Assignment 8

Topics: Advanced String Operations and Preprocessor Directives

Section A8.1: [Advanced String Operations]

A8.1a: Write a C program to concatenate two strings using the strcat() function from string.h.

1. **Instructions**:
2. Include the string.h library.
3. Declare two character arrays (strings), for example: char str1[100] and char str2[100].
4. Use the strcat() function to concatenate str2 at the end of str1.
5. Print the concatenated result.
6. **Example**:

Input: str1 = "Hello, ", str2 = "World!"
Output: "Hello, World!"

A8.1b: Write a program to compare two strings lexicographically using strcmp() from string.h.

1. **Instructions**:
2. Include the string.h library.
3. Declare two character-arrays for strings, e.g., char str1[100] and char str2[100].
4. Use the strcmp() function to compare str1 and str2.
5. Print whether the strings are equal, or if one is lexicographically larger.
6. **Example**:

Input: str1 = "apple", str2 = "banana"
Output: "apple" is lexicographically smaller than "banana".

A8.1c: Write a C program to reverse a string using strrev() (or manually reverse if strrev() is unavailable in some compilers).

1. **Instructions**:
2. Declare a character array for a string.
3. If strrev() is supported by your compiler:
4. Use strrev() to reverse the string.
5. If not supported:
6. Use a loop with two pointers, one starting at the beginning and one at the end, and swap the characters until the pointers meet in the middle.
7. Print the reversed string.
8. **Example**:

Input: "Hello"
Output: "olleH"

A8.1d: Write a program to find the length of a string using strlen() from string.h.

1. **Instructions**:
2. Include the string.h library.
3. Declare a character array for a string, e.g., char str[100].
4. Use strlen() to find and print the length of the string.
5. **Example**:

Input: "Programming"
Output: 11

A8.1e: [Bonus] Write a program to remove all occurrences of a given character from a string.

1. **Instructions**:
2. Declare a character array for the string, e.g., char str[100].
3. Accept a character from the user to remove.
4. Traverse the string using a loop, and whenever the given character is encountered, skip adding it to the result string.
5. Print the modified string.
6. **Example**:

Input: str = "hello world", char = 'o'
Output: "hell wrld"

A8.1f: Write a program to convert a string to uppercase using `strupr()` (or manually if not available).

Instructions:

1. If `strupr()` is available, use it to convert the string to uppercase.
2. If it's not supported by the compiler, loop through each character, check if it's a lowercase letter, and convert it to uppercase manually.

Example: Input: `"hello"` Output: `"HELLO"`

Hint: You can manually convert a lowercase letter by subtracting 32 from its ASCII value.

A8.1g: Write a program to tokenize a string into words using `strtok()` from string.h.

Instructions:

1. Use `strtok()` to split a string into words based on a given delimiter (e.g., space or comma).
2. Print each token (word) on a new line.

Example: Input: `"C is,fun"` Output:

C
is

fun

Hint: Call `strtok()` repeatedly in a loop until it returns `NULL`.

***A8.1h**: [Bonus]*: Write a program to count the occurrences of a substring in a string using `strstr()`.

Instructions:

1. Use `strstr()` to find all occurrences of a given substring in a string.
2. Loop through the string and count how many times the substring appears.

Example: Input: `str = "the theorems in the thesis are important"`, substring = `"the"` Output: `3`

Hint: Use `strstr()` in a loop to move through the string.

Section A8.2: [Preprocessor Directives]

A8.2a: Define and use a macro that calculates the square of a number.

1. **Instructions**:
2. Use the #define preprocessor directive to create a macro called SQUARE(x), which returns x * x.
3. Call this macro in main() to compute the square of a user-provided number.
4. **Example**:

#define SQUARE(x) (x * x)
Input: 4
Output: 16

A8.2b: Write a program that uses the #ifdef and #ifndef preprocessor directives to conditionally include code.

