PW SKILLS

Lesson Plan

Strings

PW SKILLS

Topics to be covered:

- Introduction to strings
- Indexing of strings
- ASCII table
- User input string
- String vs character array
- Commonly used inbuilt functions
- Bucket sort and its applications
- Stringstream class

What is a String in C++?

A string in C++ is a type of object representing a collection (or sequence) of different characters.
Strings in C++ are a part of the standard string class (std::string). The string class stores the characters of a string as a collection of bytes in contiguous memory locations.
To use string one must include the header the file #include or the universal header file #include.

Syntax :

string String_Name ;
Example: string str = "pwskills" ;
string subject = "C++";
Different ways of defining a string:
string str_name = "hello coders";
string str_name("physics wallah");
Indexing of characters in a string:

Let's assume any string to be pwcoder . The indexing is similar to indexing in an array. It begins with 0 from the very first character and ends with a null character(\0). An extra space is for a null character after the end of any string.
0 1 2 3 4 5 6 7
p w c o d e r \0

ASCII values:

Each character has an associated integer/numeric value.
For example 'a' to 'z' ranges from 97 to 122.
Where a has a value 97, b has a value 98, c for 99 and so on.
A -> 65
B ->66
|
|
|
Z->90
There are numeric values for other characters such as #, @, $ etc.
For reference one can check the ascii table for numeric value of any character whose link is provided as below.
www.cs.cmu.edu
https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html

Taking string input:

To provide our program's input from the user, we generally use the cin keyword along with the extraction operator (>>). By default, the extraction operator considers white space (such as space, tab, or newline) as the terminating character. So, suppose the user enters "Physics Wallah" as the input. In that case, only "Physics" will be considered input, and "Wallah" will be discarded. Let us take a simple example to understand this:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string str;
cout << "Enter your string: ";
cin >> str;
// The user input will be stored in the str variable
cout << "You have entered: " << str;
return 0;
}
```

Output:
Enter your string: pwskills
You have entered: pwskills

In the above example, the user entered "Physics Wallah" in the input. As " " is the terminating character, anything written after " " was discarded. Hence, we got "Physics" as the output. To counter this limitation of the extraction operator, we can specify the maximum number of characters to read from the user's input using the cin.get() function.

getline(cin,str)

Let us take an example to understand this:

```
#include <iostream>
using namespace std;
int main() {
char str[50];
cout << "Enter your string: ";
cin.get(str, 50);
cout << "You have entered: " << str << endl;
return 0;
}
```

Output:
enter your string: Physics Wallah
You have entered: Physics Wallah
Here we have declared a character array, of maximum size of 50 characters.
We have taken input by writing the statement:

cin.get(str_name , lengthOfString);

Commonly used inbuilt string functions:

Reverse() : This function accepts 2 parameters and reverses the string from beginning pointer to the end pointer.

Syntax: reverse(ptr1 , ptr2) . The first pointer ptr1 is included and the second pointer ptr2 is not included. That means the string from ptr1 to ptr2 - 1 will be reversed.
Time complexity: let n = ptr2 - ptr1. Therefore time taken by the reverse() function is O(n) where n is the number of characters involved in the reverse process.
The code showing reverse function is illustrated below:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string s;
cout<<"enter your string: ";
cin>>s;
reverse(s.begin() , s.end());
cout<<"The reversed string is: "<<s;
}
```

Output:
enter your string: PWskills
The reversed string is: slliksWP

substr(): This function is used to generate a substring of a given string.

