programação

Capítulo VI ***Arrays e Strings***

Engenharia Informática

Arrays unidimensionais

Definição

Um *array* é uma estrutura de dados que contém um determinado número de valores, todos do mesmo tipo. Estes valores, designados por **elementos** do *array*, podem ser individualmente acedidos através da sua posição dentro do *array*.

Os *arrays* mais simples são os unidimensionais (também designados por *vectores*). Os elementos de um *array* unidimensional estão conceptualmente organizados uns a seguir aos outros numa única linha. Pode visualizar-se um *array* unidimensional de nome **a**, da seguinte forma:



Como definir variáveis do tipo array

```
tipo_dos_elementos nome_variável[nº_de_elementos];
```

- **tipo_dos elementos:** tipo de dados de cada um dos elementos do *array*
- **nome_variável:** indica o nome pelo qual o *array* vai ser conhecido
- **nº_de_elementos:** valor constante que indica quantos elementos tem o *array*

Nota: Um *array* pode conter elementos de qualquer tipo. No entanto, os elementos de um dado *array* têm que ser **obrigatoriamente do mesmo tipo**, o qual é definido na declaração do mesmo.

Exemplos

```
int a[10];      /* a é um array com 10 elementos inteiros */
```

```
float b[100]; /* b é um array com 100 elementos reais */
```

Considere a seguinte declaração:

```
#define N 50
```

```
(...)
```

```
float notas[N];      /* notas é um array com 50 elementos reais */
```

- **float:** tipo de cada um dos elementos do *array*
- **N:** nº de elementos do *array*
- **notas:** nome do *array*
- **notas[i]:** conteúdo da posição **i** do *array* **notas**

Suponha que se pretendia declarar um *array* com 6 inteiros chamado **tabela**:

```
int tabela[6];
```

tabela[0]	tabela[1]	tabela[2]	tabela[3]	tabela[4]	tabela[5]

Nota: em C os índices de um *array* com **N** elementos variam sempre entre **0** e **N-1**.

Cada uma das 6 posições do *array* **tabela**, pode ser acedida através do respectivo **índice** colocado entre parêntesis rectos (**[]**).

Nota: O índice do primeiro elemento de qualquer *array* em C é sempre **0** (zero).

tabela[0]=5;

5					
tabela[0]	tabela[1]	tabela[2]	tabela[3]	tabela[4]	tabela[5]

tabela[5]=tabela[0]*5;

5					25
tabela[0]	tabela[1]	tabela[2]	tabela[3]	tabela[4]	tabela[5]

tabela[2]=tabela[0]+tabela[5];

5		30			25
tabela[0]	tabela[1]	tabela[2]	tabela[3]	tabela[4]	tabela[5]

Nota: Num *array*, o **i-ésimo** elemento está sempre na posição **i-1**.

À posição que um dado elemento ocupa no *array*, dá-se o nome de **índice** desse elemento.

Considere novamente o *array* **tabela** declarado anteriormente, mas inicializando agora cada posição, com o índice dessa mesma posição:

```
for (i=0 ; i<=5 ; i++)  
  
    tabela[i] = i;
```

0	1	2	3	4	5
tabela[0]	tabela[1]	tabela[2]	tabela[3]	tabela[4]	tabela[5]

O índice de um elemento pode ser qualquer expressão que resulte num valor inteiro:

```
tabela[1]=2;                /* 1 é um inteiro */  
tabela[3-1]=20;             /* equivalente a tabela[2]=20 */  
tabela[1+tabela[1]]=100;    /* equivalente a tabela[1+2]=100 pois  
                             tabela[1] contém um inteiro (igual a 2) */
```

Nota: Não existe verificação dos limites de um *array* pelo compilador, pelo que é possível aceder a um elemento que não exista! Cabe ao programador evitar que tal aconteça.

```
tabela[10]=15;              /* Possível, mas de evitar! */
```

