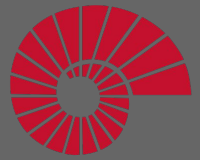
C-Strings and Valgrind

COMP201 Lab 3
Fall 2021



KOÇ
UNIVERSITY

Valgrind

Valgrind is a programming tool used for:

- memory debugging
- memory leak detection
- profiling

Memory Allocated but Never Used

main.c

```c
#include <stdlib.h>
int main()
{
    char *x = malloc(100);
    return 0;
}
```

Finding Invalid Pointer Use With Valgrind

main.c

```c
#include <stdlib.h>

int main()
{
    char *x = malloc(10);
    x[10] = 'a';
    return 0;
}
```

Valgrind Command

valgrind --tool=memcheck --leak-check=yes ./filename

Output:

When 100 bytes are allocated but not used

==2330== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2330== at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
==2330== by 0x804840F: main (main.c:5)

When Invalid pointer index is called

==767== Invalid write of size 1
==767== at 0x10916B: main (invalidPointer.c:6)

Valgrind Practice

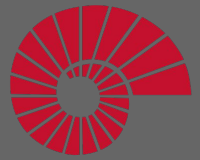
- **Memory Errors**
  invalidPointer.c
- **Memory Leaks**
  memoryLeak.c

```
$ gcc -g -o memoryLeak.o memoryLeak.c
$ valgrind --leak-check=yes ./memoryLeak.o
```

```
==807== LEAK SUMMARY:
==807==    definitely lost: 4,000 bytes in 1 blocks
==807==    indirectly lost: 0 bytes in 0 blocks
==807==      possibly lost: 0 bytes in 0 blocks
==807==    still reachable: 0 bytes in 0 blocks
==807==         suppressed: 0 bytes in 0 blocks
```

- Note: In order to check for memory leaks “--leak-check=yes” command must be given to the Valgrind. To determine which part of the code causes memory error/leak “--g” command must be given when compiling the program.

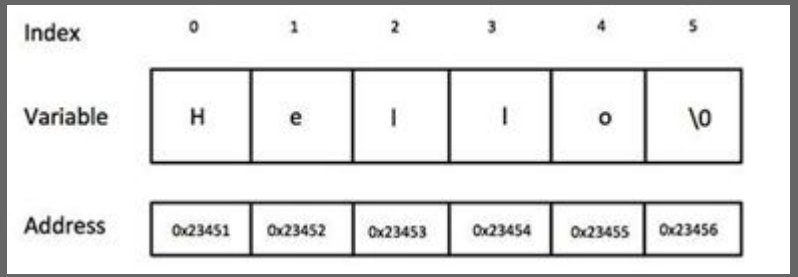
Strings in C



KOÇ
UNIVERSITY

C-Strings

- 1-D array of characters
- Terminated by **null** or **\0**
- Initializing a String
  - char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  - char greeting[] = "Hello";
  - char greeting[12] = "Hello";



KOÇ UNIVERSITY

String Functions in C

String functions	Description
strcat ( )	Concatenates str2 at the end of str1
strncat ( )	Appends a portion of string to another
strcpy ( )	Copies str2 into str1
strncpy ( )	Copies given number of characters of one string to another
strlen ( )	Gives the length of str1
strcmp ( )	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2
strcmpi ( )	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
strchr ( )	Returns pointer to first occurrence of char in str1
strrchr ( )	last occurrence of given character in a string is found
strstr ( )	Returns pointer to first occurrence of str2 in str1
strrstr ( )	Returns pointer to last occurrence of str2 in str1
strdup ( )	Duplicates the string
strlwr ( )	Converts string to lowercase
strupr ( )	Converts string to uppercase
strrev ( )	Reverses the given string
strset ( )	Sets all character in a string to given character
strnset ( )	It sets the portion of characters in a string to given character
strtok ( )	Tokenizing given string using delimiter

Using String functions

- Finding length of the str1

```
str1 = “Hello Comp201”;
len = strlen(str1);
printf("strlen(str1) :  %d\n", len );
//prints: strlen(str1) :  13
```

