Split a string in C++?

What's the most elegant way to split a string in C++? The string can be assumed to be composed of words separated by whitespace.

(Note that I'm not interested in C string functions or that kind of character manipulation/access. Also, please give precedence to elegance over efficiency in your answer.)

The best solution I have right now is:

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

int main()
{
    string s("Somewhere down the road");
    istringstream iss(s);

    do
    {
        string sub;
        iss >> sub;
        cout << "Substring: " << sub << endl;
    } while (iss);

}
```

c++ split c++-faq stdstring

edited Oct 31 '14 at 13:07

community wiki
14 revs, 7 users 59%
Ashwin

330 Dude... Elegance is just a fancy way to say "efficiency-that-looks-pretty" in my book. Don't shy away from using C functions and quick methods to accomplish anything just because it is not contained within a template ;) – nlaq Oct 25 '08 at 9:04

17 Your code won't compile (sometimes the pertinent variable is called "subs", sometimes "substr") but there's a more serious off-by-one problem: it will always try to output one more token than actually exists because you only test iss *after* printing the token. – j_random_hacker Aug 24 '09 at 8:57

5 `while (iss) { string subs; iss >> subs; cout << "Substring: " << sub << endl; }` – Eduardo León Sep 29 '09 at 15:47

21 How about some of the examples from the following: codeproject.com/KB/recipes/Tokenizer.aspx They are very efficient and somewhat elegant. – Matthieu N. Sep 15 '10 at 3:21

9 @Eduardo: that's wrong too... you need to test iss between trying to stream another value and using that value, i.e. `string sub; while (iss >> sub) cout << "Substring: " << sub << '\n';` – Tony D Apr 11 '12 at 2:24

55 Answers

1 2 next

FWIW, here's another way to extract tokens from an input string, relying only on standard library facilities. It's an example of the power and elegance behind the design of the STL.

```
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>

int main() {
    using namespace std;
    string sentence = "And I feel fine...";
    istringstream iss(sentence);
    copy(istream_iterator<string>(iss),
         istream_iterator<string>(),
         ostream_iterator<string>(cout, "\n"));
}
```

Instead of copying the extracted tokens to an output stream, one could insert them into a container, using the same generic copy algorithm.

```
vector<string> tokens;
copy(istream_iterator<string>(iss),
     istream_iterator<string>(),
     back_inserter(tokens));
```

... or create the vector directly:

```
vector<string> tokens{istream_iterator<string>{iss},
                      istream_iterator<string>{}};
```

edited Aug 22 '14 at 8:20

community wiki
6 revs, 6 users 76%
Zunino

69 Is it possible to specify a delimiter for this? Like for instance splitting on commas? – l3dx Aug 6 '09 at 11:49

19 So can you split on comma? – graham.reeds Jul 22 '10 at 9:09

406 This is a poor solution as it doesn't take any other delimiter, therefore not scalable and not maintable. – Student T Jan 10 '11 at 3:57

40 Can't deal with varying delimiters and is VERY VERY inefficient. in short DO NOT use this code anywhere. – Xander Tulip Mar 19 '12 at 0:17

17 Actually, this *can* work just fine with other delimiters (though doing some is somewhat ugly). You create a ctype facet that classifies the desired delimiters as whitespace, create a locale containing that facet, then imbue the stringstream with that locale before extracting strings. – Jerry Coffin Dec 19 '12 at 20:30

I use this to split string by a delim. The first puts the results in a pre-constructed vector, the second returns a new vector.

```
#include <string>
#include <sstream>
#include <vector>

std::vector<std::string> &split(const std::string &s, char delim, std::vector<std::string>
&elems) {
    std::stringstream ss(s);
    std::string item;
    while (std::getline(ss, item, delim)) {
        elems.push_back(item);
    }
    return elems;
}


std::vector<std::string> split(const std::string &s, char delim) {
    std::vector<std::string> elems;
    split(s, delim, elems);
    return elems;
}
```

EDIT: Note that this solution does not skip empty tokens, so the following will find 4 items, one of which is empty:

```
std::vector<std::string> x = split("one:two::three", ':');
```

edited Mar 28 at 23:13

community wiki
7 revs, 4 users 79%
Evan Teran

41 This hits the sweet spot for me - standard libraries, short, and lets me specify my delimiters. Thanks! – tfinniga Mar 29 '10 at 16:39

76 @Paul Lammertsma: Correction: Don't forget to include `string`, `sstream` and `vector`. – David Johnstone Mar 3 '11 at 3:58

6 This should be `split(s, delim, elems); return elems;`, to enable return value optimization. – Simon Buchan Oct 2 '11 at 23:59

25 In order to avoid it skipping empty tokens, do an `empty()` check: `if (!item.empty()) elems.push_back(item)` – 0x499602D2 Nov 9 '13 at 22:33

10 +1 IMHO this should have been the accepted answer. It isn't a silver bullet, but is considerably more adaptable than the accepted solution. – WhozCraig Dec 5 '13 at 20:09

Since everybody is already using Boost:

```
#include <boost/algorithm/string.hpp>
std::vector<std::string> strs;
boost::split(strs, "string to split", boost::is_any_of("\t "));
```

I bet this is much faster than the `stringstream` solution. And since this is a generic template function it can be used to split other types of strings (wchar, etc. or UTF-8) using all kinds of delimiters.

See the documentation for details.

edited Aug 5 '10 at 18:32

community wiki
2 revs, 2 users 73%
ididak

21 Speed is irrelevant here, as both of these cases are much slower than a strtok-like function. – Tom Mar 1 '09 at 16:51

21 And for those who don't already have boost... bcp copies over 1,000 files for this :) – romkyns Jun 9 '10 at 20:12

45 strtok is a trap. its thread unsafe. – tuxSlayer Apr 23 '11 at 3:30

375 Everybody is not using Boost. – Isaac May 20 '11 at 16:07

55 @Isaac Everybody who wants to save some time is using boost. – Ian Jan 20 '12 at 21:53

```
#include <vector>
#include <string>
#include <sstream>

using namespace std;

int main()
{
    string str("Split me by whitespaces");
    string buf; // Have a buffer string
    stringstream ss(str); // Insert the string into a stream

    vector<string> tokens; // Create vector to hold our words

    while (ss >> buf)
        tokens.push_back(buf);
}
```

answered Mar 6 '11 at 5:52

community wiki
kev

11 too bad it only splits on spaces `' '`... – Offirmo Jan 31 '13 at 18:47

22 It doesn't get more upvotes because with all this algorithm-iterator style we forgot about the beauty of the simple 'while' loop. – anxieux Feb 28 '13 at 7:09

4 @anxieux there's no beauty in direct imperative loops whatsoever. – Bartek Banachewicz May 15 '14 at 14:43

3 @Bartek Banachewicz: beauty is in the eye of the beholder. – Andrey Rekalo Jan 1 at 12:09

For those with whom it does not sit well to sacrifice all efficiency for code size and see "efficient" as a type of elegance, the following should hit a sweet spot (and I think the template container class is an awesomely elegant addition.):

```
template < class ContainerT >
void tokenize(const std::string& str, ContainerT& tokens,
              const std::string& delimiters = " ", bool trimEmpty = false)
{
   std::string::size_type pos, lastPos = 0;

   using value_type = typename ContainerT::value_type;
   using size_type  = typename ContainerT::size_type;

   while(true)
   {
      pos = str.find_first_of(delimiters, lastPos);
      if(pos == std::string::npos)
      {
         pos = str.length();

         if(pos != lastPos || !trimEmpty)
            tokens.push_back(value_type(str.data()+lastPos,
                  (size_type)pos-lastPos ));

         break;
      }
      else
      {
         if(pos != lastPos || !trimEmpty)
            tokens.push_back(value_type(str.data()+lastPos,
                  (size_type)pos-lastPos ));
      }

      lastPos = pos + 1;
   }
}
```

I usually choose to use `std::vector<std::string>` types as my second parameter (`ContainerT`)... but `list<>` is way faster than `vector<>` for when direct access is not needed, and you can even create your own string class and use something like `std::list<subString>` where `subString` does not do any copies for incredible speed increases.

It's more than double as fast as the fastest tokenize on this page and almost 5 times faster than some others. Also with the perfect parameter types you can eliminate all string and list copies for additional speed increases.

Additionally it does not do the (extremely inefficient) return of result, but rather it passes the tokens as a reference, thus also allowing you to build up tokens using multiple calls if you so wished.

Lastly it allows you to specify whether to trim empty tokens from the results via a last optional parameter.

All it needs is `std::string` ... the rest are optional. It does not use streams or the boost library, but is flexible enough to be able to accept some of these foreign types naturally.

edited Sep 25 '14 at 7:07

community wiki
10 revs, 4 users 88%
Marius

4 I'm quite a fan of this, but for g++ (and probably good practice) anyone using this will want typedefs and typenames: `typedef ContainerT Base; typedef typename Base::value_type ValueType; typedef typename ValueType::size_type SizeType;` Then to substitute out the value_type and size_types accordingly. – aws Nov 28 '11 at 21:41

5 For those of us for whom the template stuff and the first comment are completely foreign, a usage example cmplete with required includes would be lovely. – Wes Miller Aug 17 '12 at 11:51

2 Ahh well, I figured it out. I put the C++ lines from aws' comment inside the function body of tokenize(), then edited the tokens.push_back() lines to change the ContainerT::value_type to just ValueType and changed (ContainerT::value_type::size_type) to (SizeType). Fixed the bits g++ had been whining about. Just invoke it as tokenize(some_string, some_vector); – Wes Miller Aug 17 '12 at 14:23

```
string line = "a line of text to iterate through";
string word;

istringstream iss(line, istringstream::in);

while( iss >> word )
{

...

}
```

This is my favourite way to iterate through a string. You can do what you want per word.

edited Jul 6 '12 at 21:44

community wiki

3 revs
gnomed

3 if you declare word as a char it will iterate over every non-whitespace character. It's simple enough to try: `stringstream ss("Hello World, this is*@#&$(@ a string"); char c; while(ss >> c) cout << c;` – Wayne Werner Aug 4 '10 at 18:03

This is similar to Stack Overflow question *How do I tokenize a string in C++?*.