Inicialização explícita de *arrays*

É possível inicializar todos os elementos de um array, na altura da sua declaração, usando a sintaxe:

```
tipo var[N] = {valor1, valor2, ..., valorN};
```

Exemplo

Declaração e inicialização de um *array* com todas as vogais do alfabeto:

```
#define N 5  
  
char vogal[N] = { 'a', 'e', 'i', 'o', 'u' };
```

Esta forma de inicialização evita o seguinte código:

```
char vogal[N];  
vogal[0]='a';  
vogal[1]='e';  
vogal[2]='i';  
vogal[3]='o';  
vogal[4]='u';
```

Se um vector for declarado com N elementos e os valores de inicialização forem apenas K ($K < N$), então os primeiros K elementos serão inicializados com os K valores e os restantes com zero.

Assim, as instruções seguintes são equivalentes:

```
#define N 8
```

```
int a[N]={5,8,1};
```

```
int a[N]={5,8,1,0,0,0,0,0};
```

Quando a declaração de um *array* é seguida de inicialização, pode omitir-se o nº de elementos:

```
tipo var[] = {valor1, valor2, ..., valorN};
```

O compilador vai criar um *array* com um nº de elementos igual ao nº de valores de inicialização.

Exemplos

Qual o resultado das seguintes declarações?

```
int v[10];  
int v[3]={5,10,15};  
int v[10]={5,10,15};  
int v[]={5,10,15};  
int v[];
```


Características dos *arrays*

- Os elementos de um *array* são sempre armazenados em localizações contíguas de memória.
- Os elementos de um *array* declarado sem qualquer inicialização contêm valores aleatórios (lixo).
- O índice do 1º elemento de um *array* é sempre zero.
- Os índices de um *array* com N elementos variam sempre entre 0 e N-1.
- O valor existente numa posição do *array* **a**, pode ser acedido através do nome do *array*, seguido do índice referente à posição, entre parêntesis rectos: **a[índice]**.
- O compilador não verifica se os índices utilizados num *array* com **N** elementos, estão ou não dentro da gama **0..N-1**. Por exemplo, há muita tendência para utilizar o índice **N** (**a[N]**), que não pertence ao *array*. Tal erro pode causar problemas graves pois estamos a aceder a uma localização de memória que não nos pertence.
- Um *array* pode ser explicitamente inicializado na altura da sua declaração, através de um conjunto de valores colocados dentro de chavetas, a seguir ao sinal **=**.
- Se o nº de valores de inicialização for menor do que o nº de elementos do *array*, os elementos em 'excesso' são inicializados com zero.

- Pode declarar-se um *array* sem se indicar a sua dimensão (nº de elementos), desde que a declaração seja seguida de uma inicialização explícita (deste modo o compilador calcula automaticamente o nº de elementos que o *array* irá conter).
- Não se podem declarar *arrays* sem dimensão, excepto na situação referida anteriormente.

Cópia de *arrays*

```
#define N 10
    (...)
int a[N], b[N];
    (...)
b=a;          /* Incorrecto! */
```

A cópia tem que ser efectuada **elemento a elemento**:

```
for (i=0; i<N; i++)
    b[i]= a[i];    /* Correcto! */
```

Exemplo 1

```
/* Calcular, para cada um dos 10 alunos de uma escola, a distância a que  
a sua nota está da nota média da classe. As notas individuais dos alunos  
são indicadas pelo utilizador. */
```

```
/* Versão 1: Sem Utilizar Tabelas */
```

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

```
void main()
```

```
{
```

```
    int n1, n2, n3, n4, n5, n6, n7, n8, n9, n10;  
    int media=0;
```

```
    printf("Nota do aluno 1: ");
```

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

```
    printf("Nota do aluno 2: ");
```

