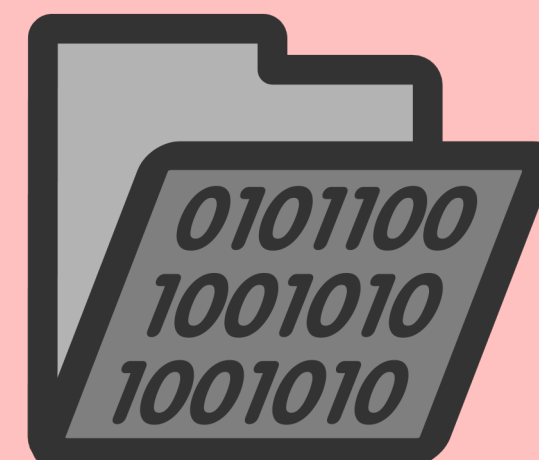


Processamento de Texto e Ficheiros

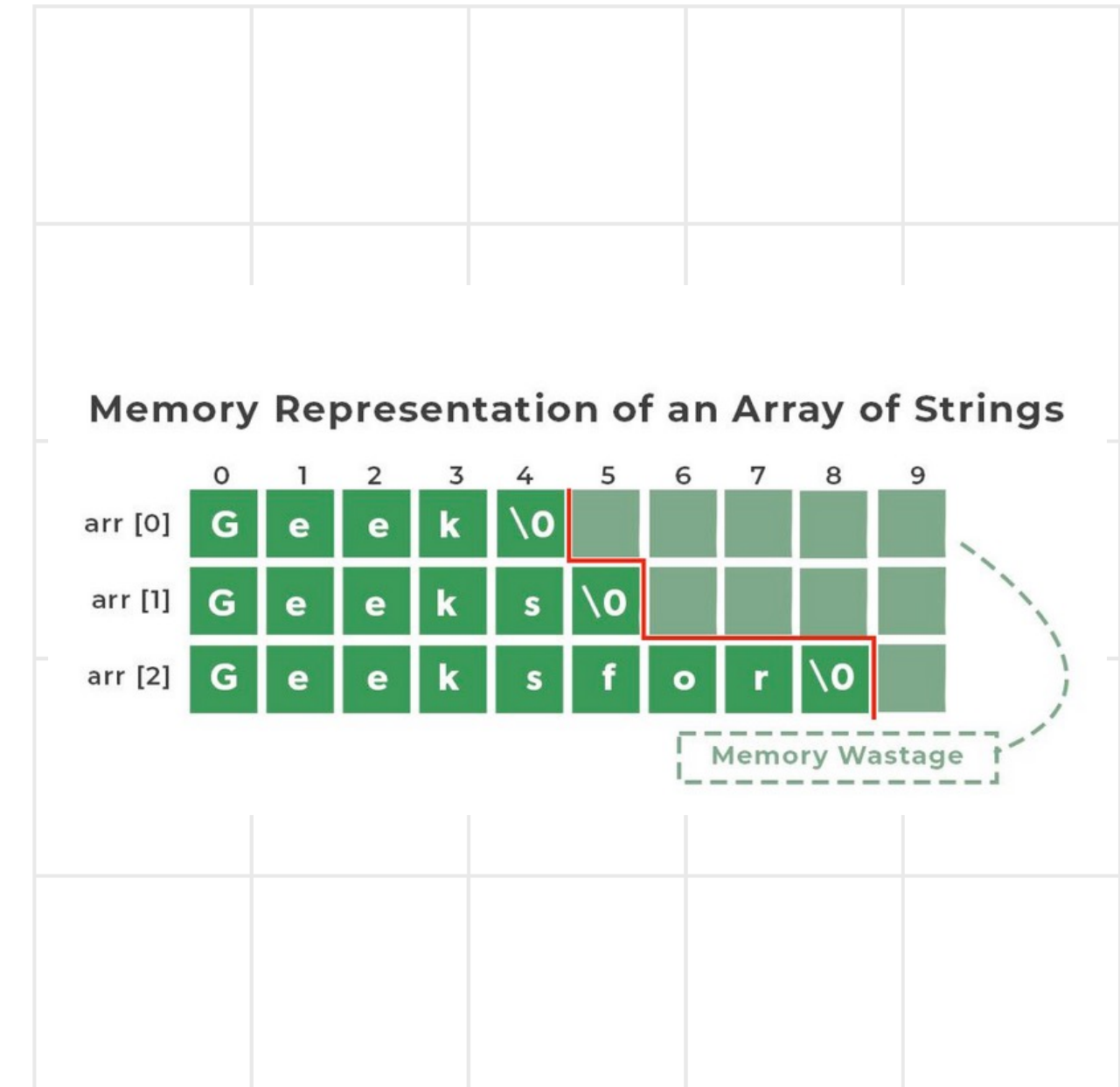
(usando funções da Standard C library)



Bem-vindos à aula!

