**Assignment 3 Solutions - String Processing**

Question 1: Implement strlen()

#include <stdio.h>

// My version of strlen()

int my\_strlen(char \*str) {

int length = 0;

// Keep counting until we hit the null character

while (str[length] != '\0') {

length++;

}

return length;

}

int main() {

char my\_string[] = "Hello, world!";

int len;

len = my\_strlen(my\_string);

printf("The string is: \"%s\"\n", my\_string);

printf("The length of the string is: %d\n", len);

return 0;

}

Sample Output:

The string is: "Hello, world!"

The length of the string is: 13

Question 2: Implement strcpy()

#include <stdio.h>

// My version of strcpy()

void my\_strcpy(char \*destination, char \*source) {

int i = 0;

// Copy each character from source to destination

while (source[i] != '\0') {

destination[i] = source[i];

i++;

}

// IMPORTANT: Add the null terminator at the end

destination[i] = '\0';

}

int main() {

char source\_str[] = "This is a test.";

// Destination array must be big enough to hold the source string

char dest\_str[50];

my\_strcpy(dest\_str, source\_str);

printf("Source string: %s\n", source\_str);

printf("Copied string: %s\n", dest\_str);

return 0;

}

Sample Output:

Source string: This is a test.

Copied string: This is a test.

Question 3: Implement strcat()

#include <stdio.h>

// My version of strcat()

void my\_strcat(char \*destination, char \*source) {

int dest\_len = 0;

int i = 0;

// First, find the end of the destination string

while (destination[dest\_len] != '\0') {

dest\_len++;

}

// Now, copy the source string to the end of destination

while (source[i] != '\0') {

destination[dest\_len] = source[i];

dest\_len++;

i++;

}

// Add the null terminator

destination[dest\_len] = '\0';

}

int main() {

// Destination must be large enough for both strings!

char first\_part[50] = "Robert ";

char second\_part[] = "Brown";

printf("First part: %s\n", first\_part);

printf("Second part: %s\n", second\_part);

my\_strcat(first\_part, second\_part);

printf("Combined string: %s\n", first\_part);

return 0;

}

Sample Output:

First part: Robert

Second part: Brown

Combined string: Robert Brown

Question 4: Implement strcmp()

#include <stdio.h>

// My version of strcmp()

int my\_strcmp(char \*str1, char \*str2) {

int i = 0;

// Loop as long as the characters are the same and we haven't hit the end

while (str1[i] == str2[i] && str1[i] != '\0') {

i++;

}

// Return the difference of the ASCII values of the differing characters.

// This will be 0 if both are '\0' (strings are identical).

return str1[i] - str2[i];

}

int main() {

char s1[] = "apple";

char s2[] = "apply";

char s3[] = "apple";

int result1 = my\_strcmp(s1, s2);

int result2 = my\_strcmp(s1, s3);

printf("Comparing \"%s\" and \"%s\":\n", s1, s2);

if (result1 < 0) {

printf("Result is negative. \"%s\" comes before \"%s\".\n", s1, s2);

} else if (result1 > 0) {

printf("Result is positive. \"%s\" comes after \"%s\".\n", s1, s2);

} else {

printf("Result is 0. The strings are identical.\n");

}

printf("\nComparing \"%s\" and \"%s\":\n", s1, s3);

if (result2 == 0) {

printf("Result is 0. The strings are identical.\n");

}

return 0;

}

Sample Output:

Comparing "apple" and "apply":

Result is negative. "apple" comes before "apply".

Comparing "apple" and "apple":

Result is 0. The strings are identical.

Question 5: Limitations of a 2D Array of Characters

#include <stdio.h>

#include <string.h>

int main() {

// A 2D array: 3 rows, each can hold up to 20 characters.

// This allocates 3 \* 20 = 60 bytes of memory.

char names[3][20];

// Let's store some names

strcpy(names[0], "Robert"); // only needs 7 bytes (6 + '\0')

strcpy(names[1], "Tom"); // only needs 4 bytes (3 + '\0')

strcpy(names[2], "Eshaan"); // only needs 7 bytes (6 + '\0')

printf("--- Limitations of a 2D char array ---\n\n");

printf("We stored the name \"%s\".\n", names[1]);

printf("Its actual length is %zu characters.\n", strlen(names[1]));

printf("But in memory, it takes up all %d character spaces.\n", 20);

printf("This leads to a lot of wasted memory for short strings.\n\n");

printf("Another limitation is the fixed size.\n");

printf("We cannot store a name longer than 19 characters.\n");

return 0;

}

Sample Output:

--- Limitations of a 2D char array ---

We stored the name "Tom".

Its actual length is 3 characters.

But in memory, it takes up all 20 character spaces.

This leads to a lot of wasted memory for short strings.

Another limitation is the fixed size.

We cannot store a name longer than 19 characters.

Question 6: Demonstrate an Array of Pointers to Strings

#include <stdio.h>

int main() {

// An array of pointers. Each element points to a string literal.

// This is very memory efficient.

char \*names[] = {

"Ujjwal",

"Tom",

"Eshaan Singh",

"A much longer name to show flexibility"

};

int num\_names = 4;

int i;

printf("--- Using an Array of Pointers to Strings ---\n\n");

printf("This method is better because there is no wasted memory.\n");

printf("Each string takes up only the space it needs.\n\n");

printf("Here are the stored names:\n");

for (i = 0; i < num\_names; i++) {

printf("- %s\n", names[i]);

}

return 0;

}

Sample Output:

--- Using an Array of Pointers to Strings ---

This method is better because there is no wasted memory.

Each string takes up only the space it needs.

Here are the stored names:

- Robert

- Tom

- Eshaan

- A much longer name to show flexibility