```
#include <iostream>
#include <string>
#include <boost/foreach.hpp>
#include <boost/tokenizer.hpp>

using namespace std;
using namespace boost;

int main(int argc, char** argv)
{
    string text = "token  test\tstring";

    char_separator<char> sep(" \t");
    tokenizer<char_separator<char> > tokens(text, sep);
    BOOST_FOREACH(string t, tokens)
    {
        cout << t << "." << endl;
    }
}
```

edited Feb 21 '14 at 2:02

community wiki
3 revs, 3 users 77%
Ferruccio

4 Thanks for pointing that out. I didn't know this operation was called tokenizing, so it never occurred to me to search for that term :-) – Ashwin Oct 27 '08 at 2:48

I like the following because it puts the results into a vector, supports a string as a delim and gives control over keeping empty values. But, it doesn't look as good then.

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

vector<string> split(const string& s, const string& delim, const bool keep_empty = true) {
    vector<string> result;
    if (delim.empty()) {
        result.push_back(s);
        return result;
    }
    string::const_iterator substart = s.begin(), subend;
    while (true) {
        subend = search(substart, s.end(), delim.begin(), delim.end());
        string temp(substart, subend);
        if (keep_empty || !temp.empty()) {
            result.push_back(temp);
        }
        if (subend == s.end()) {
            break;
        }
        substart = subend + delim.size();
    }
    return result;
}

int main() {
    const vector<string> words = split("So close no matter how far", " ");
    copy(words.begin(), words.end(), ostream_iterator<string>(cout, "\n"));
}
```

Of course, Boost has a split() that works partially like that. And, if by 'white-space', you really do mean any type of white-space, using Boost's split with is_any_of() works great.

edited Oct 25 '08 at 10:17

community wiki
2 revs

Shadow2531

Here's another solution. It's compact and reasonably efficient:

```
void split(vector<string> &tokens, const string &text, char sep) {
  int start = 0, end = 0;
  while ((end = text.find(sep, start)) != string::npos) {
    tokens.push_back(text.substr(start, end - start));
    start = end + 1;
  }
  tokens.push_back(text.substr(start));
}
```

It can easily be templatised to handle string separators, wide strings, etc.

@sehe has a more generalised version of this function here.

edited Feb 6 '12 at 18:54

community wiki
3 revs, 2 users 62%
Alec Thomas

3 The first version is simple and gets the job done perfectly. The only change I would made would be to return the result directly, instead of passing it as a parameter. – gregschlom Jan 19 '12 at 2:25

2 @AlecThomas: Even before C++11, wouldn't most compilers optimise away the return copy via NRVO? (+1 anyway; very succinct) – Marcelo Cantos Aug 17 '13 at 11:54

Yet another flexible and fast way

```
template<typename Operator>
void tokenize(Operator& op, const char* input, const char* delimiters) {
  const char* s = input;
  const char* e = s;
  while (*e != 0) {
    e = s;
    while (*e != 0 && strchr(delimiters, *e) == 0) ++e;
    if (e - s > 0) {
      op(s, e - s);
    }
    s = e + 1;
  }
}
```

To use it with a vector of strings (Edit: Since someone pointed out not to inherit STL classes... hrmf ;)) :

```
template<class ContainerType>
class Appender {
public:
  Appender(ContainerType& container) : container_(container) {;}
  void operator() (const char* s, unsigned length) {
    container_.push_back(std::string(s,length));
  }
private:
  ContainerType& container_;
};

std::vector<std::string> strVector;
Appender v(strVector);
tokenize(v, "A number of words to be tokenized", " \t");
```

That's it! And that's just one way to use the tokenizer, like how to just count words:

```
class WordCounter {
public:
  WordCounter() : noOfWords(0) {}
  void operator() (const char*, unsigned) {
    ++noOfWords;
  }
  unsigned noOfWords;
};

WordCounter wc;
tokenize(wc, "A number of words to be counted", " \t");
ASSERT( wc.noOfWords == 7 );
```

Limited by imagination ;)

community wiki

edited Sep 11 '13 at 8:11

2 revs
Robert

The STL does not have such a method available already.

However, you can either use C's strtok function by using the string.c_str() member, or you can write your own. Here is a code sample I found after a quick google search ("STL string split"):

```
void Tokenize(const string& str,
                      vector<string>& tokens,
                      const string& delimiters = " ")
{
    // Skip delimiters at beginning.
    string::size_type lastPos = str.find_first_not_of(delimiters, 0);
    // Find first "non-delimiter".
    string::size_type pos     = str.find_first_of(delimiters, lastPos);

    while (string::npos != pos || string::npos != lastPos)
    {
        // Found a token, add it to the vector.
        tokens.push_back(str.substr(lastPos, pos - lastPos));
        // Skip delimiters.  Note the "not_of"
        lastPos = str.find_first_not_of(delimiters, pos);
        // Find next "non-delimiter"
        pos = str.find_first_of(delimiters, lastPos);
    }
}
```

Taken from: http://oopweb.com/CPP/Documents/CPPHOWTO/Volume/C++Programming-HOWTO-7.html

If you have questions about the code sample, leave a comment and I will explain.