Agenda de Hoje

- Argumentos da função main
- Strings
- Processamento de ficheiros



Argumentos da função main()



maintest.c

```
#include <stdio.h>

int main (int argc, char* argv[])
{
    ...
    ...
    ...
    return 0;
}
```

Argumentos da função `main()`



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

```
int main (int argc, char* argv[])  
{  
    ...  
    ...  
    ...  
    return 0;  
}
```

argc: ARGument Count

- Variável **int** que armazena a quantidade de argumentos passados pela linha de comando.
- O valor de **argc** não pode ser negativo.

Argumentos da função main()



```
$ gcc -Wall mainteste.c -o maintest  
$ ./maintest
```

```
int main (int argc, char* argv[])  
{  
    ...  
    ...  
    ...  
    return 0;  
}
```

argc = 1

Argumentos da função main()



```
$ gcc -Wall mainteste.c -o maintest  
$ ./maintest mercury
```

```
int main (int argc, char* argv[])  
{  
    ...  
    ...  
    ...  
    return 0;  
}
```

argc = 2

Argumentos da função main()



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

```
int main (int argc, char* argv[])  
{  
    ...  
    ...  
    ...  
    return 0;  
}
```

argv: ARGument Vector
- Vector de strings que armazena todos os argumentos passados via linha de comando.

Argumentos da função main()



```
$ gcc -Wall mainteste.c -o maintest  
$ ./maintest mercury
```

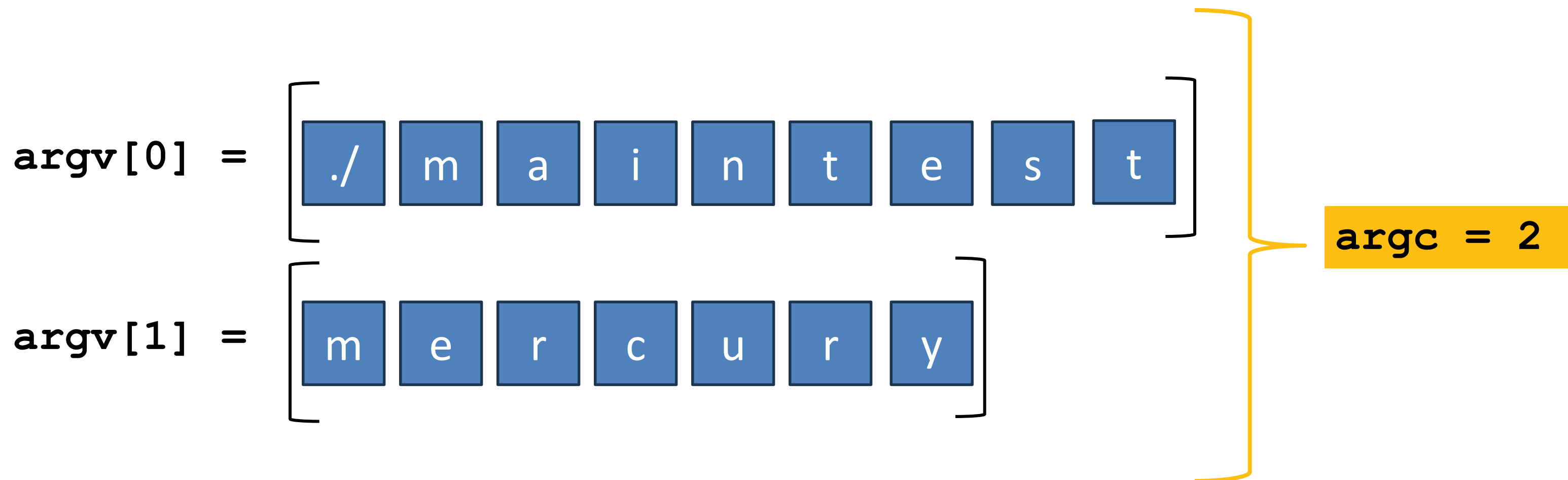
```
int main (int argc, char* argv[])  
{  
    ...  
    ...  
    ...  
    return 0;  
}
```

argv[0] = “./maintest”
argv[1] = “mercury”
argv[argc] = NULL

Como fica representação de argv?