Syntax: str_name.substr(position , length)

This is the general syntax where we provide the name of the string whose substring is required. The first parameter indicates the position from where the substring extraction begins, and the second parameter indicates the length till where you want the substring.
An example illustrating the function is given below:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string s;
cout<<"enter your string: ";
cin>>s;
string str = s.substr(3 , 5);
cout<<"The substring obtained is: "<<str;
}
```

Output:
enter your string: physicsWallah
The substring obtained is: sicsW
Here we have given the starting index as 3 and want the substring of length 5.
To get a substring after a particular character:

Syntax 2: str_name.substr(position)

As per this syntax, the complete string after the mentioned "position" will be extracted as a substring as shown below in the example.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string s;
cout<<"enter your string: ";
cin>>s;
string str = s.substr(3);
cout<<"The substring obtained is: "<<str;
}
```

Output:

enter your string: physicswallah
The substring obtained is: sicwallah
Here the complete string after index 3 (including index 3) is obtained as a substring.
To get a substring before any character:

Syntax 3:

str_name.substr(0 , end_position)
Here the string from 0 to end_position - 1 is considered as the required substring.
This is a specific case of general syntax.

push_back(): This function is used to push another character or string at the end of the current string(string with which the function is used/called).

Syntax: str_name.push_back(str2_name) str2_name string will be pushed at the end of the string str_name as shown in the following example.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string s;
cout<<"enter your string: ";
cin>>s;
cout<<"enter the new string or character to be pushed: ";
char t;
cin>>t;
s.push_back(t);
cout<<"The new string obtained is: "<<s;
}
```

Output:

enter your string : physics
enter the new string or character to be pushed: wallah
The new string obtained is: physicswallah
The "+" operator: this is directly used to concatenate 2 strings.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string s;
cout<<"enter your string: ";
cin>>s;
cout<<"enter the new string or character to be pushed: ";
string t;
cin>>t;
s += t;
cout<<"The new string obtained is: "<<s;
}
```

Output:
enter your string : physics
enter the new string or character to be pushed: wallah
The new string obtained is: physicswallah

Here it was written as s+=t that is equivalent to s = s + t, had it been written s = t +s then the resulting string would be wallahphysics .
When we write s+=t that signifies we are appending the string t at the back of string s.
Statement 2 s = s + t shows we have created a new copy of string s and added string t at the back of string s.
The only difference in the above two statements is that in the second case extra space will be taken by the reformation of string s whereas in the first case no such extra space of string s will be occupied.
Homework: Try t+s functionality on your own.

strcat(): This function is used to concatenate 2 character arrays.

Syntax: strcat(s1_name , s2_name)
This is equivalent to s1_name = s1_name + s2_name as illustrated below:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
char s[20];
cout<<"enter your string: ";
cin>>s;
cout<<"enter the new string or character to be pushed: ";
char t[20];
cin>>t;
strcat(s,t);
cout<<"The new string obtained is: "<<s;
}
```

Output:
enter your string : physics
enter the new string or character to be pushed: wallah
The new string obtained is: physicswallah

size(): this function is used to find the size of the string.

Syntax: str_name.size()

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string str;
cin>>str;
cout<<str.size();
}
```

Output:
pwcoders
8

Difference between size() and strlen() functions:

size() function is used to know the length of the strings whereas strlen() function is used to know the length of the character array. strlen() function takes O(n) time whereas size() function uses O(1) time, where n = length of the array. **to_string(): This function is used to convert a numeric value into string type.**

Syntax:
Suppose n is an integer. To convert this integer to string we have to write the following way:
to_string(n)
This function is used when we have to perform operations on digits of a number. Converting into string makes the digits accessible in an indexed manner which is quite easy to use.
The following code illustrates:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
cout<<"enter the number: ";
int n;
cin>>n;
string str = to_string(n);
cout<<str<<endl;
cout<<str[0]<<" "<<str[1]<<" "; //printing the first two
digits(from left to
right / highest to lowest priority) of the entered number
}
```

Output:
3452

Bucket sort on strings:

We have 128 different types of characters available that can be A-Z, a-z, special characters such as !, @, # etc, digits from '0' to '9'etc. Since the maximum number of characters can be at max 128 we can use an array or vector of size 128 such that each of the index of the array represents a particular character. Now we can use the ASCII values of the characters to mark the indices of the characters. For example,
The ASCII value of 'a' = 97
'b' = 98 and so on. . . . .
So if we created an array 'arr' of size 128 then arr[97] will be reserved for the frequency of 'a', similarly arr[98] will be reserved for frequency of 'b' and so on.
Once the frequencies of every character are stored we can use it to build a sorted string, because it is sure that 'a' will always be ahead of 'b', 'b' will always be ahead of 'c' and so on. This is the concept of bucket sort. Below example will clarify this more.
Example , Given a string 'str', sort the given string using count sort technique where the string only contains lowercase alphabetic characters (a - z).

