1 Lê *strings* (+++)

 (+++)

Essa questão é opcional.

"Chega!!! Estou farto dos 'problemas' do `scanf` com o '%s'. Preciso de uma função mais estável para usar em meus códigos" . Já é sabido que a função `scanf` para ler *strings* apresenta problemas por conta de caracteres residuais de outras leituras no terminal. Ao pressionarmos ENTER no terminal, são enviados dois caracteres, o "\r\n", com os respectivos códigos ASCII: 10, 13, onde '\n' indica o final de texto enviado pelo terminal. Por exemplo, ao digitarmos o texto "123"+ENTER no terminal, a sequência de caracteres passada é "123\r\n".

A função `scanf` processa o texto enviado pelo terminal de acordo com o código do formato passado como parâmetro. O '\n' é consumido e o texto residual "123\r" é processado pelo `scanf`. Ao processar o texto "123\r"com o código de formado "%d", a função `scanf` busca de uma sequência de caracteres da esquerda para a direita que representa um número inteiro até encontrar um caracter que não seja um dígito. O processamento resulta no número 123 e sobra o caracter '\r', que irá compor o próximo texto a ser lido no terminal.

Ao digitar um novo texto, por exemplo, "abc"+ENTER, o texto enviado pelo terminal torna-se "\rabc\r\n". Aí começam os problemas... Ao ler um número inteiro ou real, o '\r' é consumido e ignorado, mas ao ler um caracter ou uma *string* o '\r' é considerado. Além disso, ele indica fim de texto para a leitura de *strings*. Para a leitura de caracteres e *strings*, as expressões "%c", "%s"e "%[^\n]"precisam lidar com o problema do '\r'. O código abaixo, por exemplo, ao receber o texto "88"+ENTER via terminal, termina com a impressão "n: 88, c: 10", sem esperar pela leitura do caracter `ch`.

```c
int n;
char ch;
scanf("%d", &n);
scanf("%c", &ch);
printf("n: %d, c: %d\n", n, ch);
```

Note que uma ligeira alteração no texto de entrada resolve esse problema para `ch`. Se informarmos o texto "88a"+ENTER via terminal, o código termina com a impressão "n: 88, c: 97", isso porque passamos o caracter 'a' antes do '\r'. O '\r' será um problema para a próxima leitura de um caracter ou *string*.

Uma alternativa a esse problema é incluir uma leitura de caracter adicional sempre que for necessário ler um caracter ou uma *string*, mas essa não é uma alternativa elegante.

Outra limitação do `scanf` é a característica de não ler espaços no início de uma *string*, o que pode ser um problema dependendo da aplicação que se deseja implementar. Além disso, não é seguro ler uma *string* uma vez que a função não sabe a quantidade de caracteres que o vetor pode armazenar.

Na verdade, as características do `scanf` citadas neste texto como "problemas"são comportamentos propositalmente implementados, que podem ser configurados para resolver esses e outros problemas mais complexos na leitura de *strings*.

Em vez de aprender a usar as expressões entendidas pelo `scanf`, faça uma função que leia *strings* de modo a não sofrer com problemas de caracteres residuais e que também permita a leitura de espaços. Adicionalmente, esse função deve retornar a quantidade de caracteres lidos e limitar a quantidade de carateres a serem lidos. Os caracteres "\r\n"não podem compor a *string* final. Lembre-se que uma *string* precisa do '\0' para indicar seu final. O protótipo da função deve ser o seguinte:

```c
/**
```

```
 * @param str vetor de caracteres onde a string lida será gravada
 * @param n quantidade máxima de caracteres a ser lidos
 * @return quandidade de caracteres lidos
 */
int le_string( char * str, int n );
```

Seu programa principal deve ser o listado abaixo. A função `print_codes` imprime os códigos ASCII de cada caracter da *string* passada via parâmetro. Seu protótipo é:

```
/**
 * @param str string de entrada
 */
void print_codes( char * str );
```

Seu programa principal deve ser o listado abaixo. Esse código avalia se a função `le_string` consegue lidar com o caractere '\r'.

```
#define N 128+1

int main() {
    char str[N], s[N];
    char c;
    int i;

    scanf("%c", &c);
    le_string(str, 3);
    print_codes(str);
    printf("caracter:%c, str:%s\n", c, str);

    scanf("%c", &c);
    le_string(str, 5);
    print_codes(str);
    printf("caracter:%c, str:%s\n", c, str);

    scanf("%c", &c);
    le_string(str, 5);
    print_codes(str);
    printf("caracter:%c, str:%s\n", c, str);

    scanf("%d", &i);
    le_string(str, 3);
    print_codes(str);
    printf("inteiro:%d, str:%s\n", i, str);

    //printf("Digite inteiros separados por espaco: ");
    scanf("%d", &i);
    //printf("inteiro:%d\n", i);
    //printf("Le string (15):\n");
    le_string(str, 15);
    print_codes(str);
    printf("inteiro:%d, str:%s\n", i, str);

    //printf("Digite uma string sem espacos: ");
    scanf("%s", s);
    //printf("string:%s\n", str);
    //printf("Le string (10):\n");
    le_string(str, 100);
    print_codes(str);
```

```
42      printf("string:%s, str:%s\n", s, str);
43
44      //printf("Digite uma string com espacos: ");
45      scanf("%s", s);
46      //printf("string:%s\n", str);
47      //printf("Le string (20):\n");
48      le_string(str, 100);
49      print_codes(str);
50      printf("string:%s, str:%s\n", s, str);
51
52      return 0;
53  }
```

Entrada

- Um caracter + ENTER
- Texto com espaços + ENTER
- Sequência de caracteres sem espaços + ENTER
- Sequência de caracteres separados por espaços + ENTER
- Um número inteiro + ENTER
- Texto com espaços + ENTER
- Inteiros separados por espaços + ENTER
- Texto sem espaços + ENTER
- Texto com espaços + ENTER
- Texto com espaços + ENTER

Saída

Uma linha com os códigos de cada caractere da `str` lida pela função `le_string` seguida por uma linha com a apresentação do conteúdo do dado lido e da *string* `str`.

Observações

DICA: Você pode fazer uma função que leia caracter por caracter usando o "%c". Lembre-se que o texto passado pelo terminal é uma sequência de caracteres.

Exemplo

Entrada	Saída
x texto 123 abcdef x y z h w 99 texto com espacos 11 22 33 44 texto_sem_espacos texto2 com espacos Text with spaces	116,101,120 caracter:x, str:tex 98,99,100,101,102 caracter:a, str:bcdef 32,121,32,122,32 caracter:x, str: y z 116,101,120 inteiro:99, str:tex 32,50,50,32,51,51,32,52,52 inteiro:11, str: 22 33 44 116,101,120,116,111,50,32,99,111,109,32,101,115,112,97,99 string:texto_sem_espacos, str:texto2 com espacos 32,119,105,116,104,32,115,112,97,99,101,115 string:Text, str: with spaces