C-Strings and Valgrind

COMP201 Lab Session
Fall 2020

KOÇ UNIVERSITY

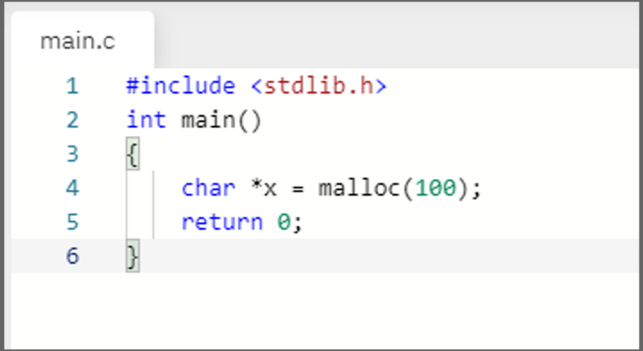
Valgrind

Valgrind is a programming tool used for:

- memory debugging
- memory leak detection
- profiling

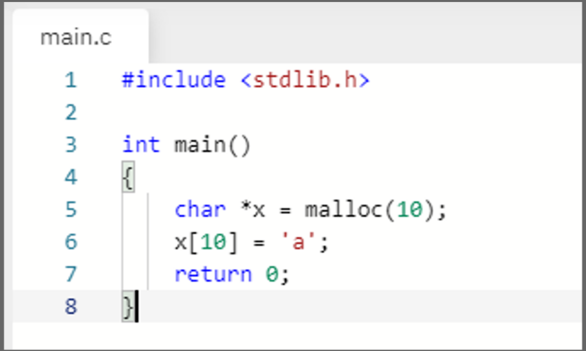
KOÇ UNIVERSITY

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)
```

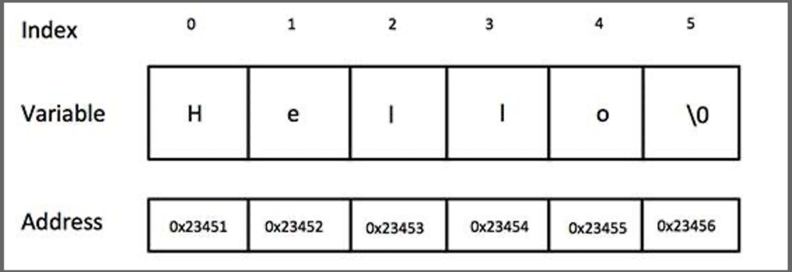
When Invalid pointer index is called

```
==9814==  Invalid write of size 1
==9814==    at 0x804841E: main (main.c:6)
```

KOÇ UNIVERSITY

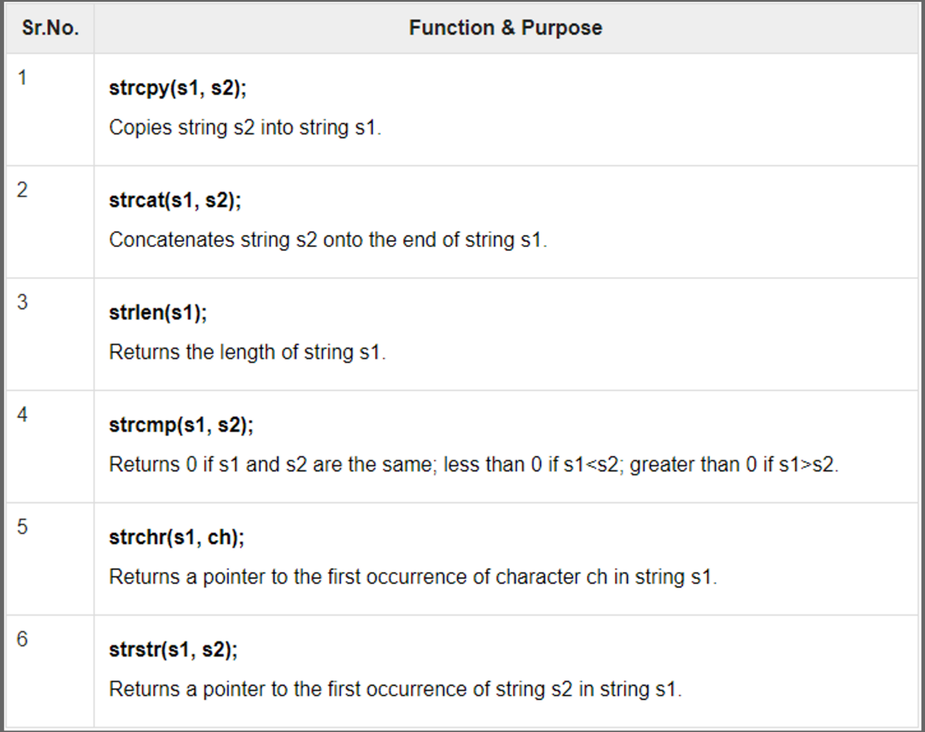
C-Strings

- 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";



Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

String Functions in C



Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

KOÇ UNIVERSITY

Using String functions

- Finding length of 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*

KOÇ UNIVERSITY

Strings In Memory

- If we create a string as a char[], we can modify its characters because its memory lives in our stack space.
- 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 *.
- If we create a new string with new characters as a char *, we cannot modify its characters because its memory lives in the data segment.
- We can set a char * equal to another value, because it is a reassign-able pointer.
- Adding an offset to a C string gives us a substring that many places past the first character.
- If we change characters in a string parameter, these changes will persist outside of the function.

Treating like an Array

- Find length without using strlen()

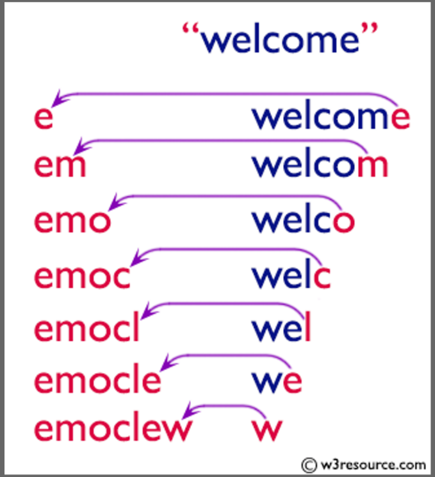
```
/*
 * We define a function countChars that counts the characters in the
string str
 * returns the last index i
 */
int countChars(char str[])
{
          int i=0;

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

KOÇ UNIVERSITY

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");
}
```


"welcome"
e welcome
em welcom
emo welco
emoc welc
emocl wel
emocle we
emoclew w
© w3resource.com