- **Vector de tamanho argc**

```
$ ./maintest mercury
```





Tente e Aprenda

Hora da Atividade

Ficha 3 – Exercício 1



Challenge:

E se os parâmetros forem numéricos?

```
$ gcc -Wall desafio.c -o desafio  
$ ./desafio 105 90 503
```

Strings



test_strings.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_SIZE 64

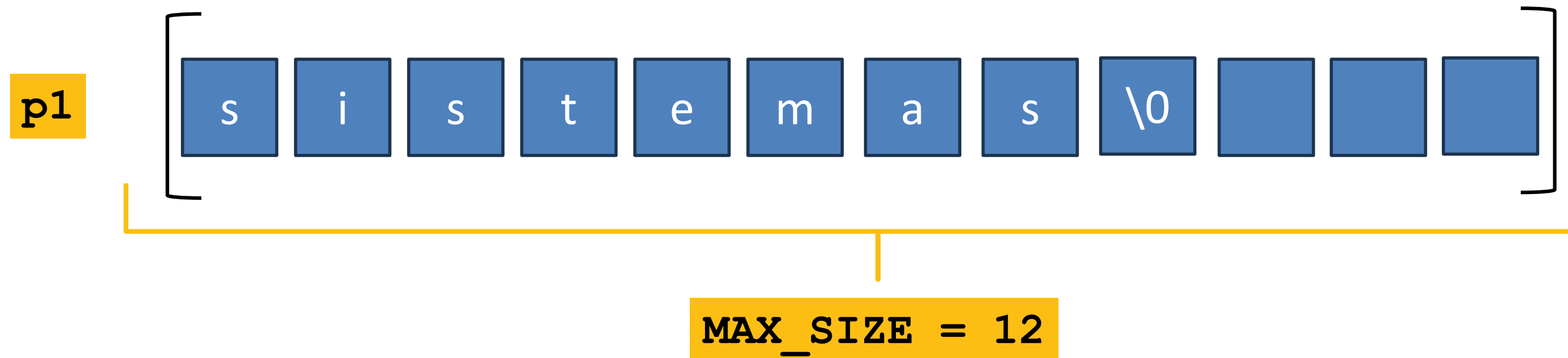
int main (int argc, char* argv[]) {
    char* p1 = (char*)malloc(MAX_SIZE * sizeof(char));
    char *p2 = (char*)malloc(MAX_SIZE * sizeof(char));

    ...

    exit(EXIT_SUCCESS);
}
```

Strings

```
char* p1 = (char*)malloc(MAX_SIZE * sizeof(char));
```



O endereço de uma string é o endereço do seu primeiro byte.

Strings

```
char* p1 = (char*)malloc(MAX_SIZE * sizeof(char));  
char* p2 = (char*)malloc(MAX_SIZE * sizeof(char));
```

p1

s i s t e m a s \0

MAX_SIZE

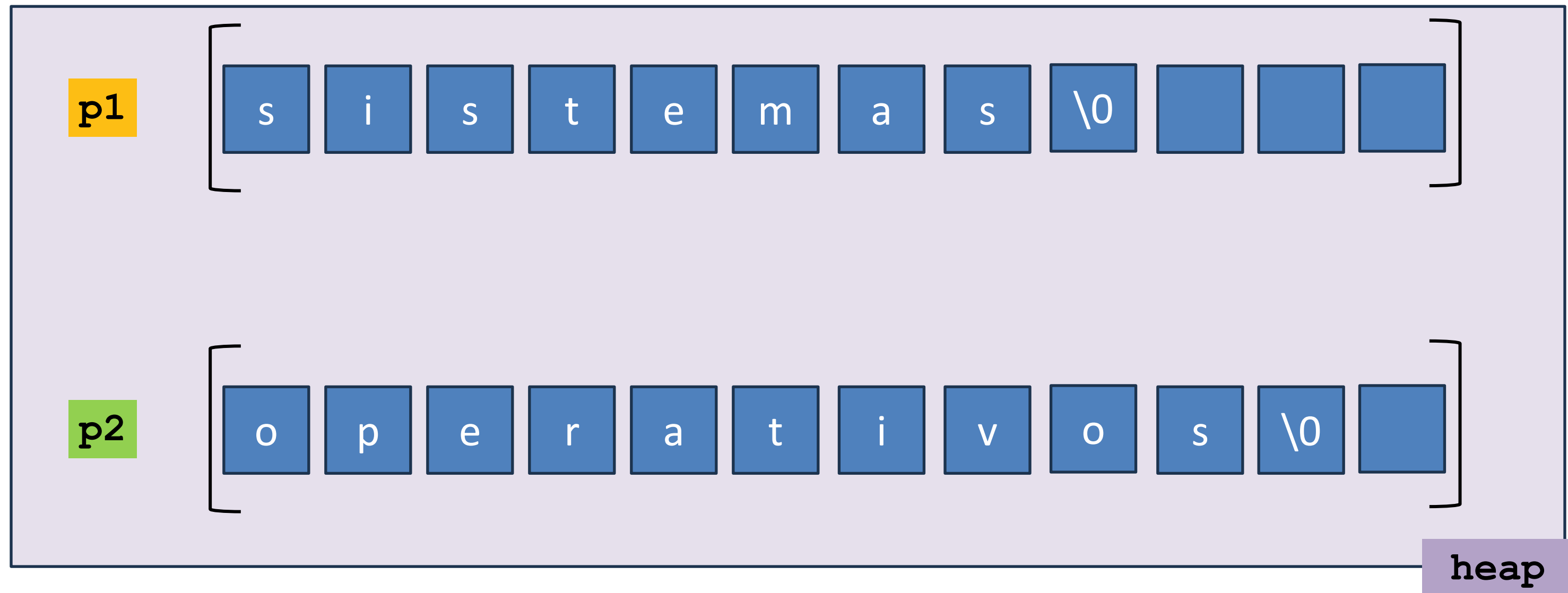
p2

o p e r a t i v o s \0

MAX_SIZE

Strings