- Concatenating two strings

```
str1 = “Ahmed”;
str2 = “Student”;
strcat( str1, str2);
printf("strcat( str1, str2):   %s\n", str1 );
//prints: strcat( str1, str2):  AhmedStudent
```

Using String functions

- Converting str1 to Lowercase

```
str1 = “Hello Comp201”;
lwr = strlwr(str1);
printf("strlwr(str1) :  %s\n", lwr );
//prints: strlwr(str1) :  hello comp201
```

- Comparing two strings

```
str1 = “Ahmed”;
str2 = “ahmed”;
str3 = strcmpi( str1, str2);
printf("strcmpi( str1, str2):   %d\n", str3 );
//prints: strcmpi( str1, str2):  0
```

Using String functions

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

```
str1 = “world”;
str2 = “word”;
loc = strspn(str1,str2);
printf("loc:  %d\n", loc );
//prints: loc:  3
```

- Find str2 inside str1

```
str1 = “Impossible”;
str2 = “possible”;
substr = strstr( str1, str2);
printf(“substr :  %d\n", substr);
//prints: substr : possible
```

KOÇ UNIVERSITY

Strings In Memory

- Strings is a char array in the memory. We can change each character because we can change contents of array.
- There is a 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. We can set a char * equal to another value, because it is a reassign-able 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()

```
/*
 * We define a function countChars that counts the characters in the string str
 * returns the last index i
 */
int countChars(char str[])
{
        int i=0;

        while ( str[i]! = '\0' ){
                i++;
        }
        return i;
}
```

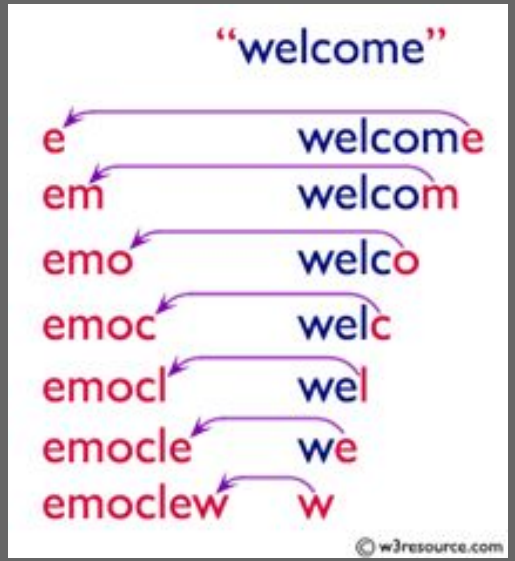
Arrays of Strings

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

- “argv” in main function parameters is an array of strings.
- Each memory location pointed by argv contains a string.
  void myFunction(char **stringArray) {
  void myFunction(char *stringArray[ ]) {
- These are equivalent and they are double pointers (a pointer containing memory location of an another pointer).

Print individual characters of string in reverse order

```
void main(){
   char str[100]; /* Declares a string of size 100 */
   int l,i;
   printf("Input the string : ");
   fgets(str, sizeof str, stdin);
   l=strlen(str);
   printf("The characters of the string in reverse are : \n");
   for(i=l ; i>=0 ; i--){
         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(“this is string one ”, “this is string two”) = “this is string one this is string two”

Note: To run exercises first run make then run the program with desired function: **./stringsO *functionName***

String Exercises

- **removeDup:** Remove duplicate characters from a string

Ex: removeDup(“silence is a source of great strength”) = “silenc aourfgth”

- **largestSmallest:** Find the largest and smallest (length) word in a string

Ex: largestSmallest (“It is a string with smallest and largest word”)
- The largest word is 'smallest'
- and the smallest word is 'a'

Note: To run exercises first run make then run the program with desired function: **./stringsO *functionName***

String Exercise

- **areAnagram:** return true if given two strings are anagram of each other

Ex: areAnagram(“earth”,”heart”) = true

Note: To run exercises first run make then run the program with desired function: **./stringsO *functionName***