Caratere e cadeia de caracteres
Professor Wladimir A. Tavares

Declarando uma variável caractere

Uma variável do tipo `char` é usada para armazenar um caractere. Essa variável pode ser usada por armazenas letras minúsculas, maiúsculas e alguns caracteres especiais. Essa variável permite também fazer alguns cálculos.

```c
#include <stdio.h>
int main(){
    char c = 'a'; //declarando uma variável char
    printf("%c\n", c); //imprimindo 'a'
    c++;
    printf("%c\n", c); //imprimindo 'b'
    c++;
    printf("%c\n", c); //imprimindo 'c'

}
```

Reconhecendo letras minúsculas e maiúsculas

Como os códigos das letras minúsculas são consecutivos, podemos fazer o seguinte código para identificar uma letra minúscula ou maiúscula.

```c
#include <stdio.h>

int is_lower(char c){
    return c >= 'a' && c <= 'z';
}

int is_upper(char c){
    return c >= 'A' && c <= 'Z';
}

int main(){
    is_lower('a'); // devolve 1
    is_lower('A'); // devolve 0
    is_upper('a'); // devolve 0
    is_upper('A'); // devolve 1
}
```

Mudando para caixa-alta e caixa-baixa

```c
#include <stdio.h>

char to_lower(char c){
    if( is_upper(c) ){
        d = c - 'A';
        return 'a' + d;
    }else{
        return c;
    }
}

int to_upper(char c){
    if( is_lower(c) ){
        d = c - 'a';
        return 'A' + d;
    }
}

int main(){
    to_lower('a'); // devolve 'a'
    to_lower('A'); // devolve 'a'
    to_upper('a'); // devolve 'A'
    to_upper('A'); // devolve 'A'
}
```

Declarando e imprimindo cadeia de caracteres

```c
#include <stdio.h>
int main(){
    char frase[20] = "o rato roeu";
    printf("%s", frase); // o rato roeu
}
```

Na linguagem C, as cadeias de caracteres possuem um tamanho dinâmico limitado, ou seja, a cadeia de caracteres `frase` pode ter entre 0 e 19 caracteres. Na linguagem C, o caractere `'/0'` é usado para representar o fim da cadeia de caracteres. Na Figura 1, temos uma imagem da representação da cadeia de caracteres na memória.

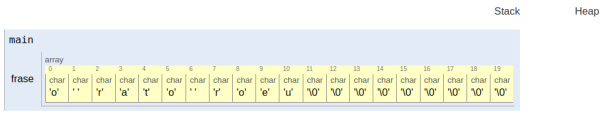


Figura 1: Representação na memória

Note que podemos adicionar mais caracteres colocando o `'/0'`.

```c
#include <stdio.h>
int main(){
    char frase[20] = "o rato roeu";
    printf("%s\n", frase);
    frase[11] = ' ';
    frase[12] = 'a';
    frase[13] = ' ';
    frase[14] = 'r';
    frase[15] = 'o';
    frase[16] = 'u';
    frase[17] = 'p';
    frase[18] = 'a';
    frase[19] = '\0';
    printf("%s\n", frase); //o rato roeu a roupa
}
```

Encontrando o tamanho de uma cadeia de caracteres

Posicionamos um apontador na primeira posição da palavra e vamos avançando enquanto o valor apontado pelo apontador seja diferente `'/0'`.

```c
#include <stdio.h>
int tam(char s[]){
    int i;
    i = 0;
    while( s[i] != '\0') i++;
    return i;
}
int main(){
    char frase[20] = "o rato roeu";
    printf("%d\n", tam(frase));//11
}
```

Checando se todos os caracteres são minúsculos

```c
#include <stdio.h>

int is_upper(char c){
    return c >= 'A' && c <= 'Z';
}

int is_all_lower(char s[]){
    int i;
    i = 0;
    while( s[i] != '\0'){
        if(s[i] == ' ') i++;
        else if( is_upper(s[i]) ) return 0;
        else i++;
    }
    return 1;
}

int main(){
    char frase[20] = "o rato roeu";
    printf("%d\n", tam(frase));//11
    printf("%d\n", is_all_lower(frase));
}
```

