

String-Concepts

1. String Length Calculation

- **Requirement:** Write a program that takes a string input and calculates its length using `strlen()`. The program should handle empty strings and output appropriate messages.
- **Input:** A string from the user.
- **Output:** Length of the string.

```
#include<stdio.h>

#include<string.h>

int main(){

    char string[20];

    printf("Enter a string\n");

    scanf("%s",string);

    printf("Length of string=%ld\n",strlen(string));

    return 0;

}
```

2. String Copy

- **Requirement:** Implement a program that copies one string to another using `strcpy()`. The program should validate if the source string fits into the destination buffer.
- **Input:** Two strings from the user (source and destination).
- **Output:** The copied string.

```
#include<stdio.h>

#include<string.h>

int main(){

    char source[20],destination[20];

    printf("Enter source string\n");

    scanf("%s",source);

    printf("Enter destination string\n");

    scanf("%s",destination);
```

```

if(strlen(source) < sizeof(destination)){

    strcpy(destination,source);

    printf("Destination=%s\n",destination);

}

else

printf("Error: Source string is too large for the destination buffer.\n");

return 0;

}

```

3. String Concatenation

- **Requirement:** Create a program that concatenates two strings using strcat(). Ensure the destination string has enough space to hold the result.
- **Input:** Two strings from the user.
- **Output:** The concatenated string.

```

#include<stdio.h>

#include<string.h>

int main(){

    char source[20],destination[20];

    printf("Enter source string\n");

    scanf("%s",source);

    printf("Enter destination string\n");

    scanf("%s",destination);

    if(sizeof(destination) > strlen(source)+strlen(destination)){

        strcat(destination,source);

        printf("Destination=%s\n",destination);

    }

    else

printf("Error: Source string is too large for the destination buffer.\n");

```

```
return 0;
```

```
}
```

4. String Comparison

- **Requirement:** Develop a program that compares two strings using strcmp(). It should indicate if they are equal or which one is greater.
- **Input:** Two strings from the user.
- **Output:** Comparison result.

```
#include<stdio.h>
```

```
#include<string.h>
```

```
int main(){
```

```
    char source[20],destination[20];
```

```
    printf("Enter source string\n");
```

```
    scanf("%s",source);
```

```
    printf("Enter destination string\n");
```

```
    scanf("%s",destination);
```

```
    if(strcmp(source,destination)){
```

```
        printf("Strings are not equal.\n");
```

```
        if(strlen(source)>strlen(destination))
```

```
            printf("%s is greater\n",source);
```

```
        else
```

```
            printf("%s is greater.\n",destination);
```

```
    }
```

```
    else
```

```
        printf("Soure and destination are equal\n");
```

```
    return 0;
```

```
}
```

5. Convert to Uppercase

- **Requirement:** Write a program that converts all characters in a string to uppercase using `strupr()`.
- **Input:** A string from the user.
- **Output:** The uppercase version of the string.

```
#include<stdio.h>

#include<string.h>

int main(){

    char source[20];

    printf("Enter string\n");

    scanf("%s",source);

    for (int i = 0; source[i] != '\0'; i++) {

        source[i] = toupper(source[i]); // Convert each character to uppercase

    }

    printf("Upper cased :%s\n",source);

    return 0;

}
```

6. Convert to Lowercase

- **Requirement:** Implement a program that converts all characters in a string to lowercase using `strlwr()`.
- **Input:** A string from the user.
- **Output:** The lowercase version of the string.

//strlwr not supported by the compiler

```
#include<stdio.h>

#include<string.h>

int main(){

    char source[20];

    printf("Enter string\n");
```

```

scanf("%s",source);

for (int i = 0; source[i] != '\0'; i++) {

    source[i] = tolower(source[i]); // Convert each character to lowercase

}

printf("Upper cased :%s\n",source);


return 0;

}

```

7. Substring Search

- **Requirement:** Create a program that searches for a substring within a given string using strstr() and returns its starting index or an appropriate message if not found.
- **Input:** A main string and a substring from the user.
- **Output:** Starting index or not found message.

```

#include <stdio.h>

#include <string.h>

int main() {

    char mainString[100], subString[50];


    // Get user input for the main string and the substring

    printf("Enter the main string:\n");

    fgets(mainString, sizeof(mainString), stdin);


    // Remove the newline character from mainString if present

    size_t len = strlen(mainString);

    if (len > 0 && mainString[len - 1] == '\n') {

        mainString[len - 1] = '\0';

    }
}