```
char* p1 = (char*)malloc(MAX_SIZE * sizeof(char));  
char* p2 = (char*)malloc(MAX_SIZE * sizeof(char));
```



Strings

Conceito e Definição

Em C, uma string pode ser referida tanto usando um apontador para caracteres como um array de caracteres.

API

<string.h>

Parte da Biblioteca Standard C (clib)

Delimitador

\0

<string.h>

- O ficheiro de cabeçalho **string.h** é um ficheiro standard da linguagem C que contém funções para manipulação de strings (arrays de caracteres).
- O ficheiro de cabeçalho **<string.h>** inclui várias funções úteis para manipular strings que podem ser usadas diretamente num programa através da diretiva de pré-processador **#include**.



<string.h>



| Nome da Função | Descrição da Função |
|-----------------------|--|
| <code>strdup()</code> | Aloca memória dinamicamente para a nova string e faz a cópia. |
| <code>strcpy()</code> | Não aloca memória, apenas copia uma string de origem para um buffer já alocado. |
| <code>strcat()</code> | Concatena duas strings |
| <code>strstr()</code> | Encontra um substring em string |
| <code>strtok()</code> | Divide a string fornecida em tokens com base em algum caractere como delimitador |
| <code>strsep()</code> | Divide uma string em tokens, com base em um conjunto de caracteres delimitadores |
| <code>strlen()</code> | Retorna o tamanho de uma string |

strlen()

Syntaxe:

```
size_t strlen(const char* str);
```

unsigned int



Tamanho do string

Não inclui

O comprimento (= *length*) de uma string é o seu número de bytes, sem contar o byte nulo final. Assim a string do exemplo tem comprimento 8. Cada byte de uma string é tratado como um [char](#) e portanto uma string é um vetor de chars.

strlen()

Exemplo:

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

int main()
{
    char str[] = "SistemasOperativos";

    int length = strlen(str);
    printf("%d", length);
    return 0;
}
```

Output:

18

strcpy()

Syntaxe:

```
char * strcpy( char * destino, char * origem );
```

Exemplo:

origem

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----|
| s | i | s | t | e | m | a | s | \0 |
|---|---|---|---|---|---|---|---|----|

```
char destino[9];  
strcpy (destino, origem);
```

destino

| | | | | | | | | |
|---|---|---|---|---|---|---|---|----|
| s | i | s | t | e | m | a | s | \0 |
|---|---|---|---|---|---|---|---|----|

A função `strcpy` recebe duas strings e copia a `origem` (inclusive o byte nulo final) para o espaço ocupado pelo `destino`. Não invoque a função se o comprimento da string de `destino` for menor que a `origem`.
(*Buffer overflow* é uma das mais comuns origens de bugs de segurança!)

strcmp()

Syntax:

```
int strcmp( const char * string1, const char * string2 );
```

Exemplo:

| | | | | | | | | | |
|----------------|---|---|---|---|---|---|---|---|----|
| string1 | s | i | s | t | e | m | a | s | \0 |
| string2 | s | i | s | t | e | m | a | s | \0 |

```
int result = strcmp(string1, string2);
```

Output:

```
0
```

A função `strcmp` compara duas strings lexicograficamente, byte-a-byte.

