All palindromic substrings in C++

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

bool isPalindrome(string s) {
    int i = 0;
    int j = s.length() - 1;
    while (i <= j) {
        if (s[i] != s[j]) {
            return false;
        } else {
            i++;
            j--;
        }
    }
    return true;
}

void solution(string str) {
    // Write your code here
    for (int i = 0; i < str.length(); i++) {
        for (int j = i + 1; j <= str.length(); j++) {
            string ss = str.substr(i, j - i);
            if (isPalindrome(ss)) {
                cout << ss << "\n";
            }
        }
    }
}

int main() {
    string str = "abcc";
    solution(str);
    return 0;
}
```

Dry Run for Input: "abcc"

Let's list all substrings and mark which are palindromes:

Substring	From i	To j	Palindrome?
"a"	0	1	✓
"ab"	0	2	✗
"abc"	0	3	✗
"abcc"	0	4	✗
"b"	1	2	✓
"bc"	1	3	✗
"bcc"	1	4	✗
"c"	2	3	✓
"cc"	2	4	✓
"c"	3	4	✓

✓ Final Output (printed substrings):

a
b
c
cc
c

a
b
c
cc
c

Difference of every two consecutive character in C++

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

string solution(string str) {
    if (str.empty()) return "";

    string result;
    result += str[0]; // Append
the first character directly
    for (int i = 1; i <
str.length(); i++) {
        char curr = str[i];
        char prev = str[i - 1];
        int gap = curr - prev;

        result += to_string(gap);
// Append the difference as a
string
        result += curr; //
Append the current
character
    }

    return result;
}

int main() {
    string str = "pepCODinG";
    cout << solution(str) <<
endl;
    return 0;
}
```

Input String: "pepCODinG"

Index (i)	prev	curr	ASCII(prev)	ASCII(curr)	Difference curr - prev	Intermediate Result
0	–	p	–	112	–	p
1	p	e	112	101	-11	p-11e
2	e	p	101	112	11	p-11e11p
3	p	C	112	67	-45	p-11e11p-45C
4	C	O	67	79	12	p-11e11p-45C12O
5	O	D	79	68	-11	p-11e11p-45C12O-11D
6	D	i	68	105	37	p-11e11p-45C12O-11D37i
7	i	n	105	110	5	p-11e11p-45C12O-11D37i5n
8	n	G	110	71	-39	p-11e11p-45C12O-11D37i5n-39G

✅ **Final Output:**

p-11e11p-45C12O-11D37i5n-39G

p-11e11p-45C12O-11D37i5n-39G

Remove Primes in C++

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

bool isPrime(int val) {
    if (val <= 1) return false; // 0 and 1 are not prime
numbers
    for (int i = 2; i * i <= val; i++) {
        if (val % i == 0) {
            return false;
        }
    }
    return true;
}

void solution(vector<int>& nums) {
    for (int i = nums.size() - 1; i >= 0; i--) {
        if (isPrime(nums[i])) {
            nums.erase(nums.begin() + i); // Remove prime
number
        }
    }
}

int main() {
    vector<int> nums = {3, 12, 13, 15};

    solution(nums);

    for (int num : nums) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

Dry Run for Input:

```
vector<int> nums = {3, 12, 13, 15};
```

Index	Value	isPrime?	Action
3	15	✗	Keep
2	13	✓	Remove
1	12	✗	Keep
0	3	✓	Remove

✓ Final `nums = {12, 15}`

🖨 Output:

```
12 15
```

12 15

String Compression in C++

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

string compression1(string str) {
    if (str.empty()) return ""; // Handle edge case

    string s;
    s += str[0]; // Append the first character directly
    for (int i = 1; i < str.length(); i++) {
        char curr = str[i];
        char prev = str[i - 1];
        if (curr != prev) {
            s += curr; // Append only if current character is
different from previous
        }
    }
    return s;
}

string compression2(string str) {
    if (str.empty()) return ""; // Handle edge case

    string s;
    s += str[0]; // Append the first character directly
    int count = 1;
    for (int i = 1; i < str.length(); i++) {
        char curr = str[i];
        char prev = str[i - 1];
        if (curr == prev) {
            count++; // Increment count for consecutive
characters
        } else {
            if (count > 1) {
                s += to_string(count); // Append count if it's
greater than 1
                count = 1; // Reset count
            }
            s += curr; // Append current character
        }
    }
    if (count > 1) {
        s += to_string(count); // Append the final count if
needed
    }
    return s;
}

int main() {
    string str = "wwwwaaadexxxxxx";
    cout << compression1(str) << endl;
    cout << compression2(str) << endl;
    return 0;
}
```

Step-by-Step Dry Run: compression2("wwwwaaadexxxxxx")

i	curr	prev	count	Output so far	Action
1	w	w	2	w	same, count++
2	w	w	3	w	same, count++
3	w	w	4	w	same, count++
4	a	w	1	w4a	append 4, then a
5	a	a	2	w4a	same, count++
6	a	a	3	w4a	same, count++
7	d	a	1	w4a3d	append 3, then d
8	e	d	1	w4a3de	different, append e
9	x	e	1	w4a3dex	append x
10	x	x	2	w4a3dex	same, count++
11	x	x	3	w4a3dex	same, count++
12	x	x	4	w4a3dex	same, count++
13	x	x	5	w4a3dex	same, count++
14	x	x	6	w4a3dex	same, count++
end				w4a3dex6	append 6

🖳 **Final Output**

```
wadex
w4a3dex6
```

wadex
w4a3dex6

Toggle in C++

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

void toggle(char ch[]) {
    for (int i = 0; ch[i] != '\0'; i++) {
        if (ch[i] >= 'A' && ch[i] <= 'Z') {
            ch[i] = ch[i] + 32; // Convert uppercase to
lowercase
        } else if (ch[i] >= 'a' && ch[i] <= 'z') {
            ch[i] = ch[i] - 32; // Convert lowercase to
uppercase
        }
    }
}

int main() {
    char st[] = "kriSh";

    toggle(st);

    cout << st << endl; // Output the modified string

    return 0;
}
```

Dry Run for char st[] = "kriSh"

Index	Character	Condition	ASCII Before	ASCII After	New Char
0	'k'	lowercase → UPPER	107	75	'K'
1	'r'	lowercase → UPPER	114	82	'R'
2	'i'	lowercase → UPPER	105	73	'I'
3	'S'	uppercase → lower	83	115	's'
4	'h'	lowercase → UPPER	104	72	'H'

🔁 Modified string becomes: **"KRIsH"**

✔ **Output**

KRIsH

KRIsH