```

```

printf("Enter the substring to search for:\n");

fgets(subString, sizeof(subString), stdin);

// Remove the newline character from subString if present

len = strlen(subString);

if (len > 0 && subString[len - 1] == '\n') {

    subString[len - 1] = '\0';

}


// Use strstr() to find the substring in the main string

char *position = strstr(mainString, subString);

if (position != NULL) {

    // Calculate the index of the starting position of the substring

    int index = position - mainString;

    printf("Substring found at index: %d\n", index);

} else {

    printf("Substring not found\n");

}


return 0;

}

```

8. Character Search

- **Requirement:** Write a program that finds the first occurrence of a character in a string using strchr() and returns its index or indicates if not found.
- **Input:** A string and a character from the user.

- **Output:** Index of first occurrence or not found message.

```
#include<stdio.h>

#include<string.h>

int main(){

    char string[50];

    char ch;

    printf("Enter string:\n");

    scanf("%s",string);

    printf("Enter character:\n");

    scanf(" %c", &ch);

    char *position=strchr(string,ch);

    if (position != NULL) {

        // Calculate the index of the character

        int index = position - string;

        printf("Character '%c' found at index: %d\n", ch, index);

    } else {

        printf("Character '%c' not found in the string.\n", ch);

    }

}
```

9. String Reversal

- **Requirement:** Implement a function that reverses a given string in place without using additional memory, leveraging strlen() for length determination.
- **Input:** A string from the user.
- **Output:** The reversed string.

```
#include <stdio.h>

#include <string.h>
```

```
void reverseString(char *str);
```

```
int main() {  
    char str[100];  
    printf("Enter a string:\n");  
    scanf("%s",str);  
    // Reverse the string  
    reverseString(str);  
    // Output the reversed string  
    printf("Reversed string: %s\n", str);  
  
    return 0;  
}
```

```
void reverseString(char *str) {  
    int left = 0;  
    int right = strlen(str) - 1;  
  
    // Swap characters from the start and end moving towards the center  
    while (left < right) {  
        char temp = str[left];  
        str[left] = str[right];  
        str[right] = temp;  
        left++;  
        right--;  
    }
```



```
}  
  
}
```

10. String Tokenization

- **Requirement:** Create a program that tokenizes an input string into words using strtok() and counts how many tokens were found.
- **Input:** A sentence from the user.
- **Output:** Number of words (tokens).

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main() {
```

```
    char str[200];
```

```
    char *token;
```

```
    int count = 0;
```

```
    // Get the input string from the user
```

```
    printf("Enter a sentence:\n");
```

```
    fgets(str, sizeof(str), stdin);
```

```
    // Tokenize the string using space as the delimiter
```

```
    token = strtok(str, " ");
```

```
    while (token != NULL) {
```

```
        printf("Token: %s\n", token); // Print each token
```

```
        count++;
```

```
        token = strtok(NULL, " "); // Get the next token
```

```
    }
```

```

// Output the total number of tokens

printf("Total number of words: %d\n", count);

return 0;

}

```

11. String Duplication

- **Requirement:** Write a function that duplicates an input string (allocating new memory) using strdup() and displays both original and duplicated strings.
- **Input:** A string from the user.
- **Output:** Original and duplicated strings.

```

#include <stdio.h>

#include <string.h>

int main() {

    char str[200];

    printf("Enter a string:\n");

    scanf("%s",str);

    // Duplicate the string using strdup()

    char *duplicatedStr = strdup(str);

    printf("Original String: %s\n", str);

    printf("Duplicated String: %s\n", duplicatedStr);

    return 0;

}

```

12. Case-Insensitive Comparison

- **Requirement:** Develop a program to compare two strings without case sensitivity using `strcasecmp()` and report equality or differences.
- **Input:** Two strings from the user.
- **Output:** Comparison result.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main() {
```

```
    char str1[100], str2[100];
```

```
    // Input two strings from the user
```

```
    printf("Enter the first string: ");
```

```
    scanf("%s",str1);
```

```
    printf("Enter the second string: ");
```

```
    scanf("%s",str2);
```

```
    // Perform case-insensitive comparison using strcasecmp()
```

```
    int result = strcasecmp(str1, str2);
```

```
    if (result == 0) {
```

```
        printf("The strings are equal (case-insensitive).\n");
```

```
    } else {
```

```
        printf("The strings are different.\n");
```

```
    }
```

```
    return 0;
```

```
}
```