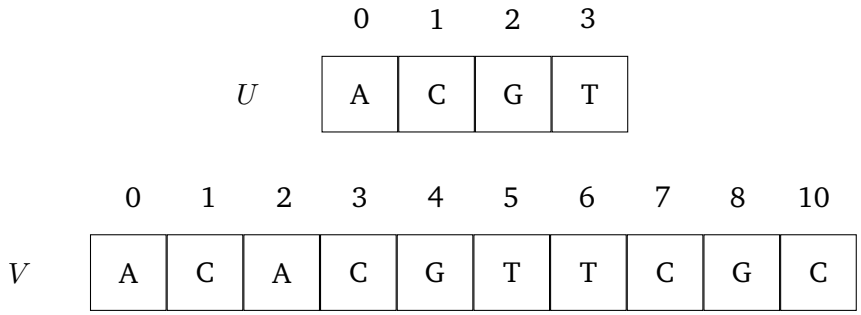
Contando o número de caracteres minúsculos

```c
#include <stdio.h>

int is_lower(char c){
    return c >= 'a' && c <= 'z';
}


int count_lower(char s[]){
    int i, count;
    i = 0;
    count = 0;
    while( s[i] != '\0'){
        if( is_lower(s[i])) count++;
        i++;
    }
    return count;
}

int main(){
    char frase[20] = "o rato roeu";
    printf("%d\n", tam(frase));//11
    printf("%d\n", count_lower(frase));
}
```

Invertendo uma palavra

```c
#include <stdio.h>

int tam(char s[]){
    int i;
    i = 0;
    while( s[i] != '\0') i++;
    return i;
}
void inverte(char s[]){
  int i, j;

  i = 0;
  j = tam(s) - 1;

  while( i < j ){
    char temp = s[i];
    s[i] = s[j];
    s[j] = temp;
    i++;
    j--;
  }

}

int main(){
    char frase[20] = "o rato roeu";
    inverte(frase);
    printf("%s\n", frase); //ueor otar o
}
```

Verifique se a palavra U aparece na palavra V em posições consecutivas



Versão 1

```
int busca(char V[], char U[]){
  int i, j;
  int N, M;
  i = 0;
  j = 0;
  N = tam(V);
  M = tam(U);
  while( i < N){
    if( j == M ){
      //palavra começa na posiçao i-M
      return i-M;
    }else if(V[i] == U[j]){
      //avanca i e j
      i++; j++;
    }else{
      //volta para o começo de U
      j = 0;
    }
  }
  return -1;
}
```

i	j	V[i]	U[j]	Ação
0	0	A	A	Avança i e j
1	1	C	C	Avança i e j
2	2	A	G	Volta j para começo
2	0	A	A	Avança os dois
3	1	C	C	Avança os dois
4	2	G	G	Avança os dois
5	3	T	T	Avança os dois
6	4	T	T	Encontra casamento na posição 6-4

O programa anterior fica em preso no laço com a seguinte entrada:

	0	1	2	3
U	A	A	G	T

	0	1	2	3	4	5	6	7	8	10
V	A	A	A	G	T	G	T	C	G	C

i	j	V[i]	U[j]	Ação
0	0	A	A	Avança i e j
1	1	A	A	Avança i e j
2	2	A	G	Volta j para começo
2	0	A	A	Avança os dois
3	1	G	A	Volta j para o começo
3	0	G	A	Volta j para o começo
3	0	G	A	Volta j para o começo

Versão 2

A abordagem utiliza três apontadores para realizar a busca de uma substring U dentro de uma string V:

- i — aponta para a posição atual em V onde a busca por U começa.
- j — aponta para a posição atual em U que está sendo comparada.
- k — acompanha a posição correspondente em V durante a comparação com U.

