Aula 06 - Strings

Assuntos

- Char e strings
- Matrizes e ponteiros
- `string.h` : funções para strings

Char e strings

Um detalhe importante do C é que o char em geral segue a tabela ASCII, permitindo apenas 256 caracteres diferentes, incluindo letras minúsculas, maiúsculas, números, símbolos e caracteres de controle, por exemplo o `\n`.

Char e strings

A representação de string e char é diferente em C:

- 'A' é um character que contém a letra A
- "A" é uma string que contem 2 caracteres, a letra A e um `NULL` (`\0`) no final

Em C, toda string é terminada pelo caracter nulo (primeiro caracter da tabela ASCII)

Char e strings

Se toda string é terminada por um caracter nulo, então a string "Hello world" tem 12 caracteres.

0	1	2	3	4	5	6	7	8	9	10	11
'H'	'e'	'l'	'l'	'o'	' '	'w'	'o'	'r'	'l'	'd'	'\0'

```
char str[12] = "Hello world";
char str2[12] = {'H', 'e', 'l', 'l', 'o', ' ',
                 'w', 'o', 'r', 'l', 'd', NULL};
```

Matrizes e ponteiros

Se uma string é um array de `char`, então uma matriz de `char` é um array de strings

```
// 12 arrays de até 9 caracteres e 1 NULL
char meses[12][10];

strcpy(meses[0], "Janeiro");  // strcpy copia string do segundo
strcpy(meses[1], "Fevereiro");// argumento para o primeiro
...
strcpy(meses[11], "Dezembro");
```

Matrizes e ponteiros

Ou podemos fazer um ponteiro para um array de `char`:

```
char *meses[12]; // 12 ponteiros de caracter

meses[0] = "Janeiro";   // não precisamos do strcpy
meses[1] = "Fevereiro"; // pois apontamos para a string
...
meses[11] = "Dezembro";
```

Funções com strings

- C possui o `string.h` que implementa algumas funções (por volta de 20) para trabalhar com strings.
- Além de algumas funções de entrada e saída de dados voltadas para strings

Leitura de strings

- `scanf("%s", &var)` : recebe o que foi digitado até o primeiro caracter em branco e armazena em `var` .
- O problema do `scanf` é o critério usado para descobrir onde a string termina, podendo ser um caracter em branco ou quebra de linha. Para resolver isso, podemos usar `scanf("%[^\n]s", &var)`

Leitura de strings

- `gets(var)` : recebe a string digitada e armazena em `var` .
- Não é mais usado por problema de gerenciamento de memória: sendo uma string um array de caracteres que ocupa um espaço contíguo na memória, se a string digitada for maior do que o espaço reservado para a string, `gets` pode sobrescrever nos espaços seguintes...

Leitura de strings

- `fgets(var, n, stdin)` : recebe `n` caracteres de `stdin` e armazena em `var` .
- Problema: se a string digitada for maior que `n` , o resto da string é descartada.

Exibição de strings

- `printf("%s", var)` : exibe o conteúdo de `var` como uma string. Permite que mais do que uma string seja exibida no mesmo comando.
- `puts(var)` : exibe o conteúdo de `var` como uma string e adiciona um caracter de quebra de linha ao final

Funções em `string.h`

- `int strlen(char *)` : percorre uma string até encontrar o caracter `\0` e retorna a quantidade de caracteres encontrados.

Exemplo

```
char *str = "Hello, world!";
int len = strlen(str);
for(int i=0; i<len; i++)
    printf("%c", str[i]);
```

```
char *str = "Hello, world!";
for(int i=0; i<strlen(str); i++)
    printf("%c", str[i]);
```

Qual a diferença?

Exemplo

```
char *str = "Hello, world!";
int len = strlen(str);
for(int i=0; i<len; i++)
    printf("%c", str[i]);
```

No primeiro caso, o tamanho da string é calculado apenas uma vez e usado no loop.

Exemplo

```
char *str = "Hello, world!";
for(int i=0; i<strlen(str); i++)
    printf("%c", str[i]);
```

No segundo caso, o comprimento da string é calculada a cada passo do loop. Sendo que `strlen` também é um loop, temos um loop dentro de um outro loop.

Exemplo

Portanto, o primeiro código tem desempenho melhor que o segundo e escala melhor que o segundo conforme o tamanho da string.

Isto se aplica para C, para outras linguagens devemos ver a documentação da linguagem.

Funções em `string.h`

- `char* strcpy(char*, char*)` : copia a segunda string no endereço da primeira
- `char* strcat(char*, char*)` : concatena a segunda string com a primeira
- `int strcmp(char*, char*)` : compara as duas strings retornando:
  - `0` se forem iguais
  - `>0` se o primeiro caracter diferente estiver na segunda string
  - `<0` se o primeiro caracter diferente estiver na primeira

Exemplo

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

int main(){
    char str[12] = "Hello world";
    char str2[12] = {
        'H', 'e', 'l', 'l', 'o', ' ',
        'w', 'o', 'r', 'l', 'd', NULL};

    int r = strcmp(str, str2);
    printf("%d\n", r);
}
```

Funções em `string.h`

- `char* strupr(char*)` : converte a string para caracteres em maiúsculo
- `char* strlwr(char*)` : converte a string para caracteres em minúsculo
- `int atoi(const char *str)` : converte `str` para um `int`
- `float atof(const char *str)` : converte `str` para `float` (depende de `stdlib.h`)

Exemplo

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

int main(){
    char *c = "1";
    int i = atoi(c);
    double f = atof(c);
    printf("%d\n%f\n", i, f);
}
```

Funções em `string.h`

- `int sprintf(char *s, const char *format, ...)` : escreve a string formatada na string `s`
- `int sscanf(const char *s, const char *format, ...)` : armazena a string formatada em `s`
- `char* strtok(char *s, cons char *delim)` : retorna a string `s` particionada até o delimitador `delim`

Exemplo

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

int main(){
    int dia = 1;
    int mes = 10;
    int ano = 2022;

    char str[11];
    sprintf(str, "%02d/%02d/%04d", dia, mes, ano);
    printf("%s\n", str);
}
```

Exemplo

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

int main(){
    int dia, mes, ano;
    char str[11] = "01/10/2022";
    printf("%s\n", str);

    sscanf(str, "%02d/%02d/%04d", &dia, &mes, &ano);
    printf("%d %d %d\n", dia, mes, ano);
}
```

Funções em `string.h`

Existem mais funções em `string.h`, mas não vamos passar por todas...