13. String Trimming

- **Requirement:** Implement functionality to trim leading and trailing whitespace from a given string, utilizing pointer arithmetic with `strlen()`.
- **Input:** A string with extra spaces from the user.
- **Output:** Trimmed version of the string.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <ctype.h>
```

```
void trimWhitespace(char *str);
```

```
int main() {
```

```
    char input[200];
```

```
    printf("Enter a string with extra spaces: ");
```

```
    scanf("%s",input);
```

```
    // Trim the whitespace
```

```
    trimWhitespace(input);
```

```
    // Output the trimmed string
```

```
    printf("Trimmed string: '%s'\n", input);
```

```
    return 0;
```

```
}
```

```
void trimWhitespace(char *str) {
```

```

char *start = str;

char *end;

// Find the first non-whitespace character
while (*start && isspace((unsigned char)*start)) {

    start++;

}

// If the string is entirely spaces, set it to an empty string
if (*start == '\0') {

    str[0] = '\0';

    return;

}

// Find the last non-whitespace character

end = start + strlen(start) - 1;

while (end > start && isspace((unsigned char)*end)) {

    end--;

}

// Null-terminate the trimmed string

*(end + 1) = '\0';

// Shift the trimmed string to the beginning of the array

memmove(str, start, end - start + 2);

```

```
}
```

14. Find Last Occurrence of Character

- **Requirement:** Write a program that finds the last occurrence of a character in a string using manual iteration instead of library functions, returning its index.
- **Input:** A string and a character from the user.
- **Output:** Index of last occurrence or not found message.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int findLastOccurrence(const char *str, char ch);
```

```
int main() {
```

```
    char input[200];
```

```
    char ch;
```

```
    // Input the string from the user
```

```
    printf("Enter a string: ");
```

```
    scanf("%s",input);
```

```
    // Input the character to search
```

```
    printf("Enter a character to find: ");
```

```
    scanf(" %c", &ch);
```

```
    // Find the last occurrence of the character
```

```
    int index = findLastOccurrence(input, ch);
```

```
    if (index != -1) {
```

```

        printf("Last occurrence of '%c' is at index %d.\n", ch, index);
    } else {
        printf("Character '%c' not found in the string.\n", ch);
    }

    return 0;
}

int findLastOccurrence(const char *str, char ch) {
    int lastIndex = -1; // Initialize to -1 (not found)

    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            lastIndex = i; // Update the index whenever the character is found
        }
    }

    return lastIndex;
}

```

15. Count Vowels in String

- **Requirement:** Create a program that counts how many vowels are present in an input string by iterating through each character.
- **Input:** A string from the user.
- **Output:** Count of vowels.

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
int countVowels(const char *str);
```

```
int main() {  
  
    char input[200];  
  
    printf("Enter a string: ");  
    scanf("%s",input);  
  
    int vowelCount = countVowels(input);  
  
    printf("Number of vowels in the string: %d\n", vowelCount);  
  
    return 0;  
}  
  
int countVowels(const char *str) {  
  
    int count = 0;  
  
    // Iterate through the string  
    for (int i = 0; str[i] != '\0'; i++) {  
  
        // Convert character to lowercase for case-insensitivity  
        char ch = tolower(str[i]);  
  
        // Check if the character is a vowel  
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {  
            count++;  
        }  
    }  
}
```



```
        return count;
    }
}
```

16. Count Specific Characters

- **Requirement:** Implement functionality to count how many times a specific character appears in an input string, allowing for case sensitivity options.
- **Input:** A string and a character from the user.
- **Output:** Count of occurrences.

