Tokenizing a string in C++

Difficulty Level : Medium • Last Updated : 02 Jan, 2023

Tokenizing a string denotes splitting a string with respect to some delimiter(s). There are many ways to tokenize a string. In this article four of them are explained:

Using stringstream

A **stringstream** associates a string object with a stream allowing you to read from the string as if it were a stream.

Below is the C++ implementation :

C++

```
// Tokenizing a string using stringstream
#include <bits/stdc++.h>

using namespace std;

int main()
{

    string line = "GeeksForGeeks is a must try";

    // Vector of string to save tokens
    vector <string> tokens;

    // stringstream class check1
    stringstream check1(line);

    string intermediate;

    // Tokenizing w.r.t. space ' '
    while(getline(check1, intermediate, ' '))
```

```
    {
        tokens.push_back(intermediate);
    }

    // Printing the token vector
    for(int i = 0; i < tokens.size(); i++)
        cout << tokens[i] << '\n';
}
```

Output


```
GeeksForGeeks
is
a
must
try
```

Time Complexity: O(n) where n is the length of string.
Auxiliary Space: O(n-d) where n is the length of string and d is the number of delimiters.

Using strtok()

```
// Splits str[] according to given delimiters.
// and returns next token. It needs to be called
// in a loop to get all tokens. It returns NULL
// when there are no more tokens.
char * strtok(char str[], const char *delims);
```

Below is the C++ implementation :

C++

```
// C/C++ program for splitting a string
// using strtok()
#include <stdio.h>
#include <string.h>
```

```
int main()
{
    char str[] = "Geeks-for-Geeks";

    // Returns first token
    char *token = strtok(str, "-");

    // Keep printing tokens while one of the
    // delimiters present in str[].
    while (token != NULL)
    {
        printf("%s\n", token);
        token = strtok(NULL, "-");
    }

    return 0;
}
```

Output

```
Geeks
for
Geeks
```

Time Complexity: O(n) where n is the length of string.

Auxiliary Space: O(1).

Another Example of strtok() :

C

```
// C code to demonstrate working of
// strtok
#include <string.h>
#include <stdio.h>

// Driver function
int main()
{
 // Declaration of string
    char gfg[100] = " Geeks - for - geeks - Contribute";

    // Declaration of delimiter
    const char s[4] = "-";
    char* tok;

    // Use of strtok
    // get first token
    tok = strtok(gfg, s);

    // Checks for delimiter
```

```
    while (tok != 0) {
        printf(" %s\n", tok);

        // Use of strtok
        // go through other tokens
        tok = strtok(0, s);
    }

    return (0);
}
```

Output

```
Geeks
for
geeks
Contribute
```

Time Complexity: O(n) where n is the length of string.
Auxiliary Space: O(1).

Using strtok_r()

Just like strtok() function in C, **strtok_r()** does the same task of parsing a string into a sequence of tokens. strtok_r() is a reentrant version of strtok().

There are two ways we can call strtok_r()

```
// The third argument saveptr is a pointer to a char *
// variable that is used internally by strtok_r() in
// order to maintain context between successive calls
// that parse the same string.
char *strtok_r(char *str, const char *delim, char **saveptr);
```

Below is a simple C++ program to show the use of strtok_r() :

C++

```
// C/C++ program to demonstrate working of strtok_r()
// by splitting string based on space character.
#include<stdio.h>
#include<string.h>

int main()
{
    char str[] = "Geeks for Geeks";
    char *token;
```

```
    char *rest = str;

    while ((token = strtok_r(rest, " ", &rest)))
        printf("%s\n", token);

    return(0);
}
```

Output

```
Geeks
for
Geeks
```

Time Complexity: O(n) where n is the length of string.
Auxiliary Space: O(1).

Using std::sregex_token_iterator

In this method the tokenization is done on the basis of regex matches. Better for use cases when multiple delimiters are needed.

Below is a simple C++ program to show the use of std::sregex_token_iterator:

C++

```
// CPP program for above approach
#include <iostream>
#include <regex>
#include <string>
#include <vector>

/**
 * @brief Tokenize the given vector
   according to the regex
 * and remove the empty tokens.
 *
 * @param str
 * @param re
 * @return std::vector<std::string>
 */
std::vector<std::string> tokenize(
                    const std::string str,
                         const std::regex re)
{
    std::sregex_token_iterator it{ str.begin(),
                              str.end(), re, -1 };
    std::vector<std::string> tokenized{ it, {} };

    // Additional check to remove empty strings
```

```
    tokenized.erase(
        std::remove_if(tokenized.begin(),
                       tokenized.end(),
                    [](std::string const& s) {
                        return s.size() == 0;
                    }),
        tokenized.end());

    return tokenized;
}

// Driver Code
int main()
{
    const std::string str = "Break string
                   a,spaces,and,commas";
    const std::regex re(R"([\s|,]+)");

    // Function Call
    const std::vector<std::string> tokenized =
                          tokenize(str, re);

    for (std::string token : tokenized)
        std::cout << token << std::endl;
    return 0;
}
```

Output

```
Break
string
a
spaces
and
commas
```

Time Complexity: O(n * d) where n is the length of string and d is the number of delimiters.
Auxiliary Space: O(n)

75

Related Articles

1. Count characters of a string which when removed individually makes the string equal to another string

2. Generate string by incrementing character of given string by number present at corresponding index of second string