Reading, Writing, and Parsing Text Files Using C++

by Robert J. Yager

Approved for public release; distribution is unlimited.
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
Reading, Writing, and Parsing Text Files Using C++

Robert J. Yager
Weapons and Materials Research Directorate, ARL

Approved for public release; distribution is unlimited.
## Reading, Writing, and Parsing Text Files Using C++

Text files are often used to store tabulated data and user-modifiable inputs for scientific modeling. This report presents a set of functions, written in C++, that can be used to read, write, and parse text files.

<table>
<thead>
<tr>
<th>1. REPORT DATE</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2013</td>
<td>Final</td>
<td>January 2012—March 2013</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading, Writing, and Parsing Text Files Using C++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5a. CONTRACT NUMBER</th>
<th>5b. GRANT NUMBER</th>
<th>5c. PROGRAM ELEMENT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AH80</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHOR(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robert J. Yager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S. Army Research Laboratory</td>
</tr>
<tr>
<td>ATTN: RDRL-WML-A</td>
</tr>
<tr>
<td