```
#include<bits/stdc++.h>
using namespace std;
string countSort(string str) {
vector<int>arr(26 , 0); //array to store the frequencies of all
alphabets
for(int i=0;i<str.size();i++){
arr[str[i] - 'a']++; //storing the required frequencies of
particular
Character
}
int n = str.size();
int j = 0;
for(int i=0;i<26;i++){
while(arr[i]--){ //iterating till the frequency of a particular
character and appending to the string
str[j++] = i + 'a';
}
}
return str; //returning the sorted string
}
int main()
{
 string s;
cout<<"Enter the string: "<<endl;
cin>>s;
}
```

Stringstream class :

A stringstream associates a string object with a stream allowing you to read from the string as if it were a stream (like cin). To use stringstream, we need to include sstream header file. The stringstream class is extremely useful in parsing input.

```
#include<bits/stdc++.h>
using namespace std;
void print(string str){
stringstream s(str);
string word;
while (s >> word){
cout<<word<<endl;
}
Return;
}
int main(){
char s[50];
cout<<"Enter the string : ";
gets(s);
cout<<"Desired output is: "<<endl;
print(s);
return 0;
}
```

Output:
Enter the string : "PW is a revolution"
Desired output is: PW is a revolution

Q. Input a string of length n and count all the vowels in the given string.

```
#include <iostream>
#include <string>
using namespace std;

int countVowels(const string& str) {
    int vowelCount = 0;

    for (char ch : str) {
        // Convert the character to lowercase for case-
insensitivity
        char lowercaseCh = tolower(ch);

        // Check if the character is a vowel
        if (lowercaseCh == 'a' || lowercaseCh == 'e' ||
lowercaseCh == 'i' || lowercaseCh == 'o' || lowercaseCh == 'u') {
            vowelCount++;
        }
    }

    return vowelCount;
}

int main() {
    string inputString;
```

```
    // Input a string
    cout << "Enter a string: ";
    getline(cin, inputString);

    // Count vowels in the string
    int vowels = countVowels(inputString);

    // Output the result
    cout << "Number of vowels in the string: " << vowels << endl;

    return 0;
}
```

Ques : Input a string of size n and Update all the even positions in the string to character 'a'. Consider 0-based indexing.

```
#include <iostream>
#include <string>
using namespace std;

void updateEvenPositions(string& str) {
    // Iterate through the string and update even positions to
'a'
    for (int i = 0; i < str.length(); i += 2) {
        str[i] = 'a';
    }
}

int main() {
    string inputString;

    // Input a string
    cout << "Enter a string: ";
    getline(cin, inputString);

    // Update even positions to 'a'
    updateEvenPositions(inputString);

    // Output the updated string
    cout << "Updated string: " << inputString << endl;

    return 0;
}
```

Q. Input a string of even length and reverse the first half of the string.

```
#include <iostream>
#include <string>
using namespace std;

void reverseFirstHalf(string& str) {
    int length = str.length();

    // Ensure the string has even length
    if (length % 2 != 0) {
        cout << "Error: Input string must have even length." <<
endl;
        return;
    }

    // Calculate the midpoint of the string
    int midpoint = length / 2;

    // Reverse the first half of the string
    for (int i = 0; i < midpoint / 2; i++) {
        swap(str[i], str[midpoint - 1 - i]);
    }
}

int main() {
    string inputString;

    // Input a string
    cout << "Enter a string of even length: ";
    getline(cin, inputString);

    // Reverse the first half of the string
    reverseFirstHalf(inputString);

    // Output the updated string
    cout << "Updated string: " << inputString << endl;

    return 0;
}
```

Q. Input a string of length greater than 5 and reverse the substring from position 2 to 5 using inbuilt functions.

