1. **String Length Calculation**

o 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.

o Input: A string from the user.

o Output: Length of the string.

```
#include <stdio.h>
#include <string.h>
int main(){
   char str[100];
   printf("Enter a string ");
   scanf("%s",str);
   int str_lenght = strlen(str);
   if(str_lenght == 0){
      printf("Its Empty");
   }
   else{
      printf("The length of giver string is %d",str_lenght);
   }
}
```

2. String Copy

o 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.

o Input: Two strings from the user (source and destination).

o Output: The copied string.

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

int main(){
```

```
    char source[100];
    printf("Enter the source string:  ");
    scanf("%s",source);
    int size;
    printf("Enter the destination size : ");
    scanf("%d",&size);
    char destination[size];

    if(strlen(source)+1 > size){
        printf("Source string doesn't fit into the destination ");
    }
    else{
        strcpy(destination,source);
        printf("The copied string is : %s \n",destination);
    }
    return 0;
}
```

3. String Concatenation

o Requirement: Create a program that concatenates two strings using strcat(). Ensure the destination string has enough space to hold the result.

o Input: Two strings from the user.

o Output: The concatenated string.

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

int main(){
    char str1[100],str2[100];
    printf("Enter the string 1 ");
    scanf("%s",str1);
```

```
    printf("Enter the string 2 ");
    scanf(" %s",str2);
    strcat(str1, str2);
    printf("The Concatenation of two strings are %s\n",str1);
}
```

4. String Comparison

o Requirement: Develop a program that compares two strings using strcmp(). It should indicate if they are equal or which one is greater.

o Input: Two strings from the user.

o Output: Comparison result.

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

int main(){
    char str1[100],str2[100];
    printf("Enter the string 1 ");
    scanf("%s",str1);
    printf("Enter the string 2 ");
    scanf(" %s",str2);
    if(strcmp(str1, str2)){
        printf("Both are equal ");
    }else{
        printf("Not equal ");
    }

}
```

5. Convert to Uppercase

o Requirement: Write a program that converts all characters in a string to uppercase using strupr().

o Input: A string from the user.

o Output: The uppercase version of the string.

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

int main(){
   char str1[100];
   printf("Enter the string 1 ");
   scanf("%s",str1);
   strupr(str1);
   printf("The uppercase version of the string is %s",str1);

}
```

6. Convert to Lowercase

o Requirement: Implement a program that converts all characters in a string to lowercase using strlwr().

o Input: A string from the user.

o Output: The lowercase version of the string.

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

int main(){
   char str1[100];
   printf("Enter the string 1 ");
   scanf("%s",str1);
```

```
    strlwr(str1);
    printf("The lowercase version of the string is %s",str1);

}
```

7. Substring Search

o 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.

o Input: A main string and a substring from the user.

o Output: Starting index or not found message.

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

int main(){
    char str1[] ="The lords of rings";
    char sub[100];
    printf("Enter the sub  ");
    scanf("%s",sub);
    char *result = strstr(str1,sub);
       if (result != NULL) {
        int index = result - str1;
        printf("Substring found at index: %d\n", index);
    } else {

        printf("Substring not found.\n");
    }

    return 0;
```

```
}
```

8. Character Search

o 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.

o Input: A string and a character from the user.

o Output: Index of first occurrence or not found message.

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

int main(){
   char str1[] ="The lords of rings";
   char sub;
   printf("Enter the sub  ");
   scanf("%c",&sub);
   char *result = strchr(str1,sub);
      if (result != NULL) {
       int index = result - str1;
       printf("Character found at index: %d\n", index);
   } else {

       printf("character not found.\n");
   }

   return 0;

}
```

9. String Reversal

o Requirement: Implement a function that reverses a given string in place without using additional memory, leveraging strlen() for length determination.

o Input: A string from the user.

o Output: The reversed string.

```
#include <stdio.h>
#include<string.h>
void reverseString(char str[]);
int main(){
   char str[100];
   printf("Enter a string  ");
   scanf("%s",str);
   reverseString(str);
   printf("Reversed string: %s\n", str);
   return 0;
}
void reverseString(char str[]){
   int start =0;
   int end = strlen(str)-1;
   while(start < end){
      char temp = str[start];
      str[start] = str[end];
      str[end] = temp;
      start++;
      end--;
   }
}
```

10. String Tokenization

o 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[100];
    char *token;
    int count = 0;
    printf("Enter a sentence: ");
    scanf("%[^\n]s", str);
    token = strtok(str, " ");
    while (token != NULL) {
        count++;
        token = strtok(NULL, " ");
    }
    printf("Number of words (tokens): %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[100];
    printf("Enter a sentence: ");
```

```
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';
    char *token = strtok(str, " ");
    int count = 0;
    while(token != NULL){
        count++;
        token = strtok(NULL, " ");
    }
    printf("Number of words (tokens): %d\n", count);
    return 0;
}
```