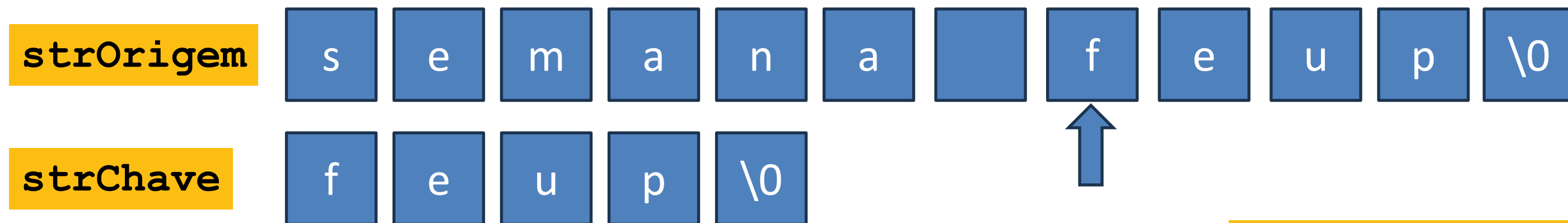
A função devolve um número negativo se a primeira string for lexicograficamente menor que a segunda, devolve 0 se as duas strings são iguais, e devolve um número positivo se a primeira string for maior que a segunda.

strstr()

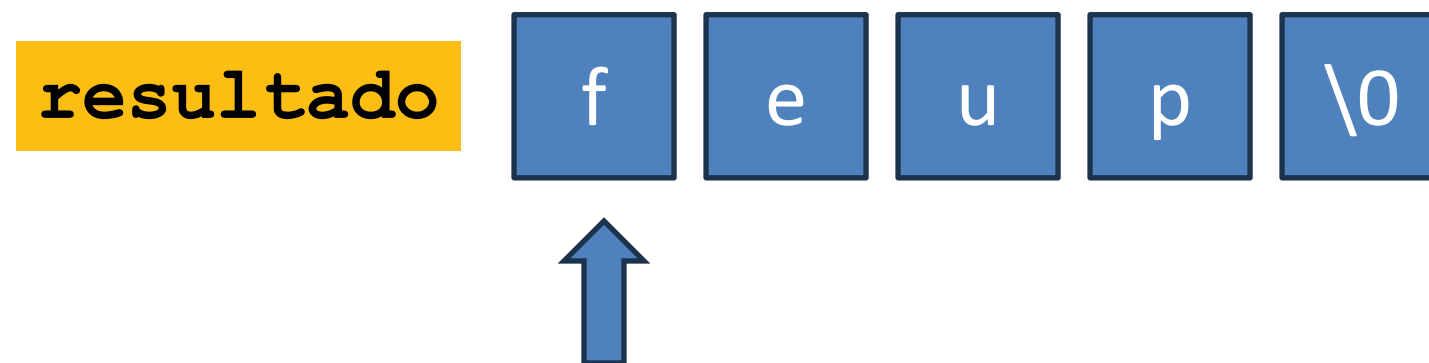
Sintaxe:

```
char * strstr(const char* strOrigem, char* strChave);
```

Exemplo:



```
char *resultado = strstr(strOrigem, strChave);
```



- Devolve um apontador para a primeira ocorrência da string apontada por `strChave` na string apontada por `strOrigem`.
- Devolve NULL se não for encontrada nenhuma coincidência.

strcat()

Syntaxe:

```
char * strcat( char * stringDestino, char * stringOrigem );
```

Exemplo:

destino

s i s t e m a s \0

origem

o p e r a t i v o s \0

```
strcat (destino, origem);
```

Output:

s i s t e m a s o p e r a t i v o s \0

Esta função irá concatenar a segunda string ao final da primeira string.

O primeiro parâmetro da função portanto deve ser uma variável e possuir o espaço suficiente para o resultado.

A função **não** irá testar se existe espaço fazendo a movimentação de caracteres do segundo parâmetro para o final do primeiro.

strcat()

Problemas:

- Sobrescrita de memória (buffer overflow)
- Erros de segmentação (segmentation fault)

Exemplo:

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

```
int main() {
    char destino[9] = "Hello";
    char origem[] = "world";
```

