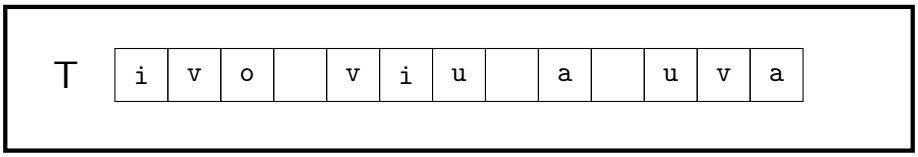
Fundamentos de programação

aula 04: Programando com palavras

1 Introdução

Na linguagem C, nós armazenamos texto na memória da mesma maneira que uma lista de números



Quer dizer, as letras ou caracteres ficam armazenados em posições diferentes, uma ao lado da outra.

Mas, ao contrário das listas de números, basta um comando `printf` para imprimir a coisa toda

Mas, como é que isso funciona?

Bom, a primeira observação é que dessa vez nós usamos o símbolo %`s` no comando `printf`.

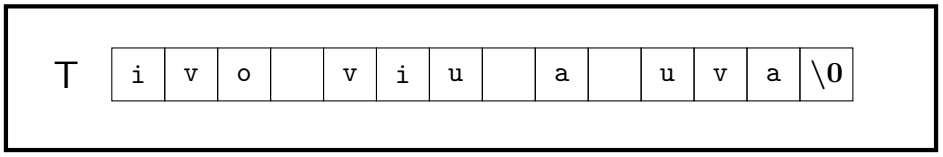
Isso é a indicação de que a variável `T` contém texto.

(A letra `s` vem da palavra *string*, que é o termo em inglês para sequência de caracteres.)

E isso significa que o programa vai imprimir um por um, todos os caracteres armazenados em `T`.

Mas, como é que ele sabe quando deve parar?

Bom, logo após a última letra ou caracter do texto, existe um símbolo especial
`0` que indica que ali o texto chegou ao fim



O comando `printf`, claro, não imprime esse símbolo na tela.

Pronto.

É basicamente isso.

Agora nós já podemos escrever o nosso primeiro programa em C que manipula texto.

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

```
char T[50] = "ivo viu a uva";

int main ()
 {
    printf("%s", T);
 }
```

Algumas observações:

- Nós já mencionamos que o texto é armazenado na memória como uma lista.

  Como o texto é uma lista de caracteres, nós temos a declaração

  ```
  char T[50]
  ```

  E, como usual, nós já colocamos alguma coisa lá no momento da declaração

  ```
  char T[50] = "ivo viu a uva";
  ```

  Essa instrução também já coloca o
  `0` logo após a última letra.

  Mas, para fazer isso é preciso ter espaço: o comprimento da frase `"ivo viu a uva"` mais 1 deve ser menor ou igual a 50 (o que é o caso).

2 Programando com palavras

Mas, se o texto é uma lista de caracteres, então nós podemos manipular os caracteres individualmente.

Sim, isso é verdade.

Por exemplo, nós podemos trocar `"ivo"` por `"bia"` na frase, da seguinte maneira

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

char T[50] = "ivo viu a uva";

int main ()
 {
    printf("%s", T);
    T[0] = 'b';
    T[1] = 'i';
    T[2] = 'a';
    printf("%s", T);
 }
```

Quando nós executamos esse programa, nós gostaríamos de ver

```
ivo viu a uva
bia viu a uva
```

Mas o que aparece na tela é

```
ivo viu a uvabia viu a uva
```

Quer dizer, não aconteceu uma quebra de linha após o primeiro `printf`.

Se nós quisermos que isso aconteça, é preciso indicar explicitamente da seguinte maneira

```
printf("%s", T);
```

(Isto é, `\n` é o símbolo especial que indica a quebra de linha.)

Outra coisa interessante é que nós também podemos colocar o símbolo `\0` em qualquer lugar.

Por exemplo

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

char T[50] = "ivo viu a uva";

int main ()
 {
    printf("%s", T);
    T[3] = '\0';
    printf("%s", T);
 }
```

E nós também podemos fazer um programa que funciona como o `printf`

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

char T[50] = "ivo viu a uva";
int  i;

int main ()
 {
    for (i=0;i<=50;i++)
     {
        if (T[i] == '\0')  break;

        printf("%c", T[i]);
     }
 }
```

Observação:

- Note que para imprimir um caracter individualmente, nós utilizamos o símbolo especial `%c`

