12 - MAIS SOBRE APONTADOR

12.1 - Ponteiros e arrays

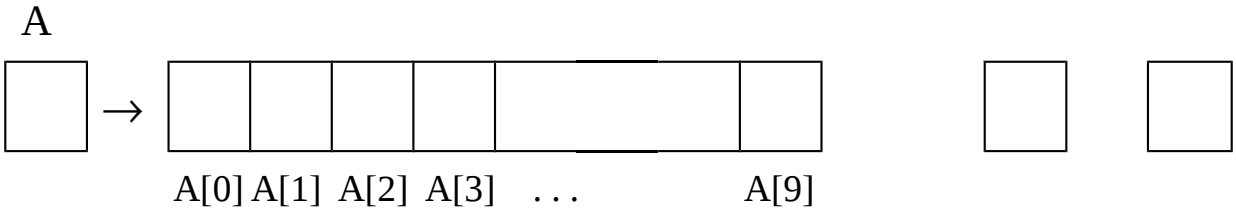
Em C, ponteiros e matrizes são tratados de maneira semelhante.

12.1.1 - Nome

- O nome é um ponteiro, ou seja, aponta (contém o endereço) para o primeiro elemento do array.
- Exemplo:

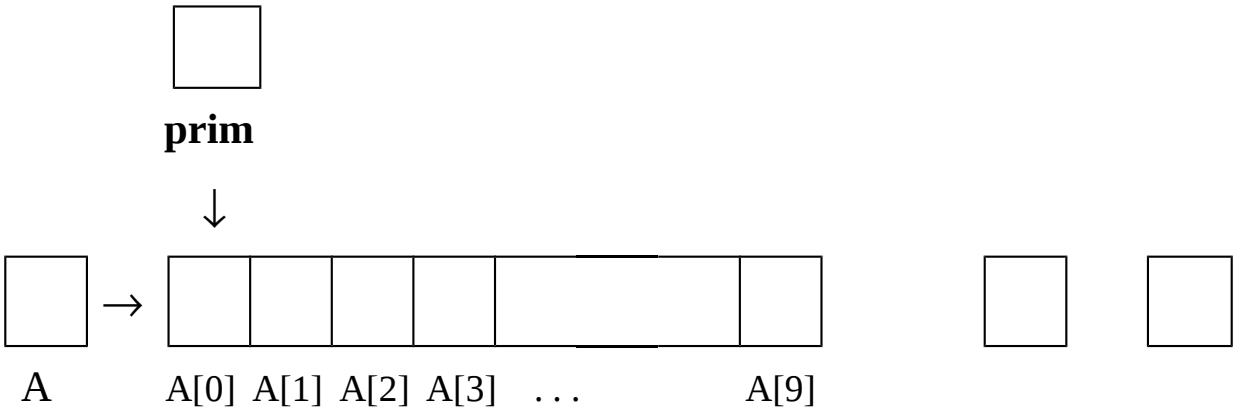
Seja a seguinte declaração:

```
int  A[10],   *prim,   x;
```



- A é um ponteiro para um inteiro: o elemento A[0], ou ainda, A corresponde a &A[0].
- Os seguintes comandos são equivalentes:

```
prim = A ;          e
prim = &A[0];
```



- Os seguintes comandos são equivalentes:

  x = A[0]; e

  x = *A; e

  x = *prim;

Nos comandos acima, o valor do primeiro elemento de A está sendo atribuído a variável prim.

- O nome de uma array não é uma variável**,** logo não é possível alterar seu valor.
- Nome de array é chamado de **ponteiro constante.**
- Exemplo:

Os seguintes comandos não são permitidos pois A é nome de array.

