Lab Assignment 3
UEE410
Naysa Kukreja
102304049
3D13

1) Write a program to implement strlen() function.

a)#include <iostream>

```
// Manual implementation of strlen()
size_t my_strlen(const char* str) {
    size_t length = 0;
    while (str[length] != '\0') {
        length++;
    }
    return length;
}

int main() {
    const char* str = "C++ is great!";

    // Using the manually implemented function
    size_t length = my_strlen(str);

    std::cout << "The string is: \"" << str << "\"" << std::endl;
    std::cout << "The length of the string is: " << length << std::endl;

    return 0;
}
```

Output

```
The string is: "C++ is great!"
The length of the string is: 13


=== Code Execution Successful ===
```

b)

```
#include <iostream>
#include <cstring>

int main() {
    const char* str = "This is a C++ string.";

    // Using the in-built strlen() function from the <cstring> header
    size_t length = strlen(str);

    std::cout << "The string is: \"" << str << "\"" << std::endl;
    std::cout << "The length of the string is: " << length << std::endl;

    return 0;
}
```

Output

```
The string is: "This is a C++ string."
The length of the string is: 21


=== Code Execution Successful ===
```

2) Write a program to implement strcpy() function.

```
#include <iostream>

char* my_strcpy(char* destination, const char* source) {
    // Check for null pointers
    if (destination == nullptr || source == nullptr) {
        return nullptr;
    }

    char* original_destination = destination;

    // Copy characters from source to destination
    while (*source != '\0') {
        *destination = *source;
        destination++;
        source++;
    }

    // Add the null terminator to the destination string
    *destination = '\0';

    return original_destination;
}

int main() {
    char source[] = "Hello, World!";
    char destination[20]; // Ensure the destination has enough space

    my_strcpy(destination, source);

    std::cout << "Source string: " << source << std::endl;
    std::cout << "Destination string: " << destination << std::endl;

    return 0;
}
```

Output

```
Source string: Hello, World!
Destination string: Hello, World!


=== Code Execution Successful ===
```

3) Write a program to implement strcat() function.

```
#include <iostream>
#include <string>

// Define a struct named 'Student'
struct Student {
    std::string name;
    int roll_number;
    double gpa;
};

int main() {
    // Declare a variable of type 'Student'
    Student s1;

    // Access and assign values to the members
    s1.name = "Naysa Kukreja";
    s1.roll_number = 4049;
    s1.gpa = 10;

    // Access and print the member values
    std::cout << "Student Name: " << s1.name << std::endl;
    std::cout << "Roll Number: " << s1.roll_number << std::endl;
    std::cout << "GPA: " << s1.gpa << std::endl;

    return 0;
}
```

Output

```
Student Name: Naysa Kukreja
Roll Number: 4049
GPA: 10


=== Code Execution Successful ===
```

4) Write a program to implement strcmp() function.

```
#include <iostream>

int my_strcmp(const char* str1, const char* str2) {
    // Iterate through both strings until a mismatch or a null terminator is found
    while (*str1 != '\0' && *str2 != '\0' && *str1 == *str2) {
        str1++;
        str2++;
    }

    // Return the difference of the characters at the point of mismatch or the null terminators
    return static_cast<int>(*str1) - static_cast<int>(*str2);
}

int main() {
    const char* s1 = "apple";
    const char* s2 = "apple";
    const char* s3 = "apply";
    const char* s4 = "apricot";

    std::cout << "Comparing \"" << s1 << "\" and \"" << s2 << "\": " << my_strcmp(s1, s2) <<
std::endl;
    std::cout << "Comparing \"" << s1 << "\" and \"" << s3 << "\": " << my_strcmp(s1, s3) <<
std::endl;
```

```
    std::cout << "Comparing \"" << s1 << "\" and \"" << s4 << "\": " << my_strcmp(s1, s4) << std::endl;
    std::cout << "Comparing \"" << s3 << "\" and \"" << s1 << "\": " << my_strcmp(s3, s1) << std::endl;

    return 0;
}
```