```
#include <iostream>
#include <string>
#include <algorithm> // for reverse function
using namespace std;

void reverseSubstring(string& str) {
    // Check if the length is greater than 5
```

```
    if (str.length() <= 5) {
        cout << "Error: Input string must have length greater
than 5." << endl;
        return;
    }

    // Reverse the substring from position 2 to 5
    reverse(str.begin() + 1, str.begin() + 5);
}

int main() {
    string inputString;

    // Input a string
    cout << "Enter a string of length greater than 5: ";
    getline(cin, inputString);

    // Reverse the substring from position 2 to 5
    reverseSubstring(inputString);

    // Output the updated string
    cout << "Updated string: " << inputString << endl;

    return 0;
}
```

Q. Input a string of even length and return the second half of that string using inbuilt substr function

```
#include <iostream>
#include <string>
using namespace std;

string getSecondHalf(const string& str) {
    int length = str.length();

    // Ensure the string has even length
    if (length % 2 != 0) {
        cout << "Error: Input string must have even length." <<
endl;
        return "";
    }

    // Calculate the midpoint of the string
    int midpoint = length / 2;

    // Return the second half of the string using substr
    return str.substr(midpoint);
}
```

```
int main() {
    string inputString;

    // Input a string
    cout << "Enter a string of even length: ";
    getline(cin, inputString);

    // Get and output the second half of the string
    string secondHalf = getSecondHalf(inputString);
    cout << "Second half of the string: " << secondHalf << endl;

    return 0;
}
```

Q. Return the total number of digits in a number without using any loop.

Hint : Try using inbuilt to_string() function.

```
#include <iostream>
#include <string>
using namespace std;

int countDigits(int number) {
    // Convert the number to a string using to_string
    string strNumber = to_string(number);

    // Return the length of the string, which is the total number
of digits
    return strNumber.length();
}

int main() {
    int inputNumber;

    // Input a number
    cout << "Enter a number: ";
    cin >> inputNumber;

    // Get and output the total number of digits
    int digitCount = countDigits(inputNumber);
    cout << "Total number of digits: " << digitCount << endl;

    return 0;
}
```

Q. Input a string and return the number of times the neighbouring characters are different from each other.

```
#include <iostream>
#include <string>
using namespace std;

int countDifferentNeighbours(const string& str) {
    int count = 0;

    // Iterate through the string (excluding the last character)
    for (int i = 0; i < str.length() - 1; ++i) {
        // Check if neighboring characters are different
        if (str[i] != str[i + 1]) {
            count++;
        }
    }

    return count;
}

int main() {
    string inputString;

    // Input a string
    cout << "Enter a string: ";
    getline(cin, inputString);

    // Get and output the number of times neighboring characters
are different
    int differenceCount = countDifferentNeighbours(inputString);
    cout << "Number of times neighboring characters are
different: " << differenceCount << endl;

    return 0;
}
```

Q. Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Input : s = physicswallah, t = wallahphysics Output: YES

```
#include <iostream>
#include <unordered_map>
using namespace std;

bool areAnagrams(const string& s, const string& t) {
    // Check if the lengths of the strings are different
    if (s.length() ! = t.length()) {
        return false;
    }

    // Use unordered_map to store the frequency of each character
in string s
    unordered_map<char, int> charFrequency;
```

```
    // Increment frequency for each character in string s
    for (char ch : s) {
        charFrequency[ch]++;
    }

    // Decrement frequency for each character in string t
    for (char ch : t) {
        charFrequency[ch]--;

        // If a character in string t is not found in string s or
its frequency becomes negative, return false
        if (charFrequency[ch] < 0) {
            return false;
        }
    }