```
#include <stdio.h>

#include <ctype.h>

int countCharacter(const char *str, char ch, int caseSensitive);

int main() {

    char input[200];

    char character;

    int caseSensitive;

    printf("Enter a string: ");

    scanf("%s",input);

    printf("Enter a character to count: ");

    scanf(" %c", &character);

    printf("Case sensitive? (1 for Yes, 0 for No): ");

    scanf("%d", &caseSensitive);
```

```

// Count the occurrences of the character

int count = countCharacter(input, character, caseSensitive);


// Output the result

printf("The character '%c' appears %d time(s) in the string.\n", character,
count);


return 0;
}


int countCharacter(const char *str, char ch, int caseSensitive) {

    int count = 0;


    // Iterate through the string
    for (int i = 0; str[i] != '\0'; i++) {

        // Check for case insensitivity
        if (!caseSensitive) {

            if (tolower(str[i]) == tolower(ch)) {

                count++;

            }

        } else {

            if (str[i] == ch) {

                count++;

            }

        }

    }

}

```

```
        return count;
    }
}
```

17. Remove All Occurrences of Character

- **Requirement:** Write a function that removes all occurrences of a specified character from an input string, modifying it in place.
- **Input:** A string and a character to remove from it.
- **Output:** Modified string without specified characters.

```
#include <stdio.h>

void removeCharacter(char *str, char ch);

int main() {

    char input[200];

    char character;

    printf("Enter a string: ");

    scanf("%s",input);

    printf("Enter a character to remove: ");

    scanf(" %c", &character);

    // Remove the character

    removeCharacter(input, character);

    // Output the modified string

    printf("Modified string: %s", input);

    return 0;

}
```

```
void removeCharacter(char *str, char ch) {

    int i = 0, j = 0;
```

```

// Loop through the string

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

    // If the current character is not the one to remove, keep it

    if (str[i] != ch) {

        str[j] = str[i];

        j++;

    }

    i++;

}

// Null-terminate the modified string

str[j] = '\0';

}

```

18. Check for Palindrome

- **Requirement:** Develop an algorithm to check if an input string is a palindrome by comparing characters from both ends towards the center, ignoring case and spaces.
- **Input:** A potential palindrome from the user.
- **Output:** Whether it is or isn't a palindrome.

```

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int isPalindrome(char *str);

int main() {

    char input[200];

    printf("Enter a string: ");

    scanf("%s",input);

```

```

    if (isPalindrome(input)) {

        printf("The string is a palindrome.\n");

    } else {

        printf("The string is not a palindrome.\n");

    }

    return 0;

}

int isPalindrome(char *str) {

    int start = 0, end = strlen(str) - 1;

    // Loop through the string, comparing characters from both ends

    while (start < end) {

        // Skip non-alphanumeric characters

        if (!isalnum(str[start])) {

            start++;

        } else if (!isalnum(str[end])) {

            end--;

        } else {

            // Compare characters, ignoring case

            if (tolower(str[start]) != tolower(str[end])) {

                return 0; // Not a palindrome

            }

            start++;

            end--;

        }

    }

```

```

    }

    return 1; // It is a palindrome
}

```

19. Extract Substring

- **Requirement:** Create functionality to extract a substring based on specified start index and length parameters, ensuring valid indices are provided by users.
- **Input:** A main string, start index, and length from the user.
- **Output:** Extracted substring or error message for invalid indices.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void extractSubstring(char *mainString, int startIndex, int length, char
*result);
```

```
int main() {
```

```
    char mainString[200], result[200];
```

```
    int startIndex, length;
```

```
    printf("Enter the main string: ");
```

```
    scanf("%s",mainString);
```

```
    // Remove newline character if present
```

```
    mainString[strcspn(mainString, "\n")] = '\0';
```

```
    // Input start index and length
```

```
    printf("Enter start index: ");
```

```
    scanf("%d", &startIndex);
```

```

printf("Enter length: ");

scanf("%d", &length);


// Call the extractSubstring function

extractSubstring(mainString, startIndex, length, result);


// Output the result

printf("Extracted Substring: %s\n", result);


return 0;
}

void extractSubstring(char *mainString, int startIndex, int length, char *result)
{

    int mainLength = strlen(mainString);

    // Check if the start index and length are valid

    if (startIndex < 0 || startIndex >= mainLength || length < 0 || (startIndex +
length) > mainLength) {

        printf("Error: Invalid indices.\n");

        return;

    }

    // Extract the substring

    for (int i = 0; i < length; i++) {

        result[i] = mainString[startIndex + i];

    }


    // Null terminate the result string