A = pa; e A + +; A = B;

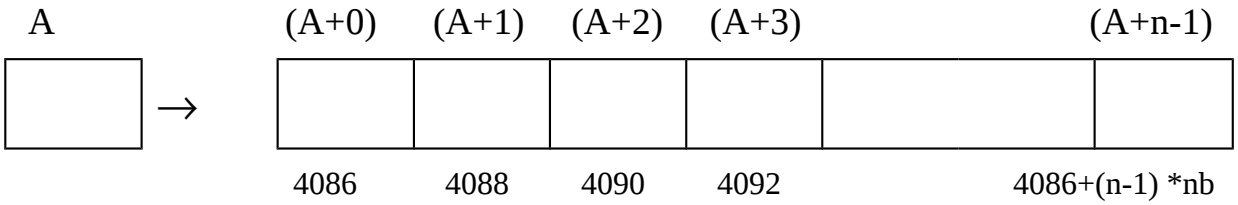
qualquer comando de atribuição a A não é possível

Ex: char nome[20], *pstr;
int val[10], *ptr;

```
pstr = nome;       /* pstr = &nome[0] */
ptr = val;         /*  ptr = &val[0] */

pstr = nome + 4;   /*  pstr = &nome[4] */
ptr = val + 5;     /*   ptr = &val[5] */

pstr = nome++;     /* ATENCAO: INCORRETO !!! */
                   /* "nome" ponteiro constante */
```

12.1.2 - Acesso a elementos do array

- Um vetor com n elemntos, também pode ser entendido da seguinte maneira:



onde nb= no. *bytes* do tipo base de A

- Os elementos de A ocupam posições contíguas na memória.

A[0]	equivale a	*(A+0) ou *A
A[1]	″	*(A + 1)
A[2]	″	*(A + 2)
.		
.		
A[i]	″	*(A + i)
.		
.		
A[n - 1]	″	*(A + n - 1)

Se um ponteiro aponta para um array, pode-se usar indistintamente as formas abaixo para acessar os elementos do array:

```
int val[10],  x,  *ptr;

ptr = val;          /*  ptr = &val[0] */

*(ptr + 3) = 7;   /* val[3] = 7 */
ptr[3] = 10;       /* val[3] = 10    ou   *(ptr + 3) = 10 */
ptr += 4;
ptr[3] = 20;      /* val[7] = 20 */
```

- Ex 1: Cálculo da média aritmética dos elementos de um vetor:

```
main( )
{
   int numeros[ ]= {92, 81, 70, 69, 58};
   int d, soma = 0;

   for (d = 0;  d < 5;  d + +)
       soma + = numeros[d];
   printf (“A média é  % f \n”,  soma/5.0);
}
```

O comando **soma + = numeros[d];** é equivalente à

soma + = *(numeros + d);

- Ex 2: Programa que lê valores e armazena em um vetor.

```
main( )
{
      int numeros[5], k;
      printf (“Digite 5 numeros \n”);
      for (k = 0;  k < 5;  k + +)
          scanf (“% d”,  &A[k]);
}
```

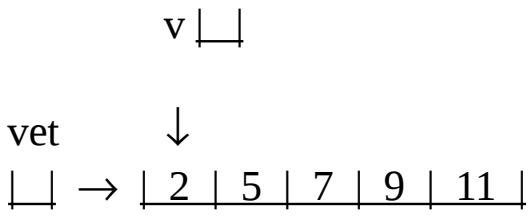
O comando **scanf (“% d”, &A[k]);** é equivalente à

scanf (“% d”, A + k);

- Ex 3: Prog que adiciona uma constante aos elementos do vetor.

```
#define   TAM   5

main( )
{
   float    vet [TAM] = {2, 5, 7, 9, 11},    *v;
   int cte = 10, k;
   v =   vet;        /*  v aponta para primeiro elemento de vet  */

   for (k = 0;  k < TAM;  k + +)
        *(v + k) + = cte;    /* adiciona constante  */

   for (j = 0;   j < TAM  ;  j + +)            /* escreve na tela *//
         printf (“% d”,   *(vet + j));
}
```

Após o comando v = vet; tem-se a seguinte situação:



O comando ***(v + k) + = cte;** é equivalente à

```
vet[k] = vet[k] + cte;

vet[k] += cte;

*(vet + k) = *(vet + k) + cte;
```

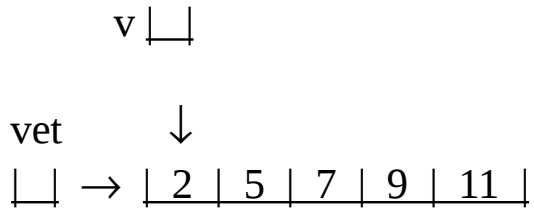
```
for (k = 0;  k < TAM;  k + +)
        *(v + k) + = cte;                é equivalente  à
for (k=0; k < TAM; k ++)
        *v = *(v+ +) + cte;   */
```

- k controla quantas vezes é repetido
- v é incrementado, logo a cada iteração passa a apontar para o próximo elemento do vetor.
- v é um ponteiro variável, logo o comando v++ é possível, mas não esqueça que v mudou seu valor ao final do laço.

