# C-Strings and Valgrind

COMP201 Lab 3
Spring 2023

KOÇ UNIVERSITY

Osman Batur İnce

# Valgrind

Valgrind is a programming tool used for:

- memory debugging
- memory leak detection
- profiling

# Some Valgrind Use Cases

```c
#include <stdlib.h>

// main_1.c
int main(){
        char *x = malloc(100);
        return 0;
}
```

**Memory Allocated but Never Used**

```c
#include <stdlib.h>

// main_2.c
int main(){
        char *x = malloc(10);
        x[10] = 'C';
        return 0;
}
```

**Finding Invalid Pointer Use With Valgrind**

**Compile the code files:**

```
gcc -std=gnu99 -g -o main_1 main_1.c
gcc -std=gnu99 -g -o main_2 main_2.c
```

**-g** is needed to see line number with Valgrind when an error happens.

# Some Valgrind Use Cases

```
valgrind --tool=memcheck --leak-check=yes ./{exec_name}
```

### 100 Bytes Allocated but Never Used

…
**100 bytes in 1 blocks are definitely lost in loss record 1 of 1**
at 0x4C29F73: malloc (vg_replace_malloc.c:309)
by 0x40053E: main (**main_1.c:5**)
…

### Finding Invalid Pointer Use With Valgrind

…
**Invalid write** of size 1 at 0x40054B: main (main_2.c:6)
Address 0x520504a is 0 bytes after a block of size 10 alloc'd
at 0x4C29F73: malloc (vg_replace_malloc.c:309)
by 0x40053E: main (**main_2.c:5**)
…

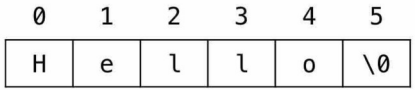
# Strings in C

KOÇ UNIVERSITY

# C-Strings

- 1D array of characters
- Terminated by **null** or '**\0**' character
- Initializing a string
  - char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  - char str[6] = "Hello";
  - char str[] = "Hello";
  - char *str = "Hello";

```
char str[6] = “Hello”;
```




# C-String Functions

| S.no | String functions | Description |
|---|---|---|
| 1 | strcat(str1, str2) | Concatenates str2 at the end of str1. |
| 2 | strcpy(str1, str2) | Copies str2 into str1 |
| 3 | strlen(strl) | gives the length of str1. |
| 4 | strcmp(str1, str2) | Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2. |
| 5 | strchr(str1, char) | Returns pointer to first occurrence of char in str1. |
| 6 | strstr(str1, str2) | Returns pointer to first occurrence of str2 in str1. |
| 7 | strcmpi(str1,str2) | Same as strcmp() function. But, this function negotiates case. “A” and “a” are treated as same. |
| 8 | strdup() | duplicates the string |
| 9 | strlwr() | converts string to lowercase |
| 10 | strncat() | appends a portion of string to another |
| 11 | strncpy() | copies given number of characters of one string to another |
| 12 | strrchr() | last occurrence of given character in a string is found |
| 13 | strrev() | reverses the given string |
| 14 | strset() | sets all character in a string to given character |
| 15 | strnset() | It sets the portion of characters in a string to given character |
| 16 | strupr() | converts string to uppercase |
| 17 | strtok() | tokenizing given string using delimiter |

# Using String Functions - 1

- **Finding length of the str1**

```
char str1[] = "Hello COMP201";
int len = strlen(str1);
printf("strlen(str1) : %d\n", len);
// prints strlen(str1) : 13
```

- **Concatenating two strings**

```
char str1[] = "COMP";
char str2[] = "201";
strcat(str1, str2);
printf("strcat(str1, str2): %s\n", str1);
// prints strcat(str1, str2): COMP201
```

# Using String Functions - 2

- **Converting str1 to lowercase**

```
char str1[] = "Hello COMP201";
char lwr[] = strlwr(str1);
printf("strlwr(str1) : %s\n", lwr);
// prints strlwr(str1) : hello comp201
```

- **Comparing two strings**

```
char str1[] = "COMP201";
char str2[] = "COMP201";
int eq = strcmp(str1, str2);
printf("strcmp(str1, str2): %d\n", eq);
// prints strcmp(str1, str2): 0
```

# Using String Functions - 3

- **Find the location of the first char in str1 which is not in str2**

```
char str1[] = "world";
char str2[] = "word";
int loc = strspn(str1, str2);
printf("loc : %d\n", loc);
// prints loc : 3
```

- **Find str2 inside str1**

```
char str1[] = "Impossible";
char str2[] = "possible";
char* substr = strstr(str1, str2);
printf("substr: %s\n", substr);
// prints substr : possible
```

# Strings in Memory

- A **string** is a **char array** in the memory. We can change each character because we can change contents of array.

- **Difference between char * and char []:**
  - When a string is created as a **char ***, its characters cannot be modified because its memory lives in the data segment (static memory). We can set a **char *** equal to another value, because it is a reassignable pointer.
  - We cannot set a **char[]** equal to another value, because it is not a pointer; it refers to the block of memory reserved for the original array. If we pass a **char[]** as a parameter, set something equal to it, or perform arithmetic with it, it's automatically converted to a **char ***.

# Treating Like an Array

- Find length without using **strlen()**

```
// Function strlen_2 counts the chars in the string str
// Returns the last index i
int strlen_2(char str[]) {
    int i = 0;
    while ( str[i] != '\0' ){
        i++;
    }
    return i;
}
```

# Arrays of Strings

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

- "argv" in the main function arguments is an array of strings
- Each memory location pointed by argv contains a string.
- The below arguments are equivalent and they are **double pointers**
  - **Double pointer:** A pointer containing memory location of another pointer
  - `void foo(char **strArr) { ... }`
  - `void bar(char *strArr[]) { ...}`

# Print chars of str in reverse order

```
void main(){
    char str[100]; // Declares a string of size 100
    int l, i;
    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    l = strlen(str);
    printf("The characters of the string in reverse are : \n");
    for(i = l-1; i >= 0; --i){ // do not forget null character
        printf("%c ", str[i]);
    }
    printf("\n");
}
```

# String Exercises

- **`lowerCase`:** Convert a string to lowercase without using `strlwr()`
  - Ex: `lowercase("HeLlO CoMp201") -> "hello comp201"`

- **`concat`:** Concatenate two strings manually
  - Ex: `concat("str one and", "str two") -> "str one and str two"`

- **`removeDup`:** Remove duplicate characters from a string
  - Ex: `removeDup("silence is a source of great") -> "silenc aourfgt"`

**Note:** To run exercises first run make then run the program with desired function:

```
./strings {func_name}
```