And just because it does not implement a typedef called iterator or overload the << operator does not mean it is bad code. I use the C functions quite frequently. For example, printf and scanf both are faster then cin and cout (significantly), the fopen syntax is a lot more friendly for binary types, and they also tend to produce smaller EXEs.

Don't get sold on this "Elegance over performance" deal.

edited Oct 25 '08 at 9:14

community wiki
2 revs
nlaq

5 @Nelson LaQuet: Let me guess: Because strtok is not reentrant? – paercebal Oct 25 '08 at 9:52

23 @Nelson don't *ever* pass string.c_str() to strtok! strtok trashes the input string (inserts '\0' chars to replace each foudn delimiter) and c_str() returns a non-modifiable string. – Evan Teran Oct 25 '08 at 18:19

2 @Nelson: That array needs to be of size str.size() + 1 in your last comment. But I agree with your thesis that it's silly to avoid C functions for "aesthetic" reasons. – j_random_hacker Aug 24 '09 at 9:08

Here is a split function that:

- is generic
- uses standard C++ (no boost)
- accepts multiple delimiters
- ignores empty tokens (can easily be changed)

```
template<typename T>
vector<T>
split(const T & str, const T & delimiters) {
    vector<T> v;
    T::size_type start = 0;
    auto pos = str.find_first_of(delimiters, start);
    while(pos != T::npos) {
        if(pos != start) // ignore empty tokens
            v.emplace_back(str, start, pos - start);
        start = pos + 1;
        pos = str.find_first_of(delimiters, start);
    }
    if(start < str.length()) // ignore trailing delimiter
        v.emplace_back(str, start, str.length() - start); // add what's left of the string
    return v;
}
```

Example usage:

```
    vector<string> v = split<string>("Hello, there; World", ";,");
    vector<wstring> v = split<wstring>(L"Hello, there; World", L";,");
```

edited Mar 21 '14 at 20:00

community wiki
5 revs
Marco M.

1 @XanderTulip: I assume you are referring to it returning the vector by value. The Return-Value-Optimization (RVO, google it) should take care of this. Also in C++11 you could return by move reference. – Joseph Garvin May 7 '12 at 13:56

2 This can actually be optimized further: instead of .push_back(str.substr(...)) one can use .emplace_back(str, start, pos - start). This way the string object is constructed in the container and thus we avoid a move operation + other shenanigans done by the .substr function. – Mihai Bişog Sep 5 '12 at 13:50

If you like to use boost, but want to use a whole string as delimiter (instead of single characters as in most of the previously proposed solutions), you can use the `boost_split_iterator`.

Example code including convenient template:

```
#include <iostream>
#include <vector>
#include <boost/algorithm/string.hpp>

template<typename _OutputIterator>
inline void split(
    const std::string& str,
    const std::string& delim,
    _OutputIterator result)
{
    using namespace boost::algorithm;
    typedef split_iterator<std::string::const_iterator> It;

    for(It iter=make_split_iterator(str, first_finder(delim, is_equal()));
            iter!=It();
            ++iter)
    {
        *(result++) = boost::copy_range<std::string>(*iter);
    }
}

int main(int argc, char* argv[])
{
    using namespace std;

    vector<string> splitted;
    split("HelloFOOworldFOO!", "FOO", back_inserter(splitted));

    // or directly to console, for example
    split("HelloFOOworldFOO!", "FOO", ostream_iterator<string>(cout, "\n"));
    return 0;
}
```

edited Feb 9 '12 at 9:32

community wiki
3 revs, 2 users 71%
zerm

I have a 2 lines solution to this problem:

```
char sep = ' ';
std::string s="1 This is an example";

for(size_t p=0, q=0; p!=s.npos; p=q)
  std::cout << s.substr(p+(p!=0), (q=s.find(sep, p+1))-p-(p!=0)) << std::endl;
```

Then instead of printing you can put it in a vector.

edited Jan 15 '13 at 0:12

community wiki
2 revs, 2 users 94%
rhomu

There is a function named `strtok`.

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

vector<string> split(char* str,const char* delim)
{
    char* saveptr;
    char* token = strtok_r(str,delim,&saveptr);

    vector<string> result;