    // Check if all character frequencies are zero (indicating
both strings have the same characters)
    for (const auto& pair : charFrequency) {
        if (pair.second ! = 0) {
            return false;
        }
    }

    return true;
}

int main() {
    string s, t;

    // Input two strings
    cout << "Enter the first string (s): ";
    cin >> s;
    cout << "Enter the second string (t): ";
    cin >> t;

    // Check if t is an anagram of s and output the result
    if (areAnagrams(s, t)) {
        cout << "YES" << endl;
    } else {
        cout << "NO" << endl;
    }

    return 0;
}
```

Q. Given n strings consisting of lowercase English alphabets. Print the character that is occurring most number of times.

```
#include <iostream>
#include <unordered_map>
using namespace std;

char mostFrequentCharacter(const string& s) {
    // Use unordered_map to store the frequency of each character
    unordered_map<char, int> charFrequency;

    // Count the frequency of each character in the string
    for (char ch : s) {
        charFrequency[ch]++;
    }

    // Find the character with the maximum frequency
    char mostFrequentChar = '\0'; // Initialize to null character
    int maxFrequency = 0;

    for (const auto& pair : charFrequency) {
        if (pair.second > maxFrequency) {
            maxFrequency = pair.second;
            mostFrequentChar = pair.first;
        }
    }

    return mostFrequentChar;
}

int main() {
    int n;
    cout << "Enter the number of strings (n): ";
    cin >> n;

    // Clear the newline character from the input buffer
    cin.ignore();

    // Input n strings
    string inputString;
    char mostFrequentChar = '\0';

    for (int i = 0; i < n; ++i) {
        cout << "Enter string " << i + 1 << ": ";
        getline(cin, inputString);

        // Call the mostFrequentCharacter function for each
string
        char currentMostFrequentChar =
mostFrequentCharacter(inputString);

        // Update mostFrequentChar if needed
        if (i == 0 || currentMostFrequentChar > mostFrequentChar)
{
            mostFrequentChar = currentMostFrequentChar;
```

```
        }
    }

    // Output the most frequent character
    cout << "Most frequent character: " << mostFrequentChar <<
endl;

    return 0;
}
```

Q. Given a sentence, split every single word of the sentence and print in a new line.

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

void splitAndPrintWords(const string& sentence) {
    // Use stringstream to split the sentence into words
    stringstream ss(sentence);
    string word;

    // Iterate through each word and print on a new line
    while (ss >> word) {
        cout << word << endl;
    }
}

int main() {
    string inputSentence;

    // Input a sentence
    cout << "Enter a sentence: ";
    getline(cin, inputSentence);

    // Split and print each word on a new line
    splitAndPrintWords(inputSentence);

    return 0;
}
```

Q. Given a sentence 'str', return the word that is occurring most number of times in that sentence.

```
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_map>
using namespace std;
```

```
string mostFrequentWord(const string& sentence) {
    // Use stringstream to split the sentence into words
    stringstream ss(sentence);
    string word;

    // Use unordered_map to store the frequency of each word
    unordered_map<string, int> wordFrequency;

    // Count the frequency of each word in the sentence
    while (ss >> word) {
        wordFrequency[word]++;
    }

    // Find the word with the maximum frequency
    string mostFrequentWord;
    int maxFrequency = 0;

    for (const auto& pair : wordFrequency) {
        if (pair.second > maxFrequency) {
            maxFrequency = pair.second;
            mostFrequentWord = pair.first;
        }
    }

    return mostFrequentWord;
}

int main() {
    string inputSentence;

    // Input a sentence
    cout << "Enter a sentence: ";
    getline(cin, inputSentence);

    // Find and output the most frequent word in the sentence
    string result = mostFrequentWord(inputSentence);
    cout << "Most frequent word: " << result << endl;

    return 0;
}
```

Q. Given n string consisting of digits from 0 to 9. Return the index of string which has maximum value. (Take 0 based indexing)

