Valgrind

Valgrind is a programming tool used for:
- memory debugging
- memory leak detection
- profiling
Memory Allocated but Never Used

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

Finding Invalid Pointer Use With Valgrind

```
#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 0x804840E: main (main.c:6)
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";`
## String Functions in C

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Function &amp; Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>strcpy(s1, s2);</code></td>
</tr>
<tr>
<td></td>
<td>Copies string s2 into string s1.</td>
</tr>
<tr>
<td>2</td>
<td><code>strcat(s1, s2);</code></td>
</tr>
<tr>
<td></td>
<td>Concatenates string s2 onto the end of string s1.</td>
</tr>
<tr>
<td>3</td>
<td><code>strlen(s1);</code></td>
</tr>
<tr>
<td></td>
<td>Returns the length of string s1.</td>
</tr>
<tr>
<td>4</td>
<td><code>strncmp(s1, s2);</code></td>
</tr>
<tr>
<td></td>
<td>Returns 0 if s1 and s2 are the same; less than 0 if s1&lt;s2; greater than 0 if s1&gt;s2.</td>
</tr>
<tr>
<td>5</td>
<td><code>strchr(s1, ch);</code></td>
</tr>
<tr>
<td></td>
<td>Returns a pointer to the first occurrence of character ch in string s1.</td>
</tr>
<tr>
<td>6</td>
<td><code>strstr(s1, s2);</code></td>
</tr>
<tr>
<td></td>
<td>Returns a pointer to the first occurrence of string s2 in string s1.</td>
</tr>
</tbody>
</table>
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
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()

```c
/*
 * 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;
}
```
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");
}