  ```
  printf("%c", T[i]);
  ```

  Dica: Substitua o `%c` por `%s` e veja o que acontece.

3 Outros exemplos

Imagine que nós queremos imprimir o texto que está armazenado em `T` de trás para frente.

Para fazer isso, o primeiro passo é saber o tamanho do texto.

E para isso nós utilizamos o comando

`strlen`(`T`)

(onde `str` vem da palavra *string*, e `len` vem da palavra *length* que significa comprimento em inglês)

Certo.

Uma vez que nós sabemos o comprimento do texto, basta imprimir os seus caracteres um a um, de trás para frente

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

char T[50] = "ivo viu a uva";
int  i, compr;

int main ()
 {
    compr = strlen(T);

    for (i=compr-1;i>=0;i--)
        printf("%c", T[i]);
 }
```

Observações:

- O comando `strlen` faz parte da biblioteca de funções para a manipulação de strings da linguagem C.

  Por isso, nós adicionamos a declaração

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

  no início do programa.

- Esse é o primeiro programa que nós fazemos, onde o comando `for` anda para trás.

Mais exemplos

a. **Palíndromos**

Existem algumas palavras e frases que são a mesma coisa quando lidas de trás para frente e de frente para trás.

Por exemplo,

```
Orava o avaro
```

Essas frases são conhecidas como *palíndromos*.

E não é difícil fazer um programa que verifica se uma frase é um palíndromo ou não.

Quer dizer, basta

- comparar o primeiro caracter com o último
- comparar o segundo caracter com o penúltimo
- e assim por diante ...

Ora, mas esse é o tipo de coisa que nós podemos fazer com o comando `for`.

Vejamos.

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

char T[50] = "eva asse essa ave";
int  i, j, compr, Aux = 0;

int main ()
 {
    compr = strlen(T);

    for (i=0;i<compr;i++)
     {
        j = compr - i - 1;

        if ( T[i] != T[j] )
         {
           Aux = 1;   break;
     }   }

     if ( Aux == 0 ) printf("T é um palíndromo");
     else            printf("T não é um palíndromo");
 }
```

Observações:

- Aqui nós utilizamos o símbolo `!=` que corresponde ao sinal de diferente ($\neq$) em C
  - `if ( T[i] != T[j] )`
- Você notou que nós comparamos cada par de elementos duas vezes?

  Por exemplo, o primeiro e o último caracteres foram comparados
  - uma vez quando `i = primeiro` e `j = último`
  - outra vez quando `i = último` e `j = primeiro`

  Isso pode ser evitado colocando a seguinte instrução dentro do comando `for`
  - `if ( j < i ) break;`

  Mas essa solução tem um pequeno problema: nós evitamos a comparação duplicada entre `T[i]` e `T[j]` introduzindo uma comparação entre `i` e `j`.

  Isso significa que a solução custa tão caro quanto o problema.

  Uma solução melhor consiste em calcular o ponto médio

$$\texttt{meio} \quad = \quad \frac{\texttt{compr}}{2}$$

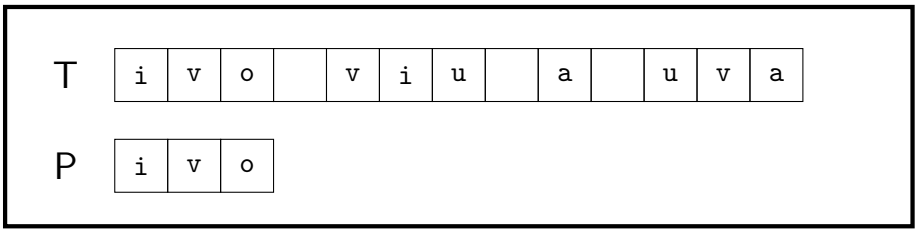
  e modificar o comando `for` para
  - `for ( i=0 ; i<=meio ; i++ )`

◇

b. CTRL-F

O comando CTRL-F dos editores de texto permite que você localize a ocorrência de uma palavra chave dentro do seu texto.

Para programar essa funcionalidade, nós podemos imaginar que o texto e a palavra chave são duas variáveis `T` e `P` na mémória



Daí, basta comparar os conteúdos de `T` e `P`