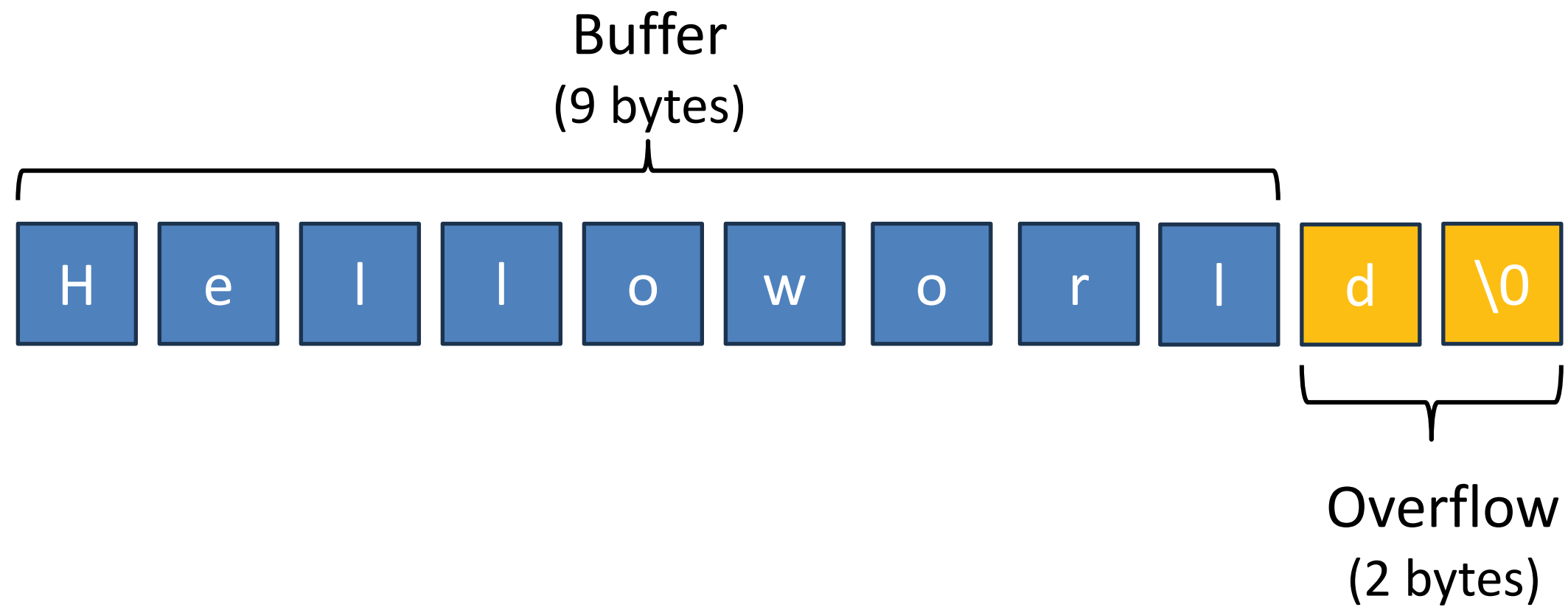
```
    strcat(destino, origem);
```

```
    printf("%s\n", destino);
    return 0;
```

```
}
```

O buffer destino tem espaço para apenas 9 caracteres, mas a string concatenada "Helloworld" precisa de 11 caracteres (incluindo o caractere nulo \0). Isso pode causar sobrescrita de memória.

strcat()



Problema de segurança: Ataque buffer overflow

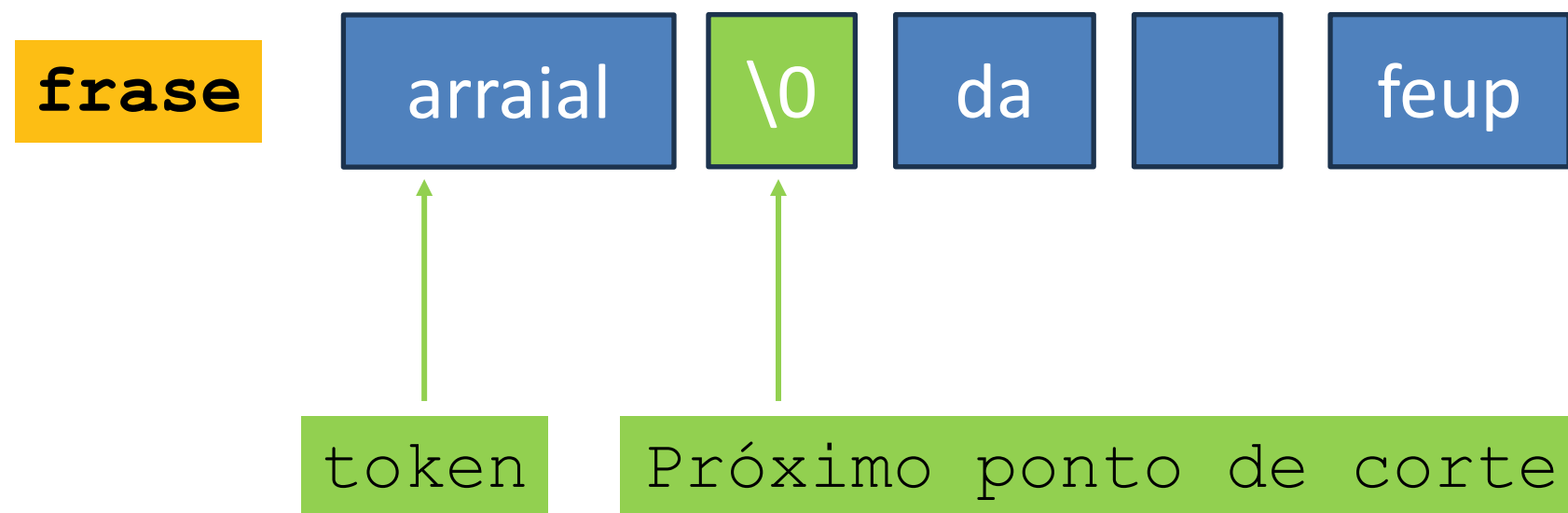
strtok()