12. Case-Insensitive Comparison

o Requirement: Develop a program to compare two strings without case sensitivity using strcasecmp() and report equality or differences.

o Input: Two strings from the user.

o Output: Comparison result.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main(){
    char str1[100], str2[100];
    printf("Enter the first string: ");
    fgets(str1, sizeof(str1), stdin);
    str1[strcspn(str1, "\n")] = '\0';
    printf("Enter the second string: ");
    fgets(str2, sizeof(str2), stdin);
    str2[strcspn(str2, "\n")] = '\0';
    if(strcasecmp(str1, str2) == 0){
        printf("The strings are equal.\n");
    } else {
```

```
        printf("The strings are different.\n");
    }
    return 0;
}
```

13. String Trimming

o Requirement: Implement functionality to trim leading and trailing whitespace from a given string, utilizing pointer arithmetic with strlen().

o Input: A string with extra spaces from the user.

o Output: Trimmed version of the string.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main(){
    char str[100];
    printf("Enter a string with extra spaces: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';

    int start = 0, end = strlen(str) - 1;
    while(isspace(str[start])) start++;
    while(isspace(str[end])) end--;

    for(int i = start; i <= end; i++){
        printf("%c", str[i]);
    }
    printf("\n");
    return 0;
}
```

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 main(){
    char str[100];
    char ch;
    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';
    printf("Enter the character: ");
    scanf("%c", &ch);

    int index = -1;
    for(int i = 0; str[i] != '\0'; i++){
        if(str[i] == ch){
            index = i;
        }
    }
    if(index != -1){
        printf("Last occurrence of '%c' is at index: %d\n", ch, index);
    } else {
        printf("Character not found.\n");
    }
    return 0;
}
```

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 <string.h>
#include <ctype.h>
int main(){
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   str[strcspn(str, "\n")] = '\0';
   int count = 0;
   for(int i = 0; str[i] != '\0'; i++){
      if(tolower(str[i]) == 'a' || tolower(str[i]) == 'e' || tolower(str[i]) == 'i' || tolower(str[i]) == 'o' || tolower(str[i]) == 'u'){
         count++;
      }
   }
   printf("Number of vowels: %d\n", count);
   return 0;
}
```

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 <string.h>
int main(){
    char str[100];
    char ch;
    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';
    printf("Enter the character: ");
    scanf("%c", &ch);

    int count = 0;
    for(int i = 0; str[i] != '\0'; i++){
        if(str[i] == ch){
            count++;
        }
    }
    printf("Character '%c' appears %d times.\n", ch, count);
    return 0;
}
```

17. Remove All Occurrences of Character

o Requirement: Write a function that removes all occurrences of a specified character from an input string, modifying it in place.

o Input: A string and a character to remove from it.

o Output: Modified string without specified characters.

```
#include <stdio.h>
#include <string.h>
int main(){
    char str[100];
```

```
    char ch;
    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';
    printf("Enter the character to remove: ");
    scanf("%c", &ch);

    int i = 0, j = 0;
    while(str[i] != '\0'){
        if(str[i] != ch){
            str[j++] = str[i];
        }
        i++;
    }
    str[j] = '\0';
    printf("Modified string: %s\n", str);
    return 0;
}
```

18. Check for Palindrome

o 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.

o Input: A potential palindrome from the user.

o Output: Whether it is or isn't a palindrome.

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

int main() {
    char str[100];
```

```
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

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

    while (start < end) {
        if (isspace(str[start])) {
            start++;
        } else if (isspace(str[end])) {
            end--;
        } else if (tolower(str[start]) != tolower(str[end])) {
            isPalindrome = 0;
            break;
        } else {
            start++;
            end--;
        }
    }

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

    return 0;
}
```

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>

int main() {
    char str[100], substr[100];
    int start, length;

    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    printf("Enter the start index: ");
    scanf("%d", &start);
    printf("Enter the length of the substring: ");
    scanf("%d", &length);

    int len = strlen(str);

    if (start < 0 || start >= len || length < 0 || (start + length) > len) {
        printf("Invalid indices!\n");
    } else {
        strncpy(substr, str + start, length);
        substr[length] = '\0';
        printf("Extracted substring: %s\n", substr);
    }

    return 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>

int main() {
    char str[100];
    printf("Enter the string: ");
    fgets(str, sizeof(str), stdin);

