

<b>Course Name:</b>	<b>Structured Programming Methodology</b>	<b>Semester:</b>	<b>I</b>
<b>Date of Performance:</b>	<b>28/10/2025</b>	<b>DIV/Batch No:</b>	<b>A1</b>
<b>Student Name:</b>	<b>Arav Arun</b>	<b>Roll No:</b>	<b>16010125013</b>

### Experiment No: 5

### Title: Strings and String Handling Functions

<b>Aim and Objective of the Experiment:</b>
Write a program in C++ to demonstrate the use of strings and string-handling functions.

<b>COs to be achieved:</b>
CO3. Apply the concepts of arrays and strings.

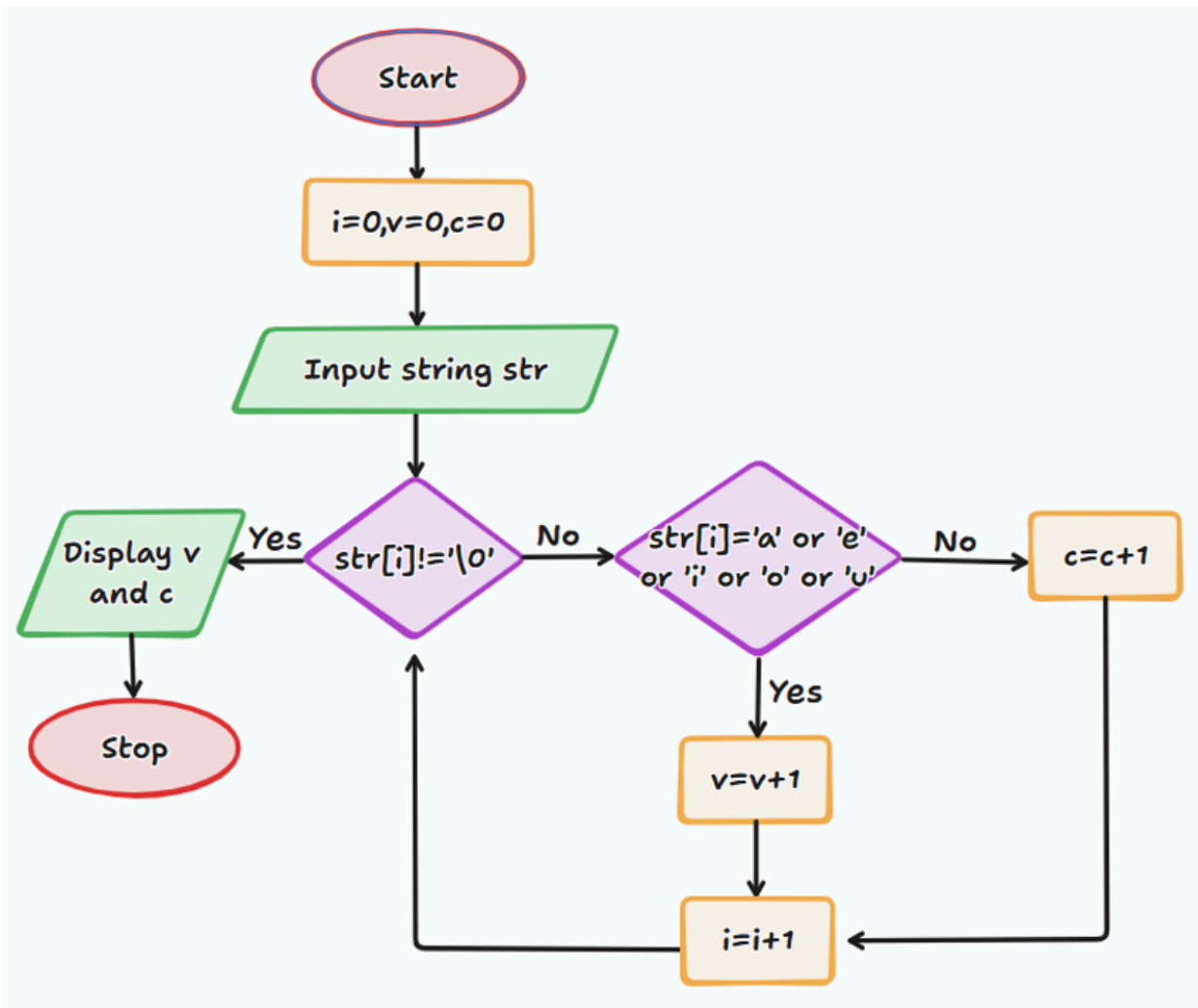
<b>Theory:</b>
<p>In C++, strings can be handled in two ways:</p> <ol style="list-style-type: none"> <li>1. <b>C-style strings</b> — arrays of characters terminated with '\0' (header: &lt;cstring&gt;).</li> <li>2. <b>C++ strings</b> — safer, more flexible objects provided by the Standard Template Library</li> </ol> <p>Common operations on a string s:</p> <ul style="list-style-type: none"> <li>• s.length() – returns number of characters.</li> <li>• s.substr(pos, len) – extracts substring.</li> <li>• s.find(str) – returns index of first occurrence or npos if not found.</li> <li>• s.append(str) or + – concatenates strings.</li> <li>• s.compare(str) – lexicographically compares strings.</li> <li>• getline(cin, s) – reads a line with spaces.</li> <li>• s.erase(), s.insert(), s.replace() – modify content.</li> </ul>