```

```
    result[length] = '\0';  
}
```

20. Sort Characters in String

- **Requirement:** Implement functionality to sort characters in an input string alphabetically, demonstrating usage of nested loops for comparison without library sorting functions.
- **Input:** A string from the user.
- **Output:** Sorted version of the characters in the string.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void sortString(char *str);
```

```
int main() {
```

```
    char str[200];
```

```
    printf("Enter a string: ");
```

```
    scanf("%s",str);
```

```
    // Call the sortString function to sort the characters
```

```
    sortString(str);
```

```
    // Output the sorted string
```

```
    printf("Sorted string: %s\n", str);
```

```
    return 0;
```

```
}
```

```
void sortString(char *str) {
```

```
    int length = strlen(str);
```



```

// Nested loop to compare and swap characters

for (int i = 0; i < length - 1; i++) {

    for (int j = i + 1; j < length; j++) {

        if (str[i] > str[j]) {

            // Swap characters if they are out of order

            char temp = str[i];

            str[i] = str[j];

            str[j] = temp;

        }

    }

}

```

21. Count Words in String

- **Requirement:** Write code to count how many words are present in an input sentence by identifying spaces as delimiters, utilizing strtok().
- **Input:** A sentence from the user.
- **Output:** Number of words counted.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int countWords(const char *sentence);
```

```
int main() {
```

```
    char sentence[200];
```

```
    printf("Enter a sentence: ");
```

```

fgets(sentence, sizeof(sentence), stdin);

// Count the words and display the result

int wordCount = countWords(sentence);

printf("Number of words: %d\n", wordCount);

return 0;
}

int countWords(const char *sentence) {

    char temp[200];

    strcpy(temp, sentence); // Make a copy of the sentence to avoid modifying
the original

    char *token;

    int count = 0;

    // Tokenize the sentence based on spaces and punctuation

    token = strtok(temp, " \t\n,!.?:;"); // spaces and punctuation are delimiters

    while (token != NULL) {

        count++;

        token = strtok(NULL, " \t\n,!.?:;"); // Continue tokenizing the string

    }

    return count;
}

```

```
}
```

22. Remove Duplicates from String

- **Requirement:** Develop an algorithm to remove duplicate characters while maintaining their first occurrence order in an input string.
- **Input:** A string with potential duplicate characters.
- **Output:** Modified version of the original without duplicates.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void removeDuplicates(char *str);
```

```
int main() {
```

```
    char str[200];
```

```
    printf("Enter a string: ");
```

```
    scanf("%s",str);
```

```
    removeDuplicates(str);
```

```
    // Output the modified string without duplicates
```

```
    printf("String after removing duplicates: %s\n", str);
```

```
    return 0;
```

```
}
```

```
void removeDuplicates(char *str) {
```

```
    int length = strlen(str);
```

```

// Iterate over the string
for (int i = 0; i < length; i++) {

    // If the character is not duplicated, move forward
    if (str[i] != '\0') {

        // Check all subsequent characters
        for (int j = i + 1; j < length; j++) {

            // If a duplicate is found, replace it with null character
            if (str[i] == str[j]) {

                str[j] = '\0';

            }

        }

    }

}

// Compact the string to remove 'holes' (i.e., null characters)
int writeIndex = 0;

for (int i = 0; i < length; i++) {

    if (str[i] != '\0') {

        str[writeIndex++] = str[i];

    }

}

str[writeIndex] = '\0'; // Null-terminate the modified string
}

```

23. Find First Non-Repeating Character

- **Requirement:** Create functionality to find the first non-repeating character in an input string, demonstrating effective use of arrays for counting occurrences.

- **Input:** A sample input from the user.
- **Output:** The first non-repeating character or indication if all are repeating.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
char findFirstNonRepeatingChar(const char *str);
```

```
int main() {
```

```
    char str[200];
```

```
    printf("Enter a string: ");
```

```
    scanf("%s",str);
```

```
    char result = findFirstNonRepeatingChar(str);
```

```
    if (result != '\0') {
```

```
        printf("First non-repeating character: %c\n", result);
```

```
    } else {
```

```
        printf("All characters are repeating.\n");
```

```
    }
```

```
    return 0;
```

```
}
```

```
char findFirstNonRepeatingChar(const char *str) {
```

```
    int count[256] = {0}; // Array to store the frequency of characters
```

```
    // Count the occurrences of each character
```

```

for (int i = 0; str[i] != '\0'; i++) {

    count[(unsigned char)str[i]]++;

}

// Find the first character with count 1 (non-repeating)

for (int i = 0; str[i] != '\0'; i++) {

    if (count[(unsigned char)str[i]] == 1) {

        return str[i]; // Return the first non-repeating character

    }

}

return '\0'; // Return null character if all are repeating

}