Input : 0123, 0023, 456, 00182, 940, 2901 Output: 5

```
#include <iostream>
#include <vector>
using namespace std;

int indexOfMaxValue(const vector<string>& strings) {
    int maxIndex = 0;
    int maxValue = stoi(strings[0]); // Convert the first string
to an integer

    for (int i = 1; i < strings.size(); ++i) {
        int currentNumber = stoi(strings[i]); // Convert the
current string to an integer

        if (currentNumber > maxValue) {
            maxValue = currentNumber;
            maxIndex = i;
        }
    }
    return maxIndex;
}

int main() {
    int n;
    cout << "Enter the number of strings (n): ";
    cin >> n;

    // Clear the newline character from the input buffer
    cin.ignore();

    vector<string> inputStrings(n);

    // Input n strings
    for (int i = 0; i < n; ++i) {
        cout << "Enter string " << i + 1 << ": ";
        getline(cin, inputStrings[i]);
    }

    // Find and output the index of the string with the maximum
value
    int result = indexOfMaxValue(inputStrings);
    cout << "Index of string with maximum value: " << result <<
endl;

    return 0;
}
```

Q. Input n strings and write a program to find the longest common prefix string of all the strings.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

string longestCommonPrefix(const vector<string>& strings) {
    if (strings.empty()) {
        return ""; // If there are no strings, return an empty
string
    }

    // Sort the strings to easily find the common prefix
    vector<string> sortedStrings = strings;
    sort(sortedStrings.begin(), sortedStrings.end());

    // Compare the first and last strings (sorted) to find the
common prefix
    string firstString = sortedStrings.front();
    string lastString = sortedStrings.back();
    string commonPrefix;

    for (size_t i = 0; i < firstString.length() && i <
lastString.length(); ++i) {
        if (firstString[i] ! = lastString[i]) {
            commonPrefix += firstString[i];
        } else {
            break;
        }
    }

    return commonPrefix;
}

int main() {
    int n;
    cout << "Enter the number of strings (n): ";
    cin >> n;

    // Clear the newline character from the input buffer
    cin.ignore();

    vector<string> inputStrings(n);

    // Input n strings
    for (int i = 0; i < n; ++i) {
        cout << "Enter string " << i + 1 << ": ";
        getline(cin, inputStrings[i]);
    }

    // Find and output the longest common prefix
    string result = longestCommonPrefix(inputStrings);
    cout << "Longest common prefix: " << result << endl;
```

```
    return 0;
}
```

Q. Given two strings s and t, determine if they are isomorphic.

```
#include <iostream>
#include <unordered_map>
#include <unordered_set>
using namespace std;

bool isIsomorphic(const string& s, const string& t) {
    if (s.length() ! = t.length()) {
        return false; // If the lengths are different, strings
cannot be isomorphic
    }
    unordered_map<char, char> charMapping;
    unordered_set<char> mappedChars;

    for (int i = 0; i < s.length(); ++i) {
        char sChar = s[i];
        char tChar = t[i];

        // If sChar is already mapped, check if the mapping is
consistent
        if (charMapping.find(sChar) ! = charMapping.end()) {
            if (charMapping[sChar] ! = tChar) {
                return false; // Inconsistent mapping
            }
        } else {
            // If tChar is already used in the mapping, return
false
            if (mappedChars.find(tChar) ! = mappedChars.end()) {
                return false;
            }

            // Create a new mapping
            charMapping[sChar] = tChar;
            mappedChars.insert(tChar);
        }
    }

    return true;
}

int main() {
    string s, t;

    // Input two strings
    cout << "Enter the first string (s): ";
    cin >> s;
```

```
    cout << "Enter the second string (t): ";
    cin >> t;

    // Check if the strings are isomorphic and output the result
    if (isIsomorphic(s, t)) {
        cout << "The strings are isomorphic." << endl;
    } else {
        cout << "The strings are not isomorphic." << endl;
    }

    return 0;
}
```

PW SKILLS

THANK YOU !

PW SKILLS