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

```
    printf("Nota do aluno 3: ");
```

```
    scanf("%d", &n3);
```

```
    printf("Nota do aluno 4: ");
```

```
    scanf("%d", &n4);
```

```
    printf("Nota do aluno 5: ");
```

```
    scanf("%d", &n5);
```

```
printf("Nota do aluno 6: ");
scanf("%d", &n6);
printf("Nota do aluno 7: ");
scanf("%d", &n7);
printf("Nota do aluno 8: ");
scanf("%d", &n8);
printf("Nota do aluno 9: ");
scanf("%d", &n9);
printf("Nota do aluno 10: ");
scanf("%d", &n10);
```

```
media = (n1+n2+n3+n4+n5+n6+n7+n8+n9+n10)/10;
```

```
printf("Nota media: %d\n", media);
printf("Diferenca no aluno 1: %d\n", n1-media);
printf("Diferenca no aluno 2: %d\n", n2-media);
printf("Diferenca no aluno 3: %d\n", n3-media);
printf("Diferenca no aluno 4: %d\n", n4-media);
printf("Diferenca no aluno 5: %d\n", n5-media);
printf("Diferenca no aluno 6: %d\n", n6-media);
printf("Diferenca no aluno 7: %d\n", n7-media);
printf("Diferenca no aluno 8: %d\n", n8-media);
printf("Diferenca no aluno 9: %d\n", n9-media);
printf("Diferenca no aluno 10: %d\n", n10-media);
}
```

```

/* Versão 2: Utiliza arrays */

#include <stdio.h>

#define AL 10

int main()
{
    int notas[AL], i, media = 0;

    for(i=0; i<AL; i++)
    {
        printf("Nota do aluno %d: ", i+1);
        scanf("%d", &notas[i]);
        media += notas[i];
    }
    media/=AL;
    printf("A media e: %d\n", media);
    for(i=0; i<AL; i++)
        printf("Diferenca no aluno %d: %d\n", i+1, notas[i]-media);
    return 0;
}

```

Exemplo de execução:

```
Nota do aluno 1: 12
Nota do aluno 2: 11
Nota do aluno 3: 17
Nota do aluno 4: 15
Nota do aluno 5: 13
Nota do aluno 6: 14
Nota do aluno 7: 19
Nota do aluno 8: 10
Nota do aluno 9: 12
Nota do aluno 10: 11
A media e: 13
Diferenca no aluno 1: -1
Diferenca no aluno 2: -2
Diferenca no aluno 3: 4
Diferenca no aluno 4: 2
Diferenca no aluno 5: 0
Diferenca no aluno 6: 1
Diferenca no aluno 7: 6
Diferenca no aluno 8: -3
Diferenca no aluno 9: -1
Diferenca no aluno 10: -2
```

Exemplo 2

```
/* Escrever uma sequência de números reais, introduzidos pelo
utilizador, por ordem inversa. O tamanho máximo da sequência é 10
(termina quando surgir o 0). */
```

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

```
#define TAM 10
```

```
int main()
```

```
{
```

```
    float a[TAM], num;
```

```
    int i, tamanho = 0;
```

```
    do{
```

```
        printf("Numero: ");
```

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

```
        if(num != 0)
```

```
            a[tamanho++] = num;
```

```
    }while((num != 0) && (tamanho < TAM));
```

```
    for(i = tamanho-1; i >=0 ; i--)
```

```
        printf("%3.2f\n", a[i]);
```

```
    return 0;
```

```
}
```

Exemplo de execução:

```
Numero: 23.5  
Numero: 2.43  
Numero: 1.2  
Numero: 7.8  
Numero: 0  
7.80  
1.20  
2.43  
23.50
```


Passagem de *arrays* para funções

Considere-se a seguinte função **f**, a qual recebe um *array* unidimensional como argumento:

```
int f(int a[], int n)
{
    ...
}
```

Em C, uma função não tem forma de saber com quantos elementos foi declarado um determinado *array* passado como argumento. Por este motivo, a **dimensão do *array*** terá que ser passada como um **argumento adicional**.