1. **Instructions**:
2. Define a macro called DEBUG.
3. Use #ifdef DEBUG to print debugging information when the macro is defined.
4. Use #ifndef DEBUG to print normal program information if DEBUG is not defined.
5. Comment/uncomment the #define DEBUG line to see the effect.
6. **Example**:

```
#define DEBUG
#ifdef DEBUG
printf("Debugging is ON\n");
#else
printf("Debugging is OFF\n");
#endif
```

A8.2c: Write a C program that uses the #include directive to include a custom header file. Create your own header file that contains a function prototype, and implement that function in the main program.

1. **Instructions**:
2. Create a custom header file mymath.h that contains a function prototype for a function that calculates the cube of a number.
3. In your main program, use #include "mymath.h" to include the header file.
4. Implement the function in the main program and use it to compute the cube of a number.
5. **Example**:

```
// mymath.h
int cube(int x);

// main.c
#include "mymath.h"
```

```
int cube(int x) {
return x * x * x;
}
```

Input: 3
Output: 27

A8.2d: [Bonus] Write a program that uses #pragma to suppress warnings in the GCC compiler for a specific part of the code.

1. **Instructions**:
2. Use the #pragma directive to disable a specific warning, such as an unused variable warning, in a section of the code.
3. Write a function that has an unused variable and suppress the warning using #pragma.
4. Re-enable warnings after the section of code.
5. **Example**:

```
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-variable"
void test() {
int unusedVar;
}
#pragma GCC diagnostic pop
```

Section A8.3: [Preprocessor: Viewing Preprocessed Code]

A8.3a: Write a program that demonstrates the use of macros and conditional compilation. Use the gcc -E command to view the preprocessed code.

1. **Instructions**:
2. Write a simple program that uses macros and #ifdef/#ifndef directives.
3. Compile the program using the following command to see the preprocessed code:

gcc -E your_program.c -o preprocessed_output.txt

1. Open the preprocessed_output.txt file to view the expanded macros and removed comments.
2. **Example**:

```
#define MAX 100
#ifdef MAX
printf("MAX is defined\n");
#endif
```

1. **Output in Preprocessed Code**:

```
printf("MAX is defined\n");
```

Section A8.4: [Additional Problems]

***A8.4a**: Problem 1 (Multiple String Operations with Preprocessor Directives)*

Write a program that accepts a sentence from the user and performs the following operations:

1. Convert the sentence to uppercase using `strupr()` (or manually if unavailable).
2. Count the number of words in the sentence using `strtok()`.
3. Find the length of the sentence using `strlen()`.
4. Remove all occurrences of a specific character using a loop.

Instructions:

1. Use the `#define` preprocessor directive to define the character to be removed (e.g., `#define REMOVE_CHAR 'a'`).
2. If `DEBUG` is defined, print the steps of each operation (e.g., "Converting to uppercase...").
3. Include `string.h` for built-in functions.

Example: Input: `"C programming is awesome!"` Output:

1. Uppercase: `"C PROGRAMMING IS AWESOME!"`
2. Word Count: `4`
3. Length: `25`
4. After removing 'a': `"C progrmming is wesome!"`

Hint: Use `strtok()` for tokenizing and `strlen()` for string length.

***A8.4b**: Problem 2 (Preprocessor Macros with String Functions)*

Write a C program that performs the following:

1. Accept two strings from the user.
2. Concatenate the two strings using `strcat()`.
3. Compare the strings using `strcmp()` and print whether they are equal or one is larger.
4. Define a macro called `TOGGLE_CASE(c)` using the preprocessor to toggle the case of a character. Use this macro to toggle the case of the concatenated string (convert uppercase to lowercase and vice versa).

Instructions:

1. Use the `#define` directive to create the `TOGGLE_CASE` macro.
2. Include conditional compilation: If `DEBUG` is defined, print intermediate steps (e.g., "Comparing strings...").
3. Use `string.h` for built-in string functions.

Example: Input: `"Hello"`, `"World"` Output:

1. Concatenated: `"HelloWorld"`
2. Comparison: `"Hello" is lexicographically smaller than "World"`
3. Toggled Case: `"hELLOwORLD"`

Hint: The macro `TOGGLE_CASE(c)` can be defined as:

#define TOGGLE_CASE(c) (c >= 'a' && c <= 'z' ? c - 32 : (c >= 'A' && c <= 'Z' ? c + 32 : c))