A ideia é que, para cada posição inicial i em V, iniciamos uma comparação entre V e U a partir dos ponteiros $k = i$ e $j = 0$. Enquanto os caracteres $V[k]$ e $U[j]$ coincidirem, ambos j e k avançam para comparar os próximos caracteres.

Se todos os caracteres de U forem comparados com sucesso ($j == M$), isso indica que U foi encontrada em V a partir da posição i, e então retornamos i como o índice de início do casamento. Caso contrário, incrementamos i para testar a próxima posição em V.

```
int busca2(char V[], char U[]){
  int i, j, k;
  int N, M;
  N = tam(V);
  M = tam(U);
  i = 0;
  while( i+M-1 < N){
    k = i;
    j = 0;
    while( j < M && V[k] == U[j]){
        j++;
        k++;
    }
    if(j == M){
        return i;
    }else{
        i++;
    }
  }
  return -1;
}
```

Verifique se todos os caracteres de U aparece na palavra V em posições possivelmente não consecutivas

```
int busca3(char V[], char U[]){
  int i, j;
  int N, M;
  N = tam(V);
  M = tam(U);
  i = 0;
  j = 0;
  while( i < N && j < M){
    if(V[i] == U[j]){
        printf("i %d j %d\n", i , j);
        i++;
        j++;
    }else{
        i++;
    }
  }
  if(j == M){
    return 1;
  }else{
    return -1;
  }
}
```

Exercícios

1. Escreva uma função que verifica se uma string é um palíndromo (lê-se da mesma forma de frente para trás). Exemplo: Entrada: ”radar”Saída: É um palíndromo Solução: Use dois ponteiros, um no início e outro no final, e compare os caracteres até o meio da string.

2. Implemente uma função para concatenar duas strings sem usar a função strcat. Exemplo: Entrada: ”Hello, ”e ”World!”Saída: ”Hello, World!”Solução: Percorra a primeira string até o final, depois copie cada caractere da segunda string na sequência.

3. Implemente uma função que compara duas strings sem usar strcmp. Exemplo: Entrada: ”abc”e ”abc”Saída: Strings são iguais Solução: Percorra ambos os arrays simultaneamente, comparando caractere por caractere até encontrar uma diferença ou até o final das strings.

4. Remover caracteres duplicados consecutivos. Exemplo: Entrada: ”aaabbbcc”Saída: ”abc”Solução: Percorra a string e copie cada caractere apenas se ele for diferente do anterior.

5. Implemente uma função que encontra o maior prefixo comum entre duas strings. Exemplo: Entrada: ”prefeitura”e ”preferido”Saída: ”prefe”Solução: Percorra as strings simultaneamente e compare os caracteres até encontrar uma diferença.

6. Escreva uma função que verifica se duas strings são anagramas (têm os mesmos caracteres em qualquer ordem). Exemplo: Entrada: ”amor”e ”roma”Saída: São anagramas Solução: Conte a frequência de cada caractere nas duas strings e compare os resultados.

7. Escreva uma função que conta o número de palavras em uma string. Considere que as palavras são separadas por espaços. Exemplo: Entrada: ”hello world”Saída: 2 Solução: Percorra a string contando o número de espaços, incrementando o contador de palavras quando uma sequência de caracteres é interrompida por um espaço.

8. Escreva uma função que inverte as palavras de uma string, mantendo cada palavra intacta, mas mudando sua ordem. Exemplo: Entrada: ”hello world”Saída: ”world hello”Solução: Identifique cada palavra e a armazene temporariamente, depois escreva cada palavra na ordem reversa.

9. Escreva uma função que conta o número de palavras em uma string. Considere que as palavras são separadas por espaços. Exemplo: Entrada: ”hello world”Saída: 2 Solução: Percorra a string contando o número de espaços, incrementando o contador de palavras quando uma sequência de caracteres é interrompida por um espaço.