    while(token != NULL)
    {
        result.push_back(token);
        token = strtok_r(NULL,delim,&saveptr);
    }
    return result;
}
```

edited May 2 '14 at 14:49

community wiki
3 revs, 2 users 91%
Pratik Deoghare

9 Because it stores the char pointer from the first call in a static variable, so that on the subsequent calls when NULL is passed, it remembers what pointer should be used. If a second thread calls `strtok` when another thread is still processing, this char pointer will be overwritten, and both threads will then have incorrect results. mkssoftware.com/docs/man3/strtok.3.asp – Kevin Panko Jun 14 '10 at 17:27

1 as mentioned before strtok is unsafe and even in C strtok_r is recommended for use – systemsfault Jul 6 '10 at 12:17

2 strtok_r can be used if you are in a section of code that may be accessed. this is the *only* solution of all of the above that isn't "line noise", and is a testament to what, exactly, is wrong with c++ – Erik Aronesty Oct 10 '11 at 18:04

Using stringstream as you have works perfectly fine, and do exactly what you wanted. If you're just looking for different way of doing things though, you can use find/find_first_of and substring.

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

int main()
{
    std::string s("Somewhere down the road");

    std::string::size_type prev_pos = 0, pos = 0;
    while( (pos = s.find(' ', pos)) != std::string::npos )
    {
        std::string substring( s.substr(prev_pos, pos-prev_pos) );

        std::cout << substring << '\n';

        prev_pos = ++pos;
    }
    std::string substring( s.substr(prev_pos, pos-prev_pos) ); // Last word
    std::cout << substring << '\n';
}
```

answered Oct 25 '08 at 9:28

community wiki
KTC

Heres a regex solution that only uses the standard regex library. (I'm a little rusty, so there may be a few syntax errors, but this is at least the general idea)

```
#include <regex.h>
#include <string.h>
#include <vector.h>

using namespace std;

vector<string> split(string s){
    regex r ("\\w+"); //regex matches whole words, (greedy, so no fragment words)
    regex_iterator<string::iterator> rit ( s.begin(), s.end(), r );
    regex_iterator<string::iterator> rend; //iterators to iterate thru words
    vector<string> result<regex_iterator>(rit, rend);
    return result;  //iterates through the matches to fill the vector
}
```

answered Oct 29 '12 at 16:15

community wiki

AJMansfield

So far I used the one in Boost, but I needed something that doesn't depends on it, so I came to this:

```
static void Split(std::vector<std::string>& lst, const std::string& input, const
std::string& separators, bool remove_empty = true)
{
    std::ostringstream word;
    for (size_t n = 0; n < input.size(); ++n)
    {
        if (std::string::npos == separators.find(input[n]))
            word << input[n];
        else
        {
            if (!word.str().empty() || !remove_empty)
                lst.push_back(word.str());
            word.str("");
        }
    }
    if (!word.str().empty() || !remove_empty)
        lst.push_back(word.str());
}
```

A good point is that in `separators` you can pass more than one character.

edited May 22 '11 at 23:02

community wiki
3 revs, 2 users 64%
Goran

I made this because I needed an easy way to split strings and c-based strings... Hopefully someone else can find it useful as well. Also it doesn't rely on tokens and you can use fields as delimiters, which is another key I needed.

I'm sure there's improvements that can be made to even further improve its elegance and please do by all means

StringSplitter.hpp:

```
#include <vector>
#include <iostream>
#include <string.h>
using namespace std;
class StringSplit
{
private:
 void copy_fragment(char*,char*,char*);
 void copy_fragment(char*,char*,char);
 bool match_fragment(char*,char*,int);
 int untilnextdelim(char*,char);
 int untilnextdelim(char*,char*);
 void assimilate(char*,char);
 void assimilate(char*,char*);
 bool string_contains(char*,char*);
 long calc_string_size(char*);
 void copy_string(char*,char*);

public:
 vector<char*> split_cstr(char);
 vector<char*> split_cstr(char*);
 vector<string> split_string(char);
 vector<string> split_string(char*);
 char* String;
 bool do_string;
 bool keep_empty;
 vector<char*> Container;
 vector<string> ContainerS;
StringSplit(char * in)
{
String = in;
}
StringSplit(string in)
{
   size_t len = calc_string_size((char*)in.c_str());
   String = new char[len+1];
   memset(String,0,len+1);
   copy_string(String,(char*)in.c_str());
   do_string = true;
}
~StringSplit()
```

```
{
  for(int i = 0;i<Container.size();i++)
    if(Container[i]!=NULL)
        delete[] Container[i];
  if(do_string)
    delete[] String;
}
```

};