<b>Problem Statements:</b>
<ol style="list-style-type: none"> <li>1. Write a C++ program that takes a string as input and counts the number of vowels and consonants in the string without using the built-in library functions. Ignore spaces and punctuation.</li> <li>2. Write a C++ program to manage student records. The program will handle the following operations using the string functions provided:             <ul style="list-style-type: none"> <li>• Input the student's name and grade (two strings).</li> <li>• Display the length of both the student's name and grade.</li> </ul> </li> </ol>

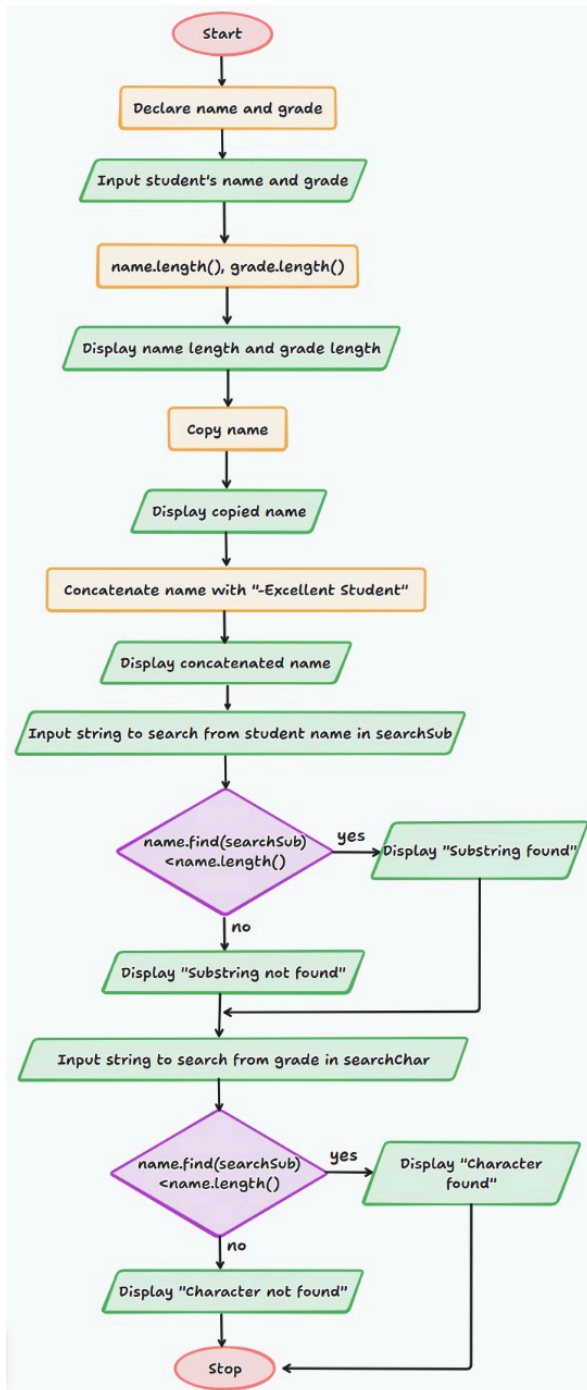
- Copy the student's name into a new string and display it.
- Concatenate a fixed string (e.g., " - Excellent Student") to the student's name and display the result.
- Search for a substring in the student's name (e.g., "John" in "Johnny") and display the position of the first occurrence.
- Search for a character in the grade string (e.g., 'A') and display the position of the first occurrence.