Output

```
Comparing "apple" and "apple": 0
Comparing "apple" and "apply": -20
Comparing "apple" and "apricot": -2
Comparing "apply" and "apple": 20


=== Code Execution Successful ===
```

5) WAP to demonstrate limitations of Two-Dimensional Array of Characters.

```
#include <iostream>
#include <cstring> // For strcpy

int main() {
    // Declaring a 2D array with fixed dimensions (3 strings, max 10 chars each)
    char colors[3][10];

    // Limitation 1: Wasted memory due to fixed row size.
    // "red" is 3 chars + '\0' = 4 bytes, but 10 bytes are allocated.
    // "blue" is 4 chars + '\0' = 5 bytes, but 10 bytes are allocated.
    // "green" is 5 chars + '\0' = 6 bytes, but 10 bytes are allocated.
    strcpy(colors[0], "red");
    strcpy(colors[1], "blue");
    strcpy(colors[2], "green");

    std::cout << "Demonstrating Wasted Memory:" << std::endl;
    for (int i = 0; i < 3; ++i) {
```

```
        std::cout << "String: \"" << colors[i] << "\" occupies " << sizeof(colors[i]) << " bytes." <<
std::endl;
    }
    std::cout << "Total allocated memory: " << sizeof(colors) << " bytes." << std::endl;
    std::cout << "Actual data size (approx): " << (3 + 1) + (4 + 1) + (5 + 1) << " bytes." <<
std::endl;
    std::cout << std::endl;

    // Limitation 2: Lack of flexibility (e.g., cannot easily add a longer string).
    char new_color[] = "yellow_is_a_long_color";

    std::cout << "Attempting to store a string that is too long." << std::endl;
    std::cout << "The string \"" << new_color << "\" is " << strlen(new_color) << " characters
long." << std::endl;
    std::cout << "The allocated space is only 10 bytes. This would cause a buffer overflow." <<
std::endl;

    // Uncommenting the line below would cause a buffer overflow and undefined behavior.
    // strcpy(colors[0], new_color);

    return 0;
}
```

Output

```
Demonstrating Wasted Memory:
String: "red" occupies 10 bytes.
String: "blue" occupies 10 bytes.
String: "green" occupies 10 bytes.
Total allocated memory: 30 bytes.
Actual data size (approx): 15 bytes.

Attempting to store a string that is too long.
The string "yellow_is_a_long_color" is 22 characters long.
The allocated space is only 10 bytes. This would cause a buffer overflow.


=== Code Execution Successful ===
```

6) WAP to demonstrate an array of Pointers to Strings.

```
#include <iostream>
#include <cstring> // This is the corrected line

int main() {
   const char* months[] = {
      "January", "February", "March", "April", "May", "June",
      "July", "August", "September", "October", "November", "December"
   };

   int num_months = sizeof(months) / sizeof(months[0]);

   std::cout << "Demonstrating an Array of Pointers to Strings:" << std::endl;
   std::cout << "----------------------------------------------" << std::endl;

   for (int i = 0; i < num_months; ++i) {
      std::cout << "Month " << i + 1 << ": " << months[i] << std::endl;
   }

   std::cout << "\nAccessing the 8th month: " << months[7] << std::endl;

   // These lines now work because <cstring> is included
   std::cout << "\nMemory occupied by 'January': " << strlen(months[0]) + 1 << " bytes" <<
std::endl;
   std::cout << "Memory occupied by 'September': " << strlen(months[8]) + 1 << " bytes" <<
std::endl;

   return 0;
}
```

Output

```
Demonstrating an Array of Pointers to Strings:
----------------------------------------------
Month 1: January
Month 2: February
Month 3: March
Month 4: April
Month 5: May
Month 6: June
Month 7: July
Month 8: August
Month 9: September
Month 10: October
Month 11: November
Month 12: December

Accessing the 8th month: August

Memory occupied by 'January': 8 bytes
Memory occupied by 'September': 10 bytes


=== Code Execution Successful ===
```