Ao contrário das outras variáveis, a passagem de *arrays* como argumentos para as funções é feita **por referência** e não por valor. Ou seja, o que é efectivamente passado para uma função não é uma cópia do *array*, mas sim o endereço inicial da zona de memória onde o *array* se encontra armazenado. Assim, é possível a uma função, alterar directamente o conteúdo de um *array* passado como argumento.

Os exemplos que se seguem, ilustram a utilização de *arrays* unidimensionais como argumentos.

Exemplo 3

```
/* Dado um array 'a' de valores inteiros, a função SomaArray devolve a soma dos seus elementos. */
```

```
#define TAM 100
```

```
int SomaArray(int a[], int n)
{
    int i, soma=0;
    for(i=0;i<n;i++)
        soma+=a[i];
    return soma;
}
```

```
/* Na chamada a esta função o 1º argumento será o nome do array e o 2º argumento o seu tamanho. */
```

```
void main()
{
    int b[TAM], total;
    ...
    total=SomaArray(b, TAM);
    ...
}
```

Exemplo 4

```
/* Função para verificar se uma tabela de inteiros está ordenada de
forma crescente. Quando a função for chamada a tabela já está
completamente preenchida. Deve devolver 1 se a tabela estiver ordenada
(0 se não estiver). */
```

```
#define TAM 10
int verifica(int a[], int n)
{
    int i;

    for(i=0; i<n-1; i++)
        if(a[i] > a[i+1])
            return 0;
    return 1;
}
```

```
/* Função auxiliar que preenche completamente a tabela com valores
indicados pelo utilizador */
```

```
void preenche_tabela(int a[], int n)
{
    int i;

    for(i=0; i<n; i++)
    {
        printf("Valor para a posicao %d: ", i);
        scanf("%d", &a[i]);
    }
}
```

```
/* Exemplo de utilização da função verifica */
```

```
int main()
{
    int tab[TAM];
    preenche_tabela(tab, TAM);
    if(verifica(tab, TAM))
        printf("Tabela Ordenada\n");
    else
        printf("Tabela nao ordenada\n");
    return 0;
}
```

Exemplo de execução:

```
Valor para a posicao 0: 2
Valor para a posicao 1: 3
Valor para a posicao 2: 7
Valor para a posicao 3: 89
Valor para a posicao 4: 90
Valor para a posicao 5: 101
Valor para a posicao 6: 101
Valor para a posicao 7: 102
Valor para a posicao 8: 106
Valor para a posicao 9: 110
Tabela Ordenada
```