### Flowchart/Algorithm:

#### Flowchart 1 :



**Flowchart 2:**



**Pseudocode:**

**Pseudocode 1 :**

Step 1: Start

Step 2: Declare variables: str to store the input string, v and c to store counts (initialize both to 0)

Step 3: Read the input string from the user.

Step 4: For each character ch in the string:

- a. If ch is an uppercase letter (between 'A' and 'Z'), convert it to lowercase by adding 32 to its ASCII value.
- b. If ch is a lowercase letter (between 'a' and 'z'): If ch is 'a', 'e', 'i', 'o', or 'u', increment v by 1. Else, increment c by 1.
- c. Ignore all other characters (digits, spaces, punctuation, etc.)

Step 5: After processing all characters, display the values of vowels (v) and consonants .

Step 6: Stop.

**Pseudocode 2 :**

Step 1: Start

Step 2: Declare the variables name to store student name, grade to store student's grade, searchSub to store the substring to search and searchChar to store the character to search.

Step 3: Input student's name and grade

Step 4: Display length of student's name and grade using .length()

Step 5: Copy student's name into new string and display it

Step 6: Concatenate the string "-Excellent Student" to the student's name and display it

Step 7: Input a substring to search within the student's name. Use find() to get the position of character. If position is less than the length of grade then display the position. Otherwise display "Character not found."

Step 8: Stop

**Code:**

**Program 1 :**

```
#include<iostream>
using namespace std;
int main()
{
    char str[200];
    int v=0, c=0;
    cout<<"Enter a string: ";
    cin.getline(str,200);
    for(int i=0; str[i]!='\0'; ++i)
    {
        char ch=str[i];
        if(ch>='A' && ch<='Z')
            ch=ch+32;

        if(ch>='a' && ch<='z')
        {
            if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
                v++;
            else
                c++;
        }
    }
    cout<<"Number of vowels: "<<v<<endl;
    cout<<"Number of consonants: "<<c<<endl;
    return 0;
}
```

**Program 2 :**

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

```
using namespace std;

int main()
{
    string name, grade, searchSub;
    char searchChar;

    cout<<"Enter the student's name: ";
    getline(cin, name);
    cout<<"Enter the student's grade: ";
    getline(cin, grade);

    cout<<"Length of student's name: "<<name.length()<<endl;
    cout<<"Length of student's grade: "<<grade.length()<<endl;

    string copiedName=name;
    cout<<"Copied name: "<<copiedName<<endl;

    string concatenatedName=name+"--Excellent Student";
    cout<<"After concatenation: "<<concatenatedName<<endl;

    cout<<"Enter a substring to search in the student's name: ";
    getline(cin, searchSub);

    int posSub=name.find(searchSub);
    if(name.find(searchSub)<name.length())
    {
        cout<<"Substring found at position: "<<posSub<<endl;
    }
    else
    {
        cout<<"Substring not found in the name."<<endl;
    }

    cout<<"Enter a character to search in the student's grade: ";
    cin>>searchChar;

    int posChar=grade.find(searchChar);
```

```
if(grade.find(searchChar)<grade.length())
{
    cout<<"Character found at position: "<<posChar<<endl;
}
else
{
    cout<<"Character not found in the grade."<<endl;
}

return 0;
}
```

**Output:****Output 1 :****Output**

```
Enter a string: My name is Arav
Number of vowels: 5
Number of consonants: 7
```

**Output 2 :****Output**

```
Enter the student's name: Arav
Enter the student's grade: A
Length of student's name: 4
Length of student's grade: 1
Copied name: Arav
After concatenation: Arav--Excellent Student
Enter a substring to search in the student's name: ra
Substring found at position: 1
Enter a character to search in the student's grade: v
Character not found in the grade.
```