```
. . .
Aux = 0;

compr-p = strlen(P);

for (i=0;i<compr-p;i++)
 {
    if ( T[i] != P[i] )
     {
       Aux = 1;  break;
 }   }

if ( Aux == 0 )  printf("P aparece no início de T");
else             printf("P não aparece no início de T");
```

Certo.

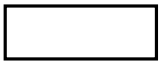
Esse programa só localiza a palavra `P` no início de `T` (i.e., começando na posição 0).

Mas é bem fácil modificar o programa para que ele procure a palavra `P` na posição `k` de `T`

```
     . . .
 =>  k = 10;

     Aux = 0;
     compr-p = strlen(P);

     for (i=0;i<compr-p;i++)
      {
 =>      if ( T[i+k] != P[i] )
          {
            Aux = 1;  break;
      }   }

     if ( Aux == 0 )  printf("P aparece no início de T");
     else             printf("P não aparece no início de T");
```

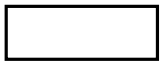
Legal.

Agora, pensando nesse programa como uma caixinha

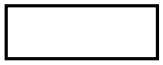


nós podemos utilizá-lo para construir um outro programa que procura a palavra `P` em todas as posições de `T`

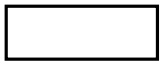
```
k = 0;
```



```
k = 1;
```



```
k = 2;
```



```
 ...
```

Mas, na prática, isso corresponde a colocar o programa dentro de um comando `for`

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

char T[50] = "ivo viu a uva";
char P[50] = "viu";

int  comprP, comprT, i,k, Aux;

int main()
 {
    comprP = strlen(P);   comprT = strlen(T);

    for (k=0;k<=comprT-comprP; k++)
     {
        Aux = 0;
        for (i=0;i<comprP;i++)
         {
            if ( T[i+k] != P[i] )
             {
               Aux = 1;  break;
         }   }

        if ( Aux == 0 )  break;
     }

    if ( Aux == 0 )  printf("P aparece na posição %d de T",k);
    else             printf("P não aparece em T");
 }
```

Observação:

- Note que a última posição de `T` onde pode começar uma ocorrência de `P` é

  $$\texttt{comprT} - \texttt{comprP}$$

  Por exemplo, se `T` tem comprimento 10 e `P` tem comprimento 3, então isso corresponderia à oitava posição.

  Mas, começando a contar do 0, isso corresponde a $7 = 10 - 3$.

  Daí que, o comando `for` mais externo foi escrito assim

  ```
  for ( k=0 ; k<=comprT-comprP ; k++)
  ```

◇

c. **Contando ocorrências**

Imagine agora que nós não queremos apenas localizar uma ocorrência da palavra chave `P` em `T`, mas contar todas as ocorrências de `P` em `T`.

Na verdade, isso é bem fácil (*depois que nós já temos o programa anterior*).

Quer dizer, basta declarar o contador

```
int Cont - 0;
```

e substituir a última instrução do comando `for` externo por

```
if ( Aux == 0 )   Cont ++;
```

Dessa maneira, o comando `for` mais externo não termina quando a primeira ocorrência de `P` é encontrada, mas continua encontrando (e contando) as demais ocorrências.

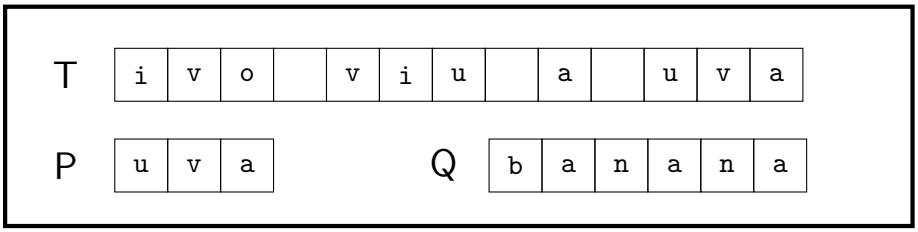
Abaixo nós temos o programa que implementa essa ideia