    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]) {
                char temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }

    printf("Sorted string: %s\n", str);
    return 0;
}
```

21. Count Words in String

o Requirement: Write code to count how many words are present in an input sentence by identifying spaces as delimiters, utilizing strtok().

o Input: A sentence from the user.

- Output: Number of words counted.

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

int main() {
    char str[100];
    printf("Enter a sentence: ");
    fgets(str, sizeof(str), stdin);

    int count = 0, i = 0;
    int len = strlen(str);

    while (i < len) {
        while (i < len && isspace(str[i])) i++;  // skip spaces
        if (i < len) count++;  // found a word
        while (i < len && !isspace(str[i])) i++;  // skip characters of the word
    }

    printf("Number of words: %d\n", count);
    return 0;
}
```

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>

int main() {
    char str[100];
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    int len = strlen(str);
    int index = 0;

    for (int i = 0; i < len; i++) {
        int j;
        for (j = 0; j < i; j++) {
            if (str[i] == str[j]) break;
        }
        if (j == i) str[index++] = str[i];
    }

    str[index] = '\0';
    printf("String after removing duplicates: %s\n", str);
    return 0;
}
```

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.**

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

int main() {
    char str[100];
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    int len = strlen(str);
    int count[256] = {0};

    for (int i = 0; i < len; i++) {
        count[str[i]]++;
    }

    int found = 0;
    for (int i = 0; i < len; i++) {
        if (count[str[i]] == 1) {
            printf("First non-repeating character: %c\n", str[i]);
            found = 1;
            break;
        }
    }

    if (!found) printf("No non-repeating character found.\n");
    return 0;
}
```

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 stringToInt(char str[]) {
    int num = 0, i = 0;

    while (str[i] != '\0') {
        if (isdigit(str[i])) {
            num = num * 10 + (str[i] - '0');
        } else {
            return -1;  // Invalid input
        }
        i++;
    }

    return num;
}

int main() {
    char str[100];
    printf("Enter a numeric string: ");
    fgets(str, sizeof(str), stdin);

    int result = stringToInt(str);

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

```
        printf("Converted integer: %d\n", result);
    } else {
        printf("Invalid numeric string.\n");
    }

    return 0;
}
```

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 len = strlen(str);
    for (int i = 0; i < len - 1; i++) {
        for (int j = i + 1; j < len; j++) {
            if (str[i] > str[j]) {
                char temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
}
```

```
int main() {
    char str1[100], str2[100];
    printf("Enter first string: ");
    fgets(str1, sizeof(str1), stdin);
    printf("Enter second string: ");
    fgets(str2, sizeof(str2), stdin);

    sortString(str1);
    sortString(str2);

    if (strcmp(str1, str2) == 0) {
        printf("The strings are anagrams.\n");
    } else {
        printf("The strings are not anagrams.\n");
    }

    return 0;
}
```

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>

int main() {
    char str1[100], str2[100];
```

```
    printf("Enter first string: ");
    fgets(str1, sizeof(str1), stdin);
    printf("Enter second string: ");
    fgets(str2, sizeof(str2), stdin);

    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int maxLength = len1 > len2 ? len1 : len2;

    printf("Merged string: ");
    for (int i = 0; i < maxLength; i++) {
        if (i < len1) printf("%c", str1[i]);
        if (i < len2) printf("%c", str2[i]);
    }
    printf("\n");

    return 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 main() {
    char str[100];
    printf("Enter a string: ");
```

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

    int consonants = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (isalpha(str[i]) && !strchr("aeiouAEIOU", str[i])) {
            consonants++;
        }
    }

    printf("Count of consonants: %d\n", consonants);
    return 0;
}
```

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.

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

int main() {
    char str[100], oldSub[100], newSub[100];
    printf("Enter the main string: ");
    fgets(str, sizeof(str), stdin);
    printf("Enter the substring to replace: ");
    fgets(oldSub, sizeof(oldSub), stdin);
    printf("Enter the new substring: ");
    fgets(newSub, sizeof(newSub), stdin);
```

```
    char* pos = strstr(str, oldSub);
    if (pos) {
        int lenBefore = pos - str;
        char temp[100];
        strcpy(temp, str + lenBefore + strlen(oldSub));
        str[lenBefore] = '\0';
        strcat(str, newSub);
        strcat(str, temp);
        printf("Updated string: %s\n", str);
    } else {
        printf("Substring not found.\n");
    }

    return 0;
}
```

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 main() {
    char str[100], sub[100];
    printf("Enter the main string: ");
    fgets(str, sizeof(str), stdin);
    printf("Enter the substring: ");
```

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

    int count = 0;
    char* pos = str;
    while ((pos = strstr(pos, sub)) != NULL) {
        count++;
        pos++;
    }

    printf("Number of occurrences: %d\n", count);
    return 0;
}
```

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.

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

```
#include <stdio.h>

int my_strlen(char str[]) {
    int length = 0;
    while (str[length] != '\0') {
        length++;
    }
    return length;
}
```

```
int main() {
    char str[100];
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    int len = my_strlen(str);
    printf("Length of the string: %d\n", len);
    return 0;
}
```