StringSplitter.cpp:

```
#include <string.h>
#include <iostream>
#include <vector>
#include "StringSplit.hpp"
using namespace std;

void StringSplit::assimilate(char*src,char delim)
{
    int until = untilnextdelim(src,delim);
    if(until>0)
    {
        char * temp = new char[until+1];
        memset(temp,0,until+1);
        copy_fragment(temp,src,delim);
        if(keep_empty||*temp!=0)
        {
            if(!do_string)
            {
                Container.push_back(temp);
            }
            else
            {
                string x = temp;
                ContainerS.push_back(x);
            }

        }
        else
        {
            delete[] temp;
        }

    }

}
void StringSplit::assimilate(char*src,char* delim)
{
    int until = untilnextdelim(src,delim);
    if(until>0)
    {
        char * temp = new char[until+1];
        memset(temp,0,until+1);
        copy_fragment(temp,src,delim);
        if(keep_empty||*temp!=0)
        {
            if(!do_string)
            {
                Container.push_back(temp);
            }
            else
            {
                string x = temp;
                ContainerS.push_back(x);
            }

        }
        else
        {
            delete[] temp;
        }

    }

}
long StringSplit::calc_string_size(char* _in)
{
long i = 0;
while(*_in++)
    i++;
return i;
}
bool StringSplit::string_contains(char*haystack,char*needle)
{
size_t len = calc_string_size(needle);
size_t lenh = calc_string_size(haystack);
while(lenh--)
    if(match_fragment(haystack+lenh,needle,len))
```

```
        return true;
return false;
}
bool StringSplit::match_fragment(char* _src, char*cmp,int len)
{
    while(len--)
    if(*(_src+len)!=*(cmp+len))
        return false;
return true;
}
int StringSplit::untilnextdelim(char* _in, char delim)
{
size_t len = calc_string_size(_in);
if(*_in==delim)
{
    _in += 1;
    return len-1;
}
int c = 0;
while(*(_in+c)!=delim&&c<len)
        c++;
return c;
}
int StringSplit::untilnextdelim(char* _in, char* delim)
{
int s = calc_string_size(delim);
int c = 1 + s;
if(!string_contains(_in,delim))
{
    return calc_string_size(_in);
}
else if(match_fragment(_in,delim,s))
{
    _in+=s;
    return calc_string_size(_in);
}
while(!match_fragment(_in+c,delim,s))
        c++;
return c;
}
void StringSplit::copy_fragment(char*dest,char*src,char delim)
{
        if(*src==delim)
            src++;
        int c = 0;
        while(*(src+c) != delim && *(src+c))
        {
            *(dest+c) = *(src+c);
            c++;
        }
        *(dest+c)=0;
}
void StringSplit::copy_string(char*dest,char*src)
{
        int i = 0;
        while(*(src+i))
        {
            *(dest+i) = *(src+i);
            i++;
        }
}
void StringSplit::copy_fragment(char*dest,char*src,char*delim)
{
        size_t len = calc_string_size(delim);
        size_t lens = calc_string_size(src);
        if(match_fragment(src,delim,len))
        {
            src += len;
            lens -= len;
        }
        int c = 0;
        while(!match_fragment(src+c,delim,len) && c<lens)
        {
            *(dest+c) = *(src+c);
            c++;
        }
        *(dest+c)=0;
}

vector<char*> StringSplit::split_cstr(char Delimiter)
{
int i = 0;
while(*String)
{
    if(*String!=Delimiter && i == 0)
        assimilate(String,Delimiter);
    if(*String==Delimiter)
        assimilate(String,Delimiter);
    i++;
    String++;
}
String -= i;
delete[] String;
return Container;
}
vector<string> StringSplit::split_string(char Delimiter)
{
do_string = true;
int i = 0;
while(*String)
{
    if(*String!=Delimiter && i == 0)
        assimilate(String,Delimiter);
    if(*String==Delimiter)
        assimilate(String,Delimiter);
    i++;
    String++;
}
String -= i;
delete[] String;
return ContainerS;
}
vector<char*> StringSplit::split_cstr(char* Delimiter)
{
int i = 0;
size_t LenDelim = calc_string_size(Delimiter);
while(*String)
{
        if(!match_fragment(String,Delimiter,LenDelim) && i == 0)
            assimilate(String,Delimiter);
        if(match_fragment(String,Delimiter,LenDelim))
            assimilate(String,Delimiter);
    i++;
    String++;

}
String -= i;
delete[] String;
return Container;
}
vector<string> StringSplit::split_string(char* Delimiter)
{
do_string = true;
int i = 0;
size_t LenDelim = calc_string_size(Delimiter);
while(*String)
{
        if(!match_fragment(String,Delimiter,LenDelim) && i == 0)
            assimilate(String,Delimiter);
        if(match_fragment(String,Delimiter,LenDelim))
            assimilate(String,Delimiter);
    i++;
    String++;

}
String -= i;
delete[] String;
return ContainerS;
}
```

Examples:

```
int main(int argc, char*argv[])
{
StringSplit ss = "This:CUT:is:CUT:an:CUT:example:CUT:cstring";
vector<char*> Split = ss.split_cstr(":CUT:");
for(int i = 0;i<Split.size();i++)
    cout << Split[i] << endl;
return 0;
}
```

Will output:

This
is
an
example
cstring

```
int main(int argc, char*argv[])
{
StringSplit ss = "This:is:an:example:cstring";
vector<char*> Split = ss.split_cstr(':');
for(int i = 0;i<Split.size();i++)
    cout << Split[i] << endl;
return 0;
}
```

```
int main(int argc, char*argv[])
{
string mystring = "This[SPLIT]is[SPLIT]an[SPLIT]example[SPLIT]string"
StringSplit ss = mystring;
vector<string> Split = ss.split_string("[SPLIT]");
for(int i = 0;i<Split.size();i++)
    cout << Split[i] << endl;
return 0;
}

int main(int argc, char*argv[])
{
string mystring = "This|is|an|example|string"
StringSplit ss = mystring;
vector<string> Split = ss.split_string('|');
for(int i = 0;i<Split.size();i++)
    cout << Split[i] << endl;
return 0;
}
```