Syntaxe:

```
char * strtok(char * strOrigem, char * strDelimitador);
```

Exemplo:

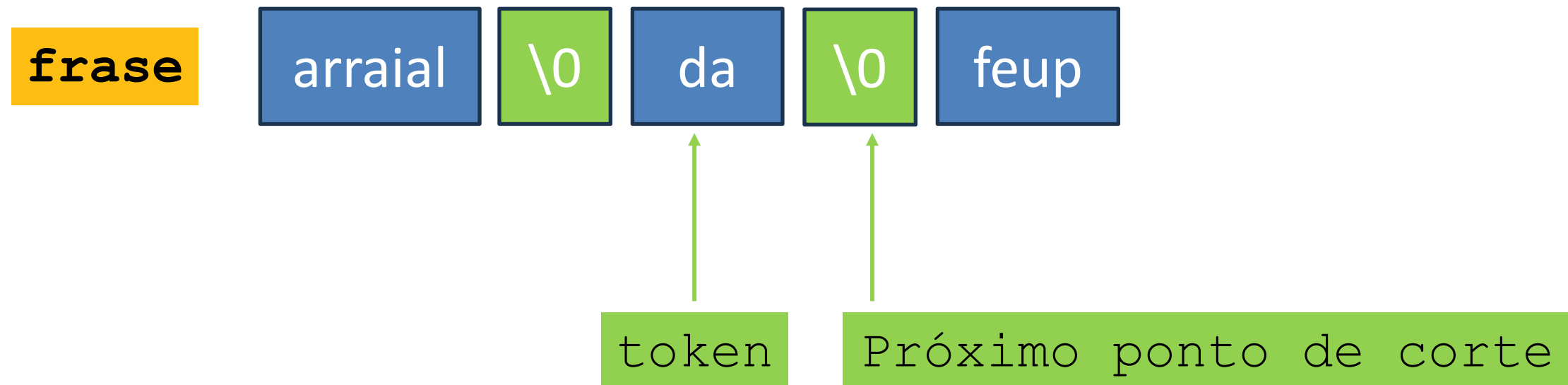
```
char frase[] = "arraial da feup";  
char *token;  
// Primeira chamada à strtok: passa a string e o delimitador (espaço)  
token = strtok(frase, " ");
```



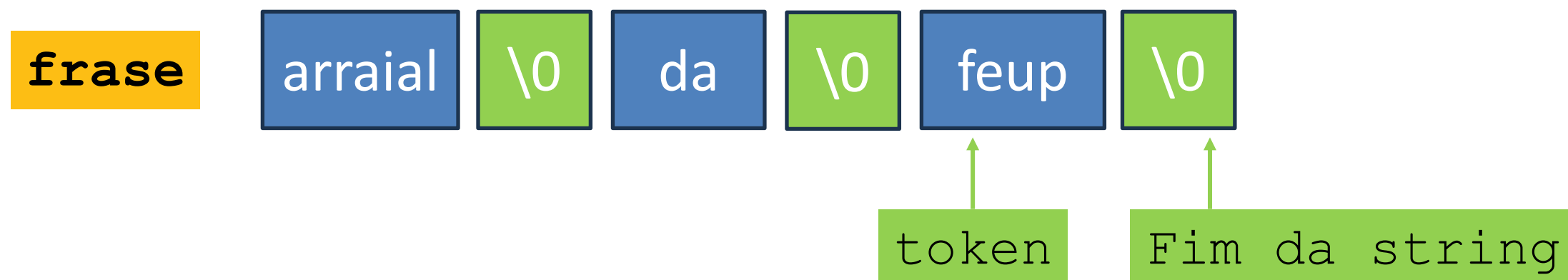
strtok()