Exemplo 5

`/* Programa que lê uma sequência de salários do pessoal de uma empresa e calcula o salário médio. O utilizador informa, no início da execução, quantos funcionários vão ser considerados. Utiliza 2 funções: uma que obtém os salários e outra que calcula o salário médio.*/`

```
#include <stdio.h>
#define N 100

float SalarioMedio(float s[],int n)
{
    int i;
    float smedio=0;

    for(i=0;i<n;i++)
        smedio+=s[i];
    smedio/=n;
    return smedio;
}
```

```

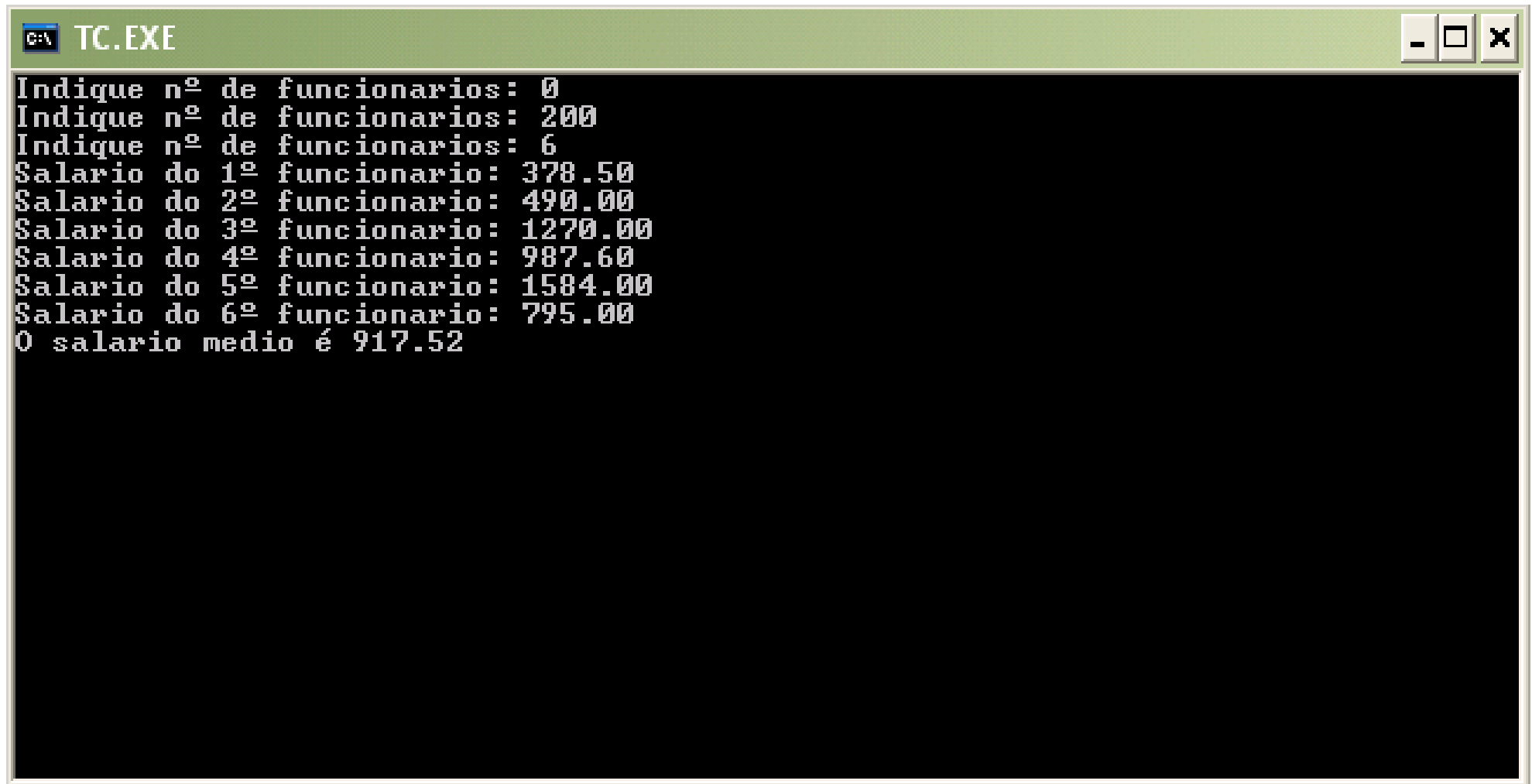
void LeSalarios(float s[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        printf("Salario do %dº funcionario: ",i+1);
        scanf("%f",&s[i]);
    }
}

void main()
{
    float salarios[N];
    int nfuncs;

    do{
        printf("Indique nº de funcionarios: ");
        scanf("%d",&nfuncs);
    }while(nfuncs<1 || nfuncs>N);
    LeSalarios(salarios,nfuncs);
    printf("O salario medio , %.2f\n",SalarioMedio(salarios,nfuncs));
}

```

Exemplo de execução:



```
C:\ TC.EXE
Indique nº de funcionarios: 0
Indique nº de funcionarios: 200
Indique nº de funcionarios: 6
Salario do 1º funcionario: 378.50
Salario do 2º funcionario: 490.00
Salario do 3º funcionario: 1270.00
Salario do 4º funcionario: 987.60
Salario do 5º funcionario: 1584.00
Salario do 6º funcionario: 795.00
O salario medio é 917.52
```