12.1.3 - Como argumento de funções

Ex 3: Função addconst que adiciona uma constante aos elementos do vetor:

```
#define   TAM   5

main( )
{   int vet [TAM] = {2, 5, 7, 9, 11};
    int cte = 10;
    addconst (vet, cte)   /*   vet: já é endereço   */
                          /*  cte: passada por valor, não alterada   */
    for (j = 0;   j < TAM  ;  j + +)
        printf (“% d”,   *(vet + j));
}

void addconst (int *v, int c)
      /*            ↓
              endereço de um inteiro (1o. Elemento)   */
{   int k;
    for (k = 0;  k < TAM;  k + +)
        *(v + k) + = c;
}
```



Obs.1: No programa anterior, em main( ) tem-se:

int vet [TAM];

Na definição de addconst:

addconst (int *v, int c)

{ *(v + k) + = c; /* ou *v = *(v+ +) + c; */

/* k controla quantas vezes é repetido */

ou v[k] + = c;

v é um ponteiro variável, por isso é possível o comando v+ +.

Outra maneira de declarar v:

addconst (int v[TAM], int c).

Passagem de parâmetros só para variáveis simples. Neste caso v é um ponteiro constante, não é possível executar o comento v+ +. Não é passada uma cópia do vetor.

Outra maneira:

addconst (int v[ ], int c)

v pode ser declarado como um vetor sem dimensão (porque é argumento).

12.1.4 Parte de matrizes podem ser tratadas como vetores.

Ex1) char nome[NNOMES][NLETRAS];

Para referenciar o endereço de cada nome (string) podemos fazer:

nome[0] ou &nome[0][0] (endereço da 1a. string)

nome[1] ou &nome[1][0] (endereço da 2a. string)

Podemos também referenciar endereços e elementos da matriz através de ponteiros.

Seja a seguinte matriz:

```
#define  LIN  4
#define  COL  5
int tabela [LIN][COL] = {{13, 15, 17, 19, 21},
                         {20, 22, 24, 26, 28},
                         {31, 33, 35, 37, 39},
                        {40, 42, 44, 46, 48}}
```

Supor que a matriz começa no endereço 1000: (supor cada int com 2 *bytes*).

tabela

•→	1000	13	15	17	19	21
	1010	20	22	24	26	28
	1020	31	33	35	37	39
	1030	40	42	44	46	48

Cada linha ocupa 5 * 2*bytes* = 10 *bytes*.

Temos então:

tabela = = tabela + 0 = = tabela[0] = = 1000

tabela + 1 = = tabela[1] = = 1010

tabela + 2 = = tabela[2] = = 1020

Temos também que:

*(tabela) = = tabela = = tabela[0] = &tabela[0][0] = 1000

*(tabela + 2) = = tabela + 2 = = tabela[2] = &tabela[2][0] = 1020

Isto é válido para matrizes!!! Vetores Bidimensionais.

Obs: Se “tabela” fosse um vetor UNIDIMENSIONAL, então *(tabela + k) seria o conteúdo da posição k.

Como “tabela” é bidimensional, então *(tabela + k) tem o mesmo efeito que tabela+ k.

Onde está a diferença entre estas duas notações?

Supor a operação:

tabela + 2 + 1 gera tabela + 3 (&tabela[3][0] ou tabela[3])

1020 1030

ou seja, somou 1 linha (5 x 2 *bytes* = 10 *bytes*)

Outra operação:

*(tabela + 2) + 1 gera &tabela[2][1]

1020 1022

ou seja, somou 1 coluna (2 *bytes*), ou ainda, foi para o próximo elemento.