To keep empty entries (by default empties will be excluded):

```
StringSplit ss = mystring;
ss.keep_empty = true;
vector<string> Split = ss.split_string(":DELIM:");
```

The goal was to make it similar to C#'s Split() method where splitting a string is as easy as:

```
String[] Split = "Hey:cut:what's:cut:your:cut:name?".Split(new[]
{":cut:"},StringSplitOptions.None);
foreach(String X in Split)
Write(X);
```

I hope someone else can find this as useful as I do

answered Aug 31 '12 at 19:11 community wiki Steve Dell

The streamstream can be convenient if you need to parse the string by non-space symbols:

```
string s = "Name:JAck; Spouse:Susan; ...";
string dummy, name, spouse;

istringstream iss(s);
getline(iss, dummy, ':');
getline(iss, name, ';');
getline(iss, dummy, ':');
getline(iss, spouse, ';')
```

answered Aug 12 '11 at 19:05 community wiki lukmac

Here's another way of doing it..

```
void split_string(string text,vector<string>& words)
{
  int i=0;
  char ch;
  string word;

  while(ch=text[i++])
  {
    if (isspace(ch))
    {
      if (!word.empty())
      {
        words.push_back(word);
      }
      word = "";
    }
    else
    {
      word += ch;
    }
  }
  if (!word.empty())
  {
    words.push_back(word);
  }
}
```

edited Jan 8 '10 at 3:27

community wiki
2 revs
user246110

I like to use the boost/regex methods for this task since they provide maximum flexibility for specifying the splitting criteria.

```
#include <iostream>
#include <string>
#include <boost/regex.hpp>

int main() {
    std::string line("A:::line::to:split");
    const boost::regex re(":+"); // one or more colons

    // -1 means find inverse matches aka split
    boost::sregex_token_iterator tokens(line.begin(),line.end(),re,-1);
    boost::sregex_token_iterator end;

    for (; tokens != end; ++tokens)
        std::cout << *tokens << std::endl;
}
```

answered Jun 12 '11 at 9:25

community wiki
Marty B

What about this:

```
#include <string>
#include <vector>

using namespace std;

vector<string> split(string str, const char delim) {
    vector<string> v;
    string tmp;

    for(string::const_iterator i; i = str.begin(); i <= str.end(); ++i) {
        if(*i != delim && i != str.end()) {
            tmp += *i;
        } else {
            v.push_back(tmp);
            tmp = "";
        }
    }

    return v;
}
```

edited Dec 19 '12 at 22:05

community wiki
3 revs, 3 users 89%
gibbz

I've rolled my own using strtok and used boost to split a string. The best method I have found is the C++ String Toolkit Library. It is incredibly flexible and fast.

```
#include <iostream>
#include <vector>
#include <string>
#include <strtk.hpp>

const char *whitespace  = " \t\r\n\f";
const char *whitespace_and_punctuation  = " \t\r\n\f;,=";

int main()
{
    {   // normal parsing of a string into a vector of strings
        std::string s("Somewhere down the road");
        std::vector<std::string> result;
        if( strtk::parse( s, whitespace, result ) )
        {
            for(size_t i = 0; i < result.size(); ++i )
            std::cout << result[i] << std::endl;
        }
    }

    {  // parsing a string into a vector of floats with other separators
        // besides spaces

        std::string s("3.0, 3.14; 4.0");
        std::vector<float> values;
        if( strtk::parse( s, whitespace_and_punctuation, values ) )
        {
            for(size_t i = 0; i < values.size(); ++i )
                std::cout << values[i] << std::endl;
        }
    }

    {  // parsing a string into specific variables

        std::string s("angle = 45; radius = 9.9");
        std::string w1, w2;
        float v1, v2;
        if( strtk::parse( s, whitespace_and_punctuation, w1, v1, w2, v2) )
        {
            std::cout << "word " << w1 << ", value " << v1 << std::endl;
            std::cout << "word " << w2 << ", value " << v2 << std::endl;
        }
    }

    return 0;
}
```

The toolkit has much more flexibility than this simple example shows but its utility in parsing a string into useful elements is incredible.

answered Jan 7 '14 at 20:28

community wiki
DannyK

Recently I had to split a camel-cased word into subwords. There are no delimiters, just upper characters.

```
#include <string>
#include <list>
#include <locale> // std::isupper

template<class String>
const std::list<String> split_camel_case_string(const String &s)
{
    std::list<String> R;
    String w;

    for (String::const_iterator i = s.begin(); i < s.end(); ++i) {  {
        if (std::isupper(*i)) {
            if (w.length()) {
                R.push_back(w);
                w.clear();
            }
        }
        w += *i;
    }

    if (w.length())
        R.push_back(w);
    return R;
}
```

For example, this splits "AQueryTrades" into "A", "Query" and "Trades". The function works with narrow and wide strings. Because it respects the current locale it splits "RaumfahrtÜberwachungsVerordnung" into "Raumfahrt", "Überwachungs" and "Verordnung".