```

24. Convert String to Integer

- **Requirement:** Implement functionality to convert numeric strings into integer values without using standard conversion functions like atoi(), handling invalid inputs gracefully.
- **Input:** A numeric string.
- **Output:** Converted integer value or error message.

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
int convertToInteger(const char *str);
```

```
int main() {
```

```
    char str[100];
```

```
    printf("Enter a numeric string: ");
```

```
scanf("%s",str);

// Convert the string to an integer

int result = convertToInteger(str);

// Check if conversion was successful

if (result != -1) {

    printf("Converted integer: %d\n", result);

} else {

    printf("Invalid input. Please enter a valid numeric string.\n");

}

return 0;

}

int convertToInteger(const char *str) {

    int num = 0;

    int i = 0;

    // Check for empty string

    if (str[i] == '\0') {

        return -1; // Return -1 to indicate error (empty string)

    }

    // Handle possible negative sign
```

```

int sign = 1;

if (str[i] == '-') {

    sign = -1; // Negative number

    i++; // Move to the next character

}

// Convert each character to integer
for (; str[i] != '\0'; i++) {

    // Check if the character is a digit

    if (isdigit(str[i])) {

        num = num * 10 + (str[i] - '0');

    } else {

        return -1; // Return -1 if a non-digit is encountered

    }

}

return sign * num; // Return the final result with sign

}

```

25. Check Anagram Status Between Two Strings

- **Requirement:** Write code to check if two strings are anagrams by sorting their characters and comparing them.
- **Input:** Two strings.
- **Output:** Whether they are anagrams.

```

#include <stdio.h>

#include <string.h>

#include <ctype.h>

```

```

void sortString(char *str);

```



```

int areAnagrams(char *str1, char *str2);

int main() {

    char str1[100], str2[100];

    // Input two strings

    printf("Enter the first string: ");

    fgets(str1, sizeof(str1), stdin);

    str1[strcspn(str1, "\n")] = '\0'; // Remove newline character if present


    printf("Enter the second string: ");

    fgets(str2, sizeof(str2), stdin);

    str2[strcspn(str2, "\n")] = '\0'; // Remove newline character if present


    // Check if they are anagrams

    if (areAnagrams(str1, str2)) {

        printf("The strings are anagrams.\n");

    } else {

        printf("The strings are not anagrams.\n");

    }


    return 0;

}

void sortString(char *str) {

```

```

int len = strlen(str);

for (int i = 0; i < len - 1; i++) {

    for (int j = i + 1; j < len; j++) {

        if (str[i] > str[j]) {

            // Swap characters if they are in the wrong order

            char temp = str[i];

            str[i] = str[j];

            str[j] = temp;

        }

    }

}

}

int areAnagrams(char *str1, char *str2) {

    // If lengths of both strings are different, they can't be anagrams

    if (strlen(str1) != strlen(str2)) {

        return 0;

    }

    // Sort both strings

    sortString(str1);

    sortString(str2);

