

```
class Solution {
public:
    string reverseWords(string s) {
        auto checkvowel=[&](string &s ){
            int cnt=0;
            for(int i =0;i<s.size();i++){
                if(s[i]
== 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u') cnt++;
            }
            return cnt;
        };

        stringstream ss(s);
        vector<string> words;
        string w;
        while(ss >> w) words.push_back(w);
        if(words.empty()){
            return s;
        }
        int x = checkvowel(words[0]);

        for(int i =1;i<words.size();i++){
            if(checkvowel(words[i])==x){
                reverse(words[i].begin(),words[i].end());
            }
        }
        string ans = "";
        for(int i=0; i<words.size();i++){
            if(i) ans += " ";
            ans += words[i];
        }
        return ans;
    }
};
```

```
}  
};
```

in this code thier is operation

```
stringstream ss(s);
```

```
vector<string>words;
```

```
string w;
```

```
while(ss>>w)words.push_back(w);
```

how stringstream works in cpp in

which type of situation and cases it can be used

take a give a good explanation

Good question — `stringstream` is a **very important CP + C++ concept**, and you're already using it correctly. Let's break it down **slowly, clearly, and practically** (no fluff).

## 1 What is `stringstream`?

`stringstream` is a **stream over a string**, just like:

- `cin` → reads from **keyboard**
- `ifstream` → reads from **file**
- `stringstream` → reads from **string**

cpp

```
#include <sstream>
```

 Copy code

It allows you to **treat a string as input/output stream**.

## 2 What does this line do?

cpp

```
stringstream ss(s);
```

 Copy code

- You create a stream object `ss`
- It **contains the string** `s`
- Now you can **extract data from** `s` using `>>` just like `cin`

Example:

cpp

```
string s = "cat and mice";  
stringstream ss(s);
```

 Copy code

Internally, the stream looks like:

nginx

```
cat and mice  
^
```

 Copy code

(pointer starts at beginning)

---

### 3 How does this loop work?

cpp

```
vector<string> words;  
string w;  
  
while (ss >> w) {  
    words.push_back(w);  
}
```

 Copy code

### Step-by-step execution

Given:

cpp

```
s = "cat and mice";
```

 Copy code

### Iteration 1

cpp


```
ss >> w;    // reads until whitespace  
w = "cat"
```

 Copy code

## Iteration 2

```
cpp
```

```
w = "and"
```

 Copy code

## Iteration 3

```
cpp
```

```
w = "mice"
```

 Copy code

## Iteration 4

- Stream reaches **end**
- `ss >> w` fails
- loop stops

## Final result

```
cpp
```

```
words = {"cat", "and", "mice"};
```

 Copy code

✓ Whitespace ( `space` , `\n` , `\t` ) is automatically handled

## 4 Why does `ss >> w` stop automatically?

Because:

- `>>` skips leading whitespace
- Reads until next whitespace
- Returns **false** when no more tokens exist

So this:

```
cpp
```

```
while (ss >> w)
```

 Copy code

means:

“Keep extracting words while extraction succeeds”

## 5 Why use `stringstream` instead of manual parsing?

### ✗ Manual way (error-prone)

```
cpp

string cur = "";
for (char c : s) {
    if (c == ' ') {
        words.push_back(cur);
        cur.clear();
    } else {
        cur += c;
    }
}
if (!cur.empty()) words.push_back(cur);
```

 Copy code

### ✓ `stringstream` way (clean)

```
cpp

stringstream ss(s);
while (ss >> w) words.push_back(w);
```

 Copy code

✓ Handles:

- multiple spaces
- leading/trailing spaces
- tabs & newlines

## 6 Common Situations Where `stringstream` Is Used

### ◆ 1. Split string into words (most common)

```
cpp
```

```
string s = " hello world ";
stringstream ss(s);
string w;
while (ss >> w) cout << w << endl;
```

 Copy code

Output:

```
nginx

hello
world
```

 Copy code

---

## ♦ 2. Convert string → number

cpp

```
string s = "12345";
stringstream ss(s);
int x;
ss >> x;
```

 Copy code

✓ Used when `stoi` is unsafe or mixed input exists

---

## ♦ 3. Parse mixed data

cpp

```
string s = "10 20 30";
stringstream ss(s);
int a, b, c;
ss >> a >> b >> c;
```

 Copy code

---

## ♦ 4. LeetCode / CP input handling

Example:

cpp

 Copy code

```
string s = "a b c d";
```

Convert into vector:

```
cpp

vector<string> v;
stringstream ss(s);
string temp;
while (ss >> temp) v.push_back(temp);
```

 Copy code

## ◆ 5. Reformat string

```
cpp

stringstream ss;
ss << 10 << " " << 20;
string res = ss.str(); // "10 20"
```

 Copy code

(This uses `stringstream` as **output stream**)

## 7 Types of String Streams

Type	Purpose
<code>stringstream</code>	read + write
<code>istringstream</code>	read only
<code>ostringstream</code>	write only

Example:

```
cpp

istringstream iss(s); // input only
ostringstream oss;    // output only
```

 Copy code

## 8 Important Properties (Interview / CP)

- ✓ Automatically splits by whitespace
- ✓ Safe and clean
- ✓ Slower than manual parsing (but fine for most problems)
- ✓ Extremely useful in **string problems**

---

## 9 Why `stringstream` is PERFECT for your problem

Your problem needs:

- word extraction
- space handling
- clean logic

So this part:

cpp

 Copy code

```
stringstream ss(s);  
while (ss >> w) words.push_back(w);
```

is 100% optimal and correct

---

## 10 Summary (Remember This)

`stringstream` lets you treat a string like `cin`.

It reads tokens separated by whitespace automatically.

Best for string parsing problems in CP & LeetCode.

---

If you want next:

- how to do same WITHOUT `stringstream`
- time complexity comparison
- edge cases ( " ", "a", " a b " )

Just tell me.

     ...