Então:

Se *(tabela + 2) + 1 é o endereço de tabela[2][1]

então *(*(tabela + 2) + 1) igual a tabela[2][1]

tabela[i][j] → *(*(tabela + i) + j)

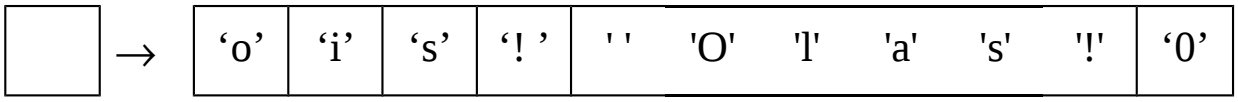
12.2 - Inicialização de Strings

Já vimos a seguinte inicialização de string:

char salute[ ] = “ois! Olas!”; (1)

É declarado um vetor sem dimensão, mas inicializado com 11 posições: 10 para os caracteres acima e 1 para \0.

salute



Esta declaração também pode ser feita da seguinte maneira:

char *salute = “ois! Olas!”; (2)

Esta declaração provoca o mesmo efeito da anterior, porém neste caso, salute é ponteiro variável, ou seja, podemos modificar seu valor.

É possível: salute = &car;

salute + +; ... (apontaria para o segundo caracter)

Em (1) salute é ponteiro constante.

12.3 - Vetor de strings x Vetor de ponteiros para strings

Um vetor de strings é um vetor de vetores ou uma matriz de duas dimensões.

Ex1: Programa que verifica se um nome a ser digitado está no vetor de nomes(list).

```
#define  NNOMES   5
#define  NLETRAS  10

main( )

{   int k, encontrou = 0;
   char  nome[NLETRAS],  list  [NNOMES][NLETRAS]=  {“Karin”, “Ana”, “Gustavo”, “Beth”, “Barney”};
   printf (“Digite nome: “);
   gets(nome);
   for (k = 0; k < NNOMES; k + +)
       if (strcmp (nome, list[k] = = 0)
           encontrou = 1;
   if (encontrou = = 1)
        printf (“\n O nome está na lista!\n”);
   else
        printf (“\n Não está!\n”);
}
```

1o. Elemento list[0][0] → conteúdo = 'K'

&list[k][0] == endereço do 1o. nome == list[k] == list + k (cuidado!)

list (=ponteiro para o 1o. elemento)

→	1000	‘K’	‘a’	‘r’	‘i’	‘n’	\0				
	1010	‘A’	‘n’	‘a'	\0						
	1020	‘G’	‘u’	‘s’	‘t’	‘a’	‘v’	‘o’	\0		
	1030	‘B’	‘e’	‘t’	‘h’	\0					
	1040	‘B’	‘a’	‘n’	‘n’	‘e’	‘y’	\0			

list[2][0] = G

&list[2][0] = 1020

list[2] = list + 2 = 1000 + 2*10 = 1020 (O compilador “faz a conta”)

list[0] acessa o endereço 1000

list[1] ″ ″ 1010

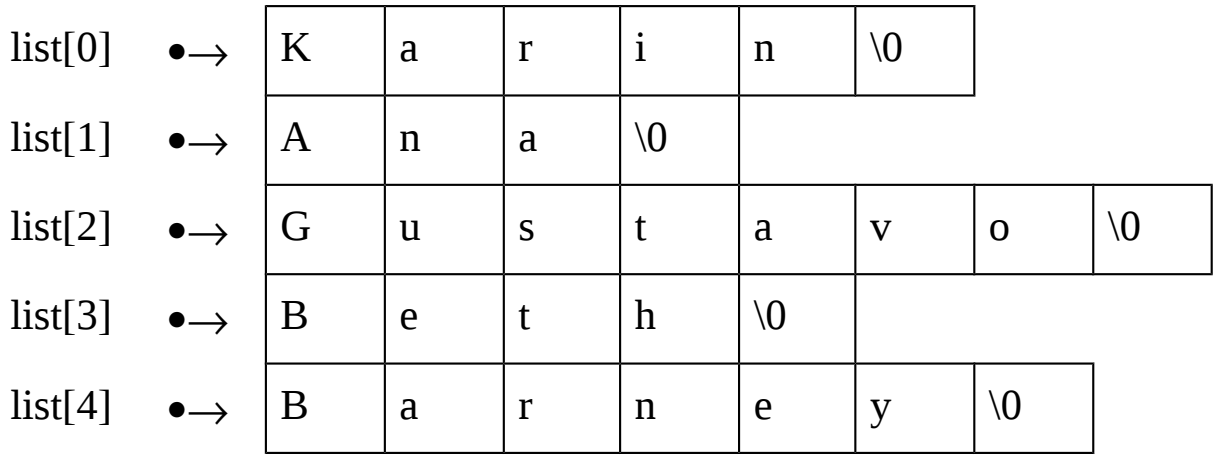
list[2] ″ ″ 1020

list[3] ″ ″ 1030

list[k] é um apontador para o nome k.