    // Compare the sorted strings

    return strcmp(str1, str2) == 0; // Return 1 if equal, 0 if not

```

```
}
```

26. Merge Two Strings Alternately

- **Requirement:** Create functionality to merge two strings alternately into one while handling cases where strings may be of different lengths.
- **Input:** Two strings.
- **Output:** Merged alternating characters.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void mergeStringsAlternately(char *str1, char *str2, char *result);
```

```
int main() {
```

```
    char str1[100], str2[100], result[200];
```

```
    printf("Enter the first string: ");
```

```
    scanf("%s",str1);
```

```
    printf("Enter the second string: ");
```

```
    scanf("%s",str2);
```

```
    // Merge the strings alternately
```

```
    mergeStringsAlternately(str1, str2, result);
```

```
    // Output the merged string
```

```
    printf("Merged string: %s\n", result);
```

```
    return 0;
```

```
}
```

```

void mergeStringsAlternately(char *str1, char *str2, char *result) {

    int i = 0, j = 0, k = 0;

    // Merge characters alternately from both strings
    while (str1[i] != '\0' && str2[j] != '\0') {

        result[k++] = str1[i++];

        result[k++] = str2[j++];

    }

    // If there are remaining characters in str1, append them
    while (str1[i] != '\0') {

        result[k++] = str1[i++];

    }

    // If there are remaining characters in str2, append them
    while (str2[j] != '\0') {

        result[k++] = str2[j++];

    }

    // Null-terminate the result string
    result[k] = '\0';

}

```

27. Count Consonants in String

- **Requirement:** Develop code to count consonants while ignoring vowels and whitespace characters.

- **Input:** Any input text.
- **Output:** Count of consonants.

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
int countConsonants(const char *str);
```

```
int main() {
```

```
    char str[100];
```

```
    // Input a string from the user
```

```
    printf("Enter a string: ");
```

```
    scanf("%s",str);
```

```
    // Count the consonants
```

```
    int consonantCount = countConsonants(str);
```

```
    // Output the count of consonants
```

```
    printf("The number of consonants in the string is: %d\n", consonantCount);
```

```
    return 0;
```

```
}
```

```
int countConsonants(const char *str) {
```

```
    int count = 0;
```

```

while (*str != '\0') {

    // Convert the character to lowercase for case-insensitive comparison

    char ch = tolower(*str);

    // Check if the character is a letter and not a vowel

    if ((ch >= 'a' && ch <= 'z') && !(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch
== 'u')) {

        count++;

    }

    // Move to the next character

    str++;

}

return count;

}

```

28. Replace Substring with Another String

- **Requirement:** Write functionality to replace all occurrences of one substring with another within a given main string.
- **Input:** Main text, target substring, replacement substring.
- **Output:** Modified main text after replacements.

29. Count Occurrences of Substring

- **Requirement:** Create code that counts how many times one substring appears within another larger main text without overlapping occurrences.
- **Input:** Main text and target substring.
- **Output:** Count of occurrences.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int countOccurrences(const char *mainText, const char *target);
```

```

int main() {

    char mainText[1000], target[100];


    // Input the main string and the target substring

    printf("Enter the main text: ");

    fgets(mainText, sizeof(mainText), stdin);

    mainText[strcspn(mainText, "\n")] = '\0'; // Remove trailing newline


    printf("Enter the target substring: ");

    fgets(target, sizeof(target), stdin);

    target[strcspn(target, "\n")] = '\0'; // Remove trailing newline


    // Count occurrences of the target substring in the main string

    int count = countOccurrences(mainText, target);


    // Output the result

    printf("The target substring '%s' appears %d times.\n", target, count);


    return 0;
}


int countOccurrences(const char *mainText, const char *target) {

    int count = 0;

    const char *temp = mainText;

```

```

// Loop to search for occurrences of target in mainText

while ((temp = strstr(temp, target)) != NULL) {

    count++;

    temp++; // Move past the current occurrence to avoid overlap

}

return count;

}

```

30. Implement Custom String Length Function

- **Requirement:** Finally, write your own implementation of strlen() function from scratch, demonstrating pointer manipulation techniques.
- **Input:** Any input text.
- **Output:** Length calculated by custom function.

```
#include <stdio.h>
```

```
#include<string.h>
```

```
int customStrlen(const char *str);
```

```
int main() {
```

```
    char inputText[100];
```

```
    // Input a string from the user
```

```
    printf("Enter a string: ");
```

```
    scanf("%s",inputText);
```

```
    // Calculate and output the length of the string using customStrlen
```

```
    int length = customStrlen(inputText);
```

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



```

    return 0;
}

int customStrlen(const char *str) {

    const char *ptr = str; // Pointer to the input string

    int length = 0;

    // Iterate over the string until the null terminator is found
    while (*ptr != '\0') {

        length++;

        ptr++; // Move the pointer to the next character
    }

    return length; // Return the calculated length
}

```

These problem statements provide comprehensive requirements for practicing various functionalities offered by <string.h>, enhancing understanding through practical application in C programming tasks.