Note `std::upper` should be really passed as function template argument. Then the more generalized from of this function can split at delimiters like `","`, `";"` or `" "` too.

edited Sep 14 '11 at 9:47

community wiki
2 revs
Andreas Spindler

I use this simpleton because we got our String class "special" (i.e. not standard):

```
void splitString(const String &s, const String &delim, std::vector<String> &result) {
    const int l = delim.length();
    int f = 0;
    int i = s.indexOf(delim,f);
    while (i>=0) {
        String token( i-f > 0 ? s.substring(f,i-f) : "");
        result.push_back(token);
        f=i+l;
        i = s.indexOf(delim,f);
    }
    String token = s.substring(f);
    result.push_back(token);
}
```

edited Sep 1 '10 at 14:32

community wiki
3 revs, 3 users 68%
Abe

The following is a much better way to do this. It can take any character, and doesn't split lines unless you want. No special libraries needed (well, besides std, but who really considers that an extra library), no pointers, no references, and it's static. Just simple plain C++.

```
#pragma once
#include <vector>
#include <sstream>
using namespace std;
class Helpers
{
    public:
        static vector<string> split(string s, char delim)
        {
            stringstream temp (stringstream::in | stringstream::out);
            vector<string> elems(0);
            if (s.size() == 0 || delim == 0)
                return elems;
            for each(char c in s)
            {
                if(c == delim)
                {
                    elems.push_back(temp.str());
                    temp = stringstream(stringstream::in | stringstream::out);
                }
                else
                    temp << c;
            }
            if (temp.str().size() > 0)
                elems.push_back(temp.str());
                return elems;
            }

        //Splits string s with a list of delimiters in delims (it's just a list, like if
we wanted to
        //split at the following letters, a, b, c we would make delims="abc".
        static vector<string> split(string s, string delims)
        {
            stringstream temp (stringstream::in | stringstream::out);
            vector<string> elems(0);
            bool found;
            if(s.size() == 0 || delims.size() == 0)
                return elems;
            for each(char c in s)
            {
                found = false;
                for each(char d in delims)
                {
                    if (c == d)
                    {
                        elems.push_back(temp.str());
                        temp = stringstream(stringstream::in | stringstream::out);
                        found = true;
                        break;
                    }
                }
                if(!found)
                    temp << c;
            }
            if(temp.str().size() > 0)
                elems.push_back(temp.str());
            return elems;
        }
};
```

edited Feb 27 '12 at 16:29

community wiki
5 revs, 2 users 62%
kelton52

I wrote the following piece of code. You can specify delimiter, which can be a string. The result is similar to Java's String.split, with empty string in the result.

For example, if we call split("ABCPICKABCANYABCTWO:ABC", "ABC"), the result is as follows:

```
0  <len:0>
1 PICK <len:4>
2 ANY <len:3>
3 TWO: <len:4>
4  <len:0>
```

Code:

```
vector <string> split(const string& str, const string& delimiter = " ") {
    vector <string> tokens;

    string::size_type lastPos = 0;
    string::size_type pos = str.find(delimiter, lastPos);

    while (string::npos != pos) {
        // Found a token, add it to the vector.
        cout << str.substr(lastPos, pos - lastPos) << endl;
        tokens.push_back(str.substr(lastPos, pos - lastPos));
        lastPos = pos + delimiter.size();
        pos = str.find(delimiter, lastPos);
    }

    tokens.push_back(str.substr(lastPos, str.size() - lastPos));
    return tokens;
}
```

answered Oct 7 '12 at 5:26 community wiki Jim Huang

Get Boost ! : -)

```
#include <boost/algorithm/string/split.hpp>
#include <boost/algorithm/string.hpp>
#include <iostream>
#include <vector>

using namespace std;
using namespace boost;

int main(int argc, char**argv) {
    typedef vector < string > list_type;

    list_type list;
    string line;

    line = "Somewhere down the road";
    split(list, line, is_any_of(" "));

    for(int i = 0; i < list.size(); i++)
    {
        cout << list[i] << endl;
    }

    return 0;
}
```

This example gives the output -

```
Somewhere
down
the
road
```

answered Apr 7 '13 at 16:07 community wiki Aleksey Bykov

This is my versión taken the source of Kev:

```
#include <string>
#include <vector>
```

```
void split(vector<string> &result, string str, char delim ) {
  string tmp;
  string::iterator i;
  result.clear();

  for(i = str.begin(); i <= str.end(); ++i) {
    if((const char)*i != delim  && i != str.end()) {
      tmp += *i;
    } else {
      result.push_back(tmp);
      tmp = "";
    }
  }
}
```

After, call the function and do something with it:

```
vector<string> hosts;
split(hosts, "192.168.1.2,192.168.1.3", ',');
for( size_t i = 0; i < hosts.size(); i++){
  cout <<  "Connecting host : " << hosts.at(i) << "..." << endl;
}
```

answered Jul 28 '11 at 12:38

community wiki
Jairo Abdiel Toribio Cisne

1 2 next

protected by Blorgbeard Dec 4 '12 at 23:26

Thank you for your interest in this question. Because it has attracted low-quality answers, posting an answer now requires 10 reputation on this site.

Would you like to answer one of these unanswered questions instead?