Versão 2: Usando um vetor de ponteiros para strings

char *list[NNOMES] (vetor de ponteiros)




```
#define  NNOMES  5

main( )
{   int k, encontrou = 0;
    char nome[20];
    char  *list[NNOMES] = {“Karin”, “Ana”, ...};
    /*foi declarado e inicializado ao mesmo tempo, e economizou
espaço*/
     printf ...
     gets (nome);
     for (k = 0; k < NNOMES; k + +)
         if (strcmp(nome, list[k]) = = 0)
            encontrou = 1;
     if ...
 }
```

Comentários: Na Versão 2 alocamos espaço para o número de caracteres necessários.

Este tipo de inicialização é mais eficiente, pois ocupa menos espaço de memória. Cada linha da matriz tem tamanho diferente.

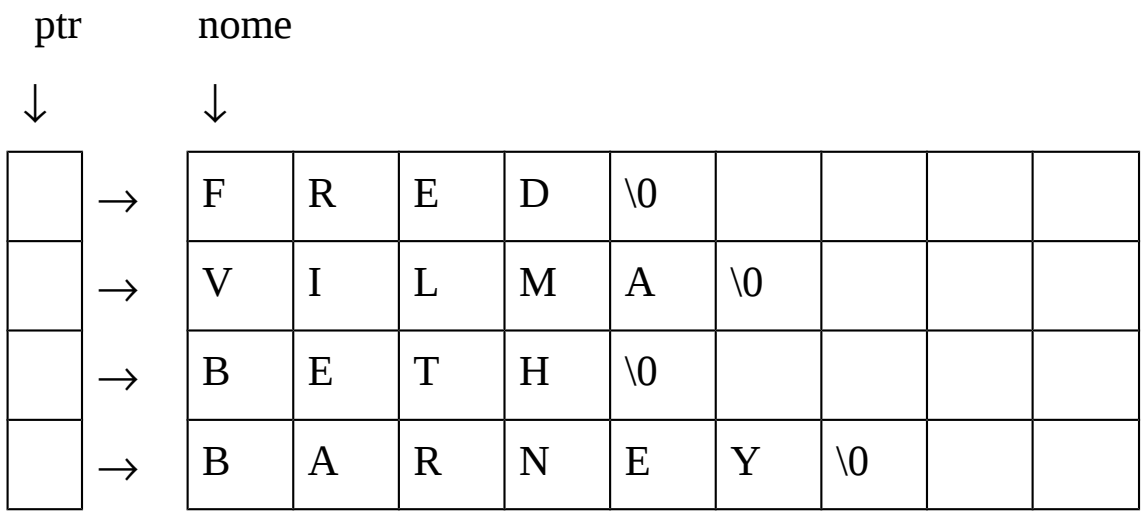
Exemplo 2: Ler um vetor de nomes e ordená-lo usando “seleção”.

Idéia: Ler a matriz de strings e declarar um vetor de ponteiros para cada nome. (apontando para o começo de cada nome). Em vez de mudar os nomes, depois da ordenação, mudar os ponteiros.