## Post Lab Questions

**1. What is the main difference between a C-style string and a C++ std::string?**

**Sol:**

	C-style	C++
TYPE	Array of characters ending with a null terminator ('\0')	Class from the C++ Standard Library
MEMORY MANAGEMENT	Manual — you must allocate and deallocate memory yourself	Automatic — handles allocation, resizing, and deallocation internally
FUNCTIONALITY	Limited; relies on functions from like <code>strcpy</code> , <code>strlen</code> , <code>strcmp</code> , etc.	Rich set of member functions (e.g., <code>.length()</code> , <code>.substr()</code> , <code>.find()</code> , <code>.append()</code> , etc.)
SAFETY	Prone to buffer overflows and memory leaks if not handled carefully	Safer and more convenient due to bounds checking and exception handling

**2. What does the function `s.find("abc")` return if "abc" is not present in s?**

**Sol:** In cpp, the result will be '-1' ; as function won't find any "abc" substring inside s.

**3. How can you safely concatenate two strings s1 and s2 in C++?**

**Sol:**

- Using + Operator  

```
std::string s1 = "Hello, ";
std::string s2 = "World!";
std::string result = s1 + s2; — //Hello, world!
```
- Using .append() Method  

```
std::string s1 = "Hello, ";
std::string s2 = "World!";
s1.append(s2); — // s1 now becomes "Hello, World!"
```

**4. Write a program that takes a sentence as input and converts all lowercase letters to uppercase, and all uppercase letters to lowercase. In C, how does memory allocation for**

**strings work? What are the potential risks associated with string manipulation in C, and how can buffer overflow issues be prevented?**

**Sol:**

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string sentence;
    cout << "Enter a sentence: ";
    getline(cin, sentence); // Reads the full line including spaces
    for (int i = 0; i < sentence.length(); i++) {
        if (isupper(sentence[i])) {
            sentence[i] = tolower(sentence[i]);
        }
        else if (islower(sentence[i])) {
            sentence[i] = toupper(sentence[i]);
        }
    }
    cout << "Converted sentence: " << sentence << endl;
    return 0;
}
```

In C, strings are stored as arrays of characters with a null terminator ('\0'), and memory must be manually allocated and managed.

**Static Allocation:** You can declare a string with a fixed size:

```
char str[100]; // Allocates 100 bytes on the stack
```

**Dynamic Allocation:** You can allocate memory at runtime using malloc or calloc:

```
char *str = malloc(100 * sizeof(char)); // Allocates 100 bytes on the heap
```

**String Literals:** You can assign a string literal to a pointer:

```
char *str = "Hello";
```

### **Risks of String Manipulation in C**

**Buffer Overflow:** Writing beyond the allocated memory can overwrite adjacent memory, crash the program, or allow attackers to inject malicious code.

**Memory Leaks:** Forgetting to free dynamically allocated memory leads to leaks, especially in

long-running programs.

Dangling Pointers: Freeing memory and then using the pointer again causes undefined behavior.

Uninitialized Memory: Using memory before setting its contents can lead to unpredictable Results.

### **How to Prevent Buffer Overflow**

Use Safer Functions: Prefer strncpy, snprintf, and strncat over strcpy, sprintf, and strcat. These limit the number of characters copied.

Always Null-Terminate: Ensure strings end with '\0' to prevent reading past the buffer.

Bounds Checking: Always check the size of the destination buffer before copying or appending.

Use Static and Dynamic Analysis Tools: Tools like Valgrind, AddressSanitizer, and static analyzers can catch memory errors early.

Compiler Hardening Flags: Use flags like -fstack-protector, -D\_FORTIFY\_SOURCE=2, and -Wall to catch risky code during compilation.

Avoid Mixing String Literals and Modifiable Buffers: Never try to modify a string literal.

### **Conclusion:**

This experiment effectively used the concepts of arrays and strings to analyse and process textual data. By storing user input in a character array, the application demonstrated how to manually navigate and work with string elements without relying on built-in functions. We understood how arrays and string processing are applied in actual C++ programming situations. We also learnt about various built in functions that helps us write our programs faster.

**Signature of faculty in charge with date:**