Exemplo 6

```
/* Programa que determina qual o valor máximo de uma sequência de
números inteiros introduzidos pelo utilizador (em que o valor zero
assinala o fim da sequência). Utiliza 2 funções: uma que obtém os
números do utilizador e outra que calcula o valor máximo dos números
introduzidos */
```

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

```
#define N 100
```

```
int LeNumeros(int n[],int dim)
```

```
{
```

```
    int i=0,cont=0;
```

```
    printf("Numero: ");
```

```
    scanf("%d",&n[i]);
```

```
    while(n[i] && i<dim)
```

```
    {
```

```
        cont++;
```

```
        printf("Numero: ");
```

```
        scanf("%d",&n[++i]);
```

```
    }
```

```
    return cont;
```

```
}
```

```

int CalculaMaximo(int n[],int tam)
{
    int i,max;

    max=n[0];

    for(i=0;i<tam;i++)
        if(n[i]>max)
            max=n[i];
    return max;
}

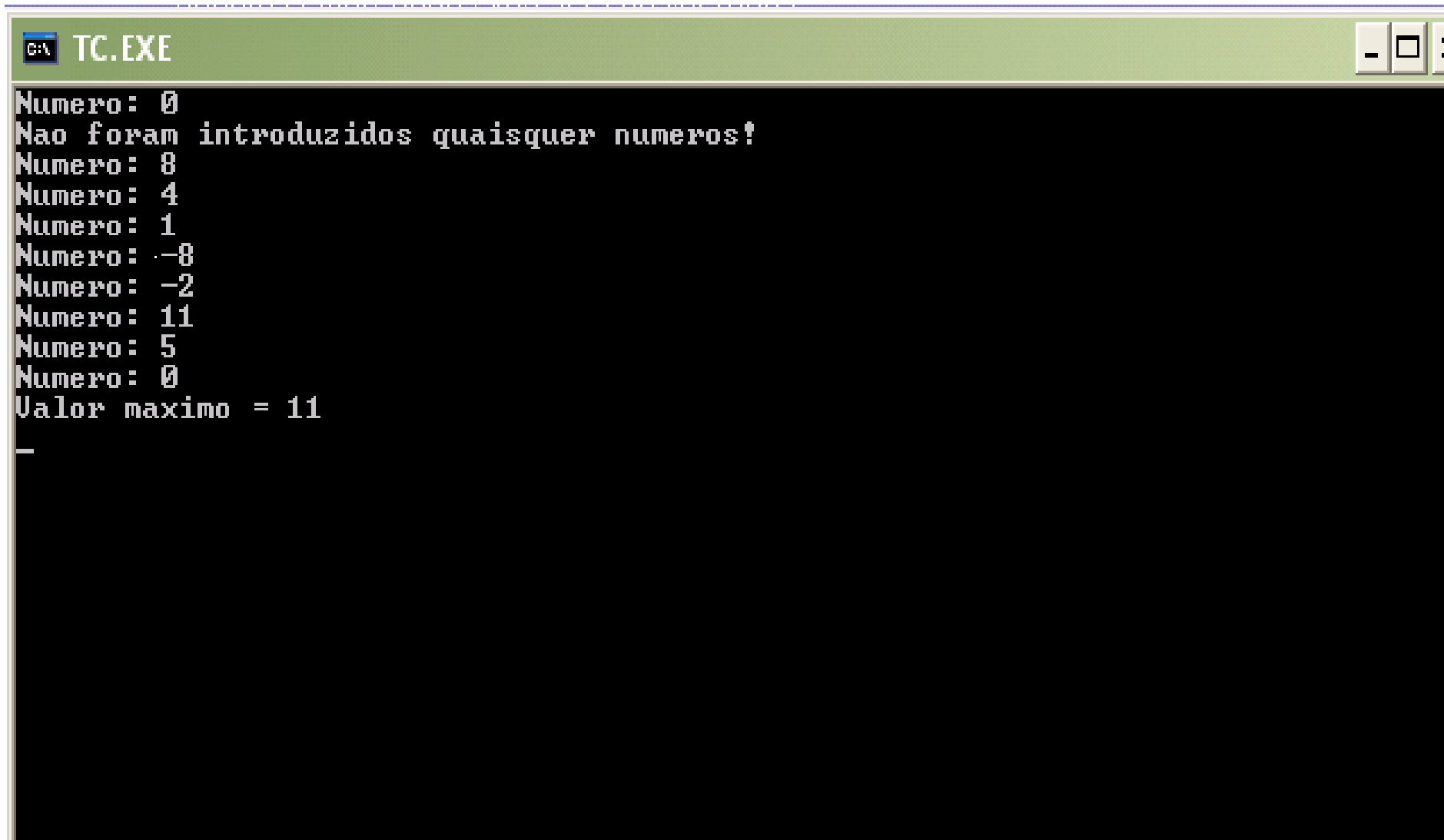
void main()
{
    int numeros[N],tam;

    if((tam=LeNumeros(numeros,N))>0)
        printf("Valor maximo = %d\n",CalculaMaximo(numeros,tam));

    else
        printf("Nao foram introduzidos quaisquer numeros!\n");
}