Exemplo:

```
// Segunda chamada passa NULL para continuar a partir da string original  
token = strtok(NULL, " ");
```



```
// Última chamada não encontra mais nenhum delimitador na string  
token = strtok(NULL, " ");
```



A função retorna NULL
quando não há mais
tokens



Tente e Aprenda

Hora da Atividade

Ficha 3 – Exercício 2



Processamento de Ficheiros



filetest.c

```
#include <stdio.h>
#include <stdlib.h>
#define BUFFER_SIZE 1024

int main (int argc, char* argv[]) {
    FILE* file = fopen(argv[1], "r");
    char buffer[BUFFER_SIZE];
    int nchars = fread(buffer, sizeof(char),
BUFFER_SIZE, file);
    while (nchars > 0) {
        fwrite(buffer, sizeof(char), nchars, stdout);
        nchars=fread(buffer, sizeof(char),
BUFFER_SIZE, file);
    }
    fclose(file);
}
```

fopen()

Sintaxe

```
FILE *fopen(const char *filename, const char *mode);
```

Parâmetros

filename: nome do ficheiro a ser aberto.

mode: especifica o modo de abertura do ficheiro.

Modos comuns de abertura

r: abre o ficheiro para leitura

w: abre o ficheiro para escrita

Retorno

Sucesso: Retorna um apontador para o tipo FILE

Falha: Retorna NULL

- A função é usada para abrir um ficheiro e associá-lo a um fluxo
- Está definida no cabeçalho `<stdio.h>`
- Retorna um apontador para o tipo `FILE`

fread()

Sintaxe

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Parâmetros

ptr: apontador para o bloco de memória onde os dados lidos serão armazenados.

size: tamanho (em bytes) de cada elemento a ser lido.

nmemb: número de elementos que deseja ler

stream: apontador para o ficheiro

- A função é usada para ler dados de um ficheiro para um bloco de memória
- Está definida no cabeçalho `<stdio.h>`

Retorno

Sucesso: Retorna o número total de elementos lidos com sucesso.

Falha: Se ocorrer um erro ou o fim do ficheiro for atingido antes de qualquer elemento ser lido, a função retorna 0.

fseek()

Sintaxe

```
int fseek(FILE *stream, long int offset, int origin);
```

Parâmetros

stream: apontador para o ficheiro `FILE`

offset: número de bytes a mover o apontador de leitura/escrita.

origin: define o ponto de referência para o deslocamento (offset). Há 3 valores possíveis:

- `SEEK_SET`: início do ficheiro
- `SEEK_CUR`: posição atual do ficheiro
- `SEEK_END`: final do ficheiro

Retorno

Sucesso: retorna 0

Falha: retorna -1 em caso de erro

- A função é usada para mover o apontador de leitura/escrita do ficheiro para uma nova posição.
- Permite saltar para diferentes partes do ficheiro sem ler sequencialmente.
- Está definida no cabeçalho `<stdio.h>`

fwrite()

Sintaxe

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Parâmetros

ptr: apontador para onde estão os dados a serem lidos

size: tamanho (bytes) de cada elemento a ser lido

nmemb: número de elementos a serem escritos

stream: apontador para o ficheiro onde estão os dados a serem escritos. Posso usar a flag `stdout` para escrever diretamente no terminal.

Retorno

Sucesso: retorna o número total de elementos escritos

Falha: retorna um valor inferior ao esperado

- A função é utilizada para escrever dados binários a partir de uma área de memória para um ficheiro.
- Está definida no cabeçalho `<stdio.h>`

fclose()

Sintaxe

```
int fclose(FILE *stream);
```

Parâmetro

stream: apontador para o ficheiro `FILE`

Retorno

Sucesso: retorna 0

Falha: EOF (End Of File) se ocorrer erro ao fechar

- Fecha um ficheiro que foi previamente aberto com `fopen()` e libera todos os recursos associados a ele
- No caso de operações de escrita, todos os dados que estão no buffer são gravados em disco
- O ficheiro é liberado para que outros processos possam usá-lo, ou o próprio programa em outra chamada `fopen()`
- `<stdio.h>`



Tente e Aprenda

Hora da Atividade

Ficha 3 – Exercício 3 – 8*



**E por hoje
terminamos!**