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

char T[50] = "ivo viu a uva";
char P[50] = "viu";

int  comprP, comprT, i,k, Aux, Cont=0;

int main()
 {
    comprP = strlen(P);   comprT = strlen(T);

    for (k=0;k<=comprT-comprP; k++)
     {
        Aux = 0;
        for (i=0;i<comprP;i++)
         {
            if ( T[i+k] != P[i] )
             {
               Aux = 1;  break;
         }   }
        if ( Aux == 0 )  Cont++;
     }
    printf("Existem %d ocorrências de P em T", Cont);
 }
```

◇

d. CTRL-H

Os editores de texto também possuem o comando CTRL-H, que permitem encontrar uma ocorrência da palavra chave `P` e substituí-la por uma outra palavra `Q`.



Como indicado no esquema acima, essa operação pode modificar o tamanho do texto armazenado em `T`.

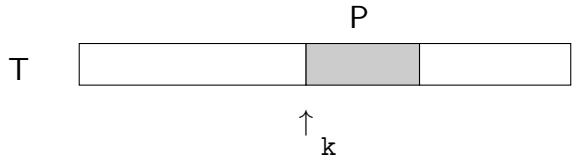
E aqui nós precisamos tomar algum cuidado.

A melhor estratégia para resolver esse problema consiste em pensar por etapas.

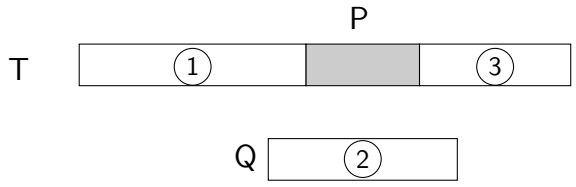
Quer dizer, primeiro nós devemos encontrar uma ocorrência de `P` em `T` (se houver);

Isso pode ser feito com o programa do exemplo (b).

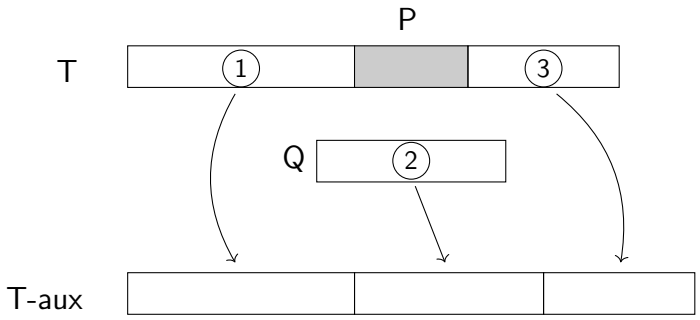
Digamos que essa ocorrência foi encontrada na posição k.



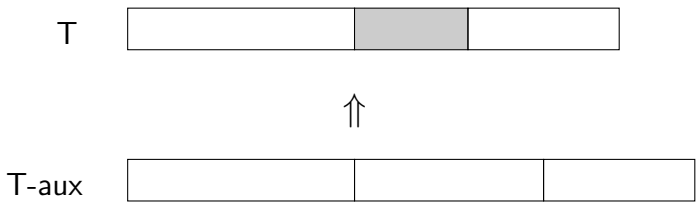
Daí, nós visualizamos as 3 partes da solução



E para juntar essas 3 partes, nós utilizamos uma outra variável `T-aux`



Daí, uma vez que a solução já está pronta em `T-aux`, basta levá-la de volta para `T`



Pronto, é só isso.

O programa abaixo implementa essa ideia

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

```
char T[50] = "ivo viu a uva", T-aux[50];
char P[50] = "viu",  Q[50] = "banana";

int  comprP, comprQ, comprT, comprA;
int  i,j,k,m Aux;

int main()
 {
    //  ETAPA 1: localizar a ocorrência de P em T

    comprP = strlen(P);   comprT = strlen(T);
    for (k=0;k<=comprT-comprP; k++)
     {
        Aux = 0;
        for (i=0;i<comprP;i++)
            if ( T[i+k] != P[i] )   { Aux = 1;  break; }
        if ( Aux == 0 )  break;
     }
    if ( Aux == 1 )  return();

    //  ETAPA 2: montar a solução em T-aux

    comprQ = strlen(Q);
    for (i=0;i<k;i++)  T-aux[i] = T[i];
    for (j=0;j<comprQ;j++)  T-aux[j] = Q[j];
    m = comprQ - comprP;
    for (i=k+comprP;i<comprT;i++)  T-aux[i+m] = T[i];

    //  ETAPA 3: copiar a solução de volta para T

    comprA = strlen(T-aux);
    for (i=0;i<=comprA;i++)  T[i] = T-aux[i];
 }
```

◇