```

Exemplo de execução:



```
TC.EXE
Numero: 0
Nao foram introduzidos quaisquer numeros!
Numero: 8
Numero: 4
Numero: 1
Numero: -8
Numero: -2
Numero: 11
Numero: 5
Numero: 0
Valor maximo = 11
_
```

```

#include <stdio.h>
#define N 10

int CalculaMaxMin(int n[],int tam, int * max, int * min)
{
    int i;

    *max=*min=n[0];

    for(i=0;i<tam;i++)
    {
        if(n[i]>*max)
            *max=n[i];

        if(n[i]<*min)
            *min=n[i];
    }

    if (*min==*max)
        return 1;
    else
        return 0;
}

void main()
{
    int numeros[N];
    int i, maximo, minimo, iguais;

    for(i=0;i<N;i++)
    {
        printf("Diga o numero %d: ", i+1);
        scanf("%d", &numeros[i]);
    }
    iguais = CalculaMaxMin(numeros, N, &maximo, &minimo);

    if (iguais)
        printf("vetor todo com valores %d.\n", maximo);
    else
    {
        printf("Maximo valor do vetor = %d.\n", maximo);
        printf("Minimo valor do vetor = %d.\n", minimo);
    }
}

```

```

#include <stdio.h>
void main(void)
{
    int c, i, nBranços, nOutros;
    int nDigitos[10];

    nBranços = nOutros=0;
    for (i=0; i<10; i++)
        nDigitos[i]=0;

    while ( (c=getchar()) != '%')
        switch (c) {
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                ++nDigitos[c-'0'];
                break;
            case ' ':
                ++nBranços;
                break;
            default:
                ++nOutros;
                break;
        };

    /* OU
        if (c>='0' && c<='9')
            ++nDigitos[c-'0'];
    */
}

```

```

        else
            if ( c==' ' )
                ++nBrancos;
            else
                ++nOutros;
    */

    printf("Digitos:\n");
    for (i=0; i<10; i++)
        printf("%d -> %d\n", i, nDigitos[i]);

    printf("Brancos -> %d\n", nBrancos);
    printf("Outros  -> %d\n", nOutros);

}

```

```
C:\aa\bin\Debug\aa.exe
ola hoje 12345
dafa
ajs
111
%
Digitos:
in 0 -> 0
bj 1 -> 4
a. 2 -> 1
ra 3 -> 1
ID 4 -> 1
JS 5 -> 1
tp 6 -> 0
gs 7 -> 0
ta 8 -> 0
fl 9 -> 0
'G Brancos -> 2
Jr Outros -> 18
er
no
no
Process returned 14 (0xE)   execution time : 37.047 s
Press any key to continue.
```