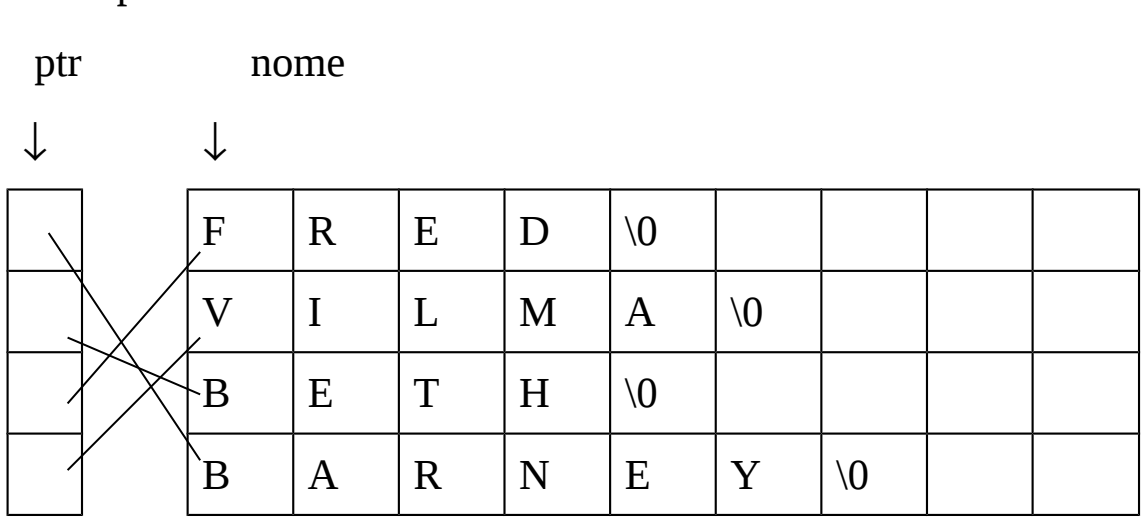
```
#include <stdio.h>
#include <string.h>

#define  NNOMES  4
#define  NLETRAS  10

void leitura (char n[] [NLETRAS], char **pt);
void ordena (char **pt);
void imprime (char **pt);   /* ou (char *pt[ ]) */

void leitura (char n[][NLETRAS], char **pt)       /* ou char *pt[ ] ) */
{   int k;
    for (k = 0; k < NNOMES; k++)
    {  printf ("Digite nome: ");
       fgets (n[k],10,stdin);            //   guarda os nomes (= &n[k][0] )
       pt[k] = n[k];          //  possível usar *(pt+k)  ou *(pt++)
     }
 }                /*  Obs: char **pt = ponteiro variável
                      char *pt[ ] = ponteiro constante   */

void ordena (char **p)
{  int i, j, min;
   char *aux;
   for (i = 0; i < NNOMES; i++)
   {
      min = i;
      for (j = i + 1; j < NNOMES; j++)
      {  if (strcmp (p[min], p[j]) > 0)
           min = j;
      }
      aux = p[min];
      p[min] = p[i];
      p[i] = aux;
   }
}
```

```
void imprime (char **p)
{   int i;
    for (i = 0; i < NNOMES; i++)
       printf ("%s", p[i]);
}

int main( )
{  char *ptr[NNOMES], nomes[NNOMES][NLETRAS];
   leitura (nomes, ptr);
   ordena (ptr);
   imprime (ptr);
}
```

Antes:



Depois:



Ex: Outra versão da função ordena, usando seleção.

```
void  ordena2 (char **pt)
{
   void troca (char **a, char **b);     /*   função usada por ordena  */
   int i, j;

   for (i = 0; i < NNOMES;  i++)
       for (j = i + 1;  j < NNOMES;  j++)
           if (strcmp(pt[j], pt[i]) < 0)
              troca (&pt[j], &pt[i]);  /* pq não usar pt[j] ??
                                          quero alterar os endereços
                                          &pt[j]  é equivalente a  pt + j  */
}

void troca (char **a, char **b)
{   char *temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
```

Observações:

&pt[j] é equivalente a pt + j;

Qual a diferença entre passar &pt[j] e passar simplesmente pt[j] como argumento de troca???

Por que não foi usado min = i ???

12.4 - Argumentos de Linha de Comandos

Permitem chamar o programa por seu nome, através do sistema operacional na linha de comando e passar argumentos para o mesmo.

