Goal:

Computer Science teachers normally grade programming assignments based on their **correctness**. (That is, as long a program generates the required output and agrees with the specification, it will get most of the points.) But this is an obvious misleading to the students since many other software properties are as important as **correctness**. In the past several decades it happened very often that a software company had to abandon correct programs due to other reasons. Here we list a few the most important software properties, i.e. **performance**, **scalability** and **robustness**, which may cause a correct program to be abandoned.

Performance, scalability and robustness:

Performance shows the response of the system to performing certain actions for a certain period of time. For example, if we search a target record in an array, there can be at least three different methods to do the work and the performances of them are very different.

Method	Time complexity (Big O)
Linear search	N
Binary search	$\text{Log}_2 n$
Hash function/table	C

Scalability is the property of a system to handle a growing amount of work by adding resources to the system. For example, if a system can process ship records with N data fields, can the program easily be modified to handle ship records with N+ or N- data fields?

Robustness is the ability of a computer system to cope with errors during execution and cope with invalid input. But sometimes the line between valid and invalid input is not clear. As a programmer we should train ourselves to write program which can cope with most cases, instead of just certain cases, since we do not know what exceptional cases may occur in the coming years.

Let us use the following example to elaborate these points.

Ship Name	Owner	Year Built	Status
“Royal Caribbean International”	“Jurre Van Wouw”	1987	“For sale”
“American Spirit”	“John West”	2005	Pending
“Carnival Breeze”	“Alberto Alvarado”	2012	“For sale”
“Carnival Fantasy”	“Donald Trump”	2010	Sold
“Celebrity Apex”	“Ivanka Johnson”	1999	Sold

Each row in the table above is a record of a ship which is currently for sale, pending or sold. The data provided in an input file consists of four columns: Ship name, Owner, Year Built, and Status. Double quotes are used to delimit a data token in case there are blanks in the data token. And a blank is used to separate two adjacent data tokens.

The function getline() is used to read a row in the file at a time. You are assigned to write a function (or functions) to decompose a row into four data tokens.

```
int main()
{
    string str1  = "  \"Royal Caribbean International \"  \" Jurre Van Wouw\"  1987
Pending";
   parseString(str1);
/// The function parseString() will display:
/// Royal Caribbean International
/// Jurre Van Wouw
/// 1987
/// Pending

    return 0;
}
```

After using getline(), we can use the parseString() function, as shown in the example above, to handle the string of four data tokens assuming that there is exactly one blank between two adjacent data tokens. As long as every row in the data file conforms to this format, system can run smoothly. However, in practice very often this is not the case since the following two situations may occur.

The first situation is that the number of data fields in a ship record is increased or reduced due to various reasons.

The second situation is that data files may have extra blanks in the data rows.

That is, for an average person the following three data strings are the same. But for computer systems, any data string, which does not strictly conform to the format described above, will cause system errors.

"   \"Royal Caribbean International \" \" Jurre Van Wouw\" 1987 \"For Sale\""

" \" Royal   Caribbean International \" \" Jurre Van Wouw\" 1987 \"For Sale\" "

"\" Royal Caribbean International \" \" Jurre Van Wouw\" 1987 \"For Sale\""

Therefore, when we write a function like parseString(), we must consider its scalability and robustness.

Exercise:

Implement the parseString() function which decomposes a string into data tokens. A data token is either a string which is delimited by a pair of double quotes or a string without any blank in it. The following is a test program for such function. The data tokens are displayed in the way that all leading blanks, all tailing blanks and extra blanks between two words are removed. In each case different number of data tokens are displayed.

```
int main()
{
    string str1  = "   \"Royal Caribbean International \"  \" Jurre Van Wouw\"  1987
\"For Sale\"   ";
   parseString(str1);
/// The function parseString() will display four tokens:
/// Royal Caribbean International
/// Jurre Van Wouw
/// 1987
/// For Sale
```

```
    string str2  = "   \" Royal   Caribbean  International \" \" Jurre Van Wouw\"
1987  \"For Sale\"  \\10\\02\\2009  ";
   parseString(str2);
/// The function parseString() will display five tokens:
/// Royal Caribbean International
/// Jurre Van Wouw
/// 1987
/// For Sale
/// 10\02\2009


     string str3  = "\" Royal Caribbean International \"\" Jurre Van Wouw\"  1987  ";
     parseString(str3);
/// The function parseString() will display three tokens:
/// Royal Caribbean International
/// Jurre Van Wouw
/// 1987


      return 0;
}
```