Ex: c > ws relatorio.txt

argumento (dado de entrada)

ws: nome do programa executável

Estes argumentos são colocados na função main:

Ex 1:

```
main(int argc, char *argv[ ]))
/* int argc;     (sempre inteiro)
char *argv[ ];     (vetor de pont. para char (strings), )
 argc : no. de argumentos incluindo o nome do programa
 argv : vetor de pont. para char  tem dimensão argc */

{   int j;
    printf ("Número de argumentos é % d \n", argc);
    for (j = 0; j < argc; j++)
        printf (" arg %d é %s\n", j, argv[j]);
}
```

Execução:

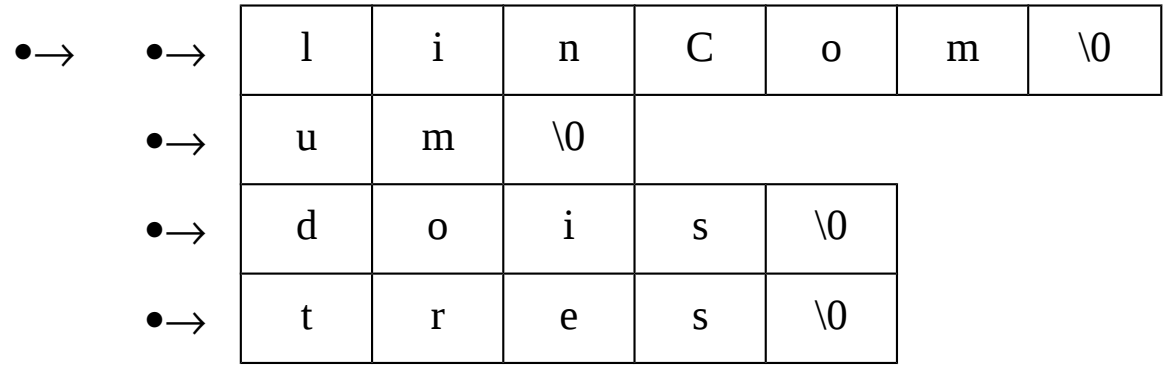
c > linCom um dois tres (executável é linCom)

número de argumentos é 4.
arg 0 é linCom
arg 1 é um
arg 2 é dois
arg 3 é três
arg v[0] → "lincom"
arg v[1] → "um"
arg v[2] → "dois"
arg v[3] → "tres"

A primeira string argv[0] ou argv + 0 ou *(argv + 0) contém o nome do programa (endereço); as outras strings servem para outras finalidades.

Se argc é 1, não existem dados de entrada (só tem o nome do programa através da linha de comando), para o programa anterior:

argv



Ex 2: E se os argumentos forem números?

```
main (int argc, char *argv[ ])
{   int num;
    char nome [20];
    strcpy (nome, argv[1]);
    num = atoi (argv[2]);
     ...
       /*   atoi : converte string para inteiro   */
       /*   atof : converte string para float   */
```

Deve-se digitar:

prog nome idade

Ex:

```
prog        Ana         20
 ↓          ↓           ↓
argv[0]   argv[1]    argv[2]
```

Ex 3: Programa para concatenar vários arquivos. O primeiro é o destino.

```
#include <stdio.h>
void copia (FILE *, FILE *);

int main (int argc,  char *argv[ ])
{  FILE   *destino,   *arquivo;
   int k;
   if (argc == 1)                          /*   nome do programa   */
      printf ("Você não digitou os nomes dos arquivos \n");
   else
       if (argc == 2)          /*   nome do programa e destino  */
           printf("Só digitou arquivo destino \n");
       else
       {
           destino = fopen (argv[1], "a");
          for (k = 2; k < argc; k++)     /*   0 : nome do programa
                                             1 : destino
                                             2 : origem   */
          {   arquivo = fopen (argv[k], "r");
              copia (destino,  arquivo);
              fclose (arquivo);
          }
         fclose (destino);
       }
 }

void copia (FILE  *dest, FILE *arq)
{   int c;
    while ((c = getc (arq)) != EOF)
        putc (c, dest);
}
```

OBS: os separadores dos argumentos são espaços brancos, tabs, ou mudança de linha. Dica: aspas e apóstrofes podem agrupar palavras como um argumento único, mesmo que tenham espaçamento.

12.5 – Valores de retorno da função main

A função main pode retornar um valor a ser utilziado pelo sisema operacional. No terminal UNIX, usando-se o shell bash, o valor retornado é dado pela variável $?

```
int main ()
{
  return (1) ;
}
```

O comando echo usado para mostrar o valor de variáveis de ambiente

```
> gcc -oexe programa.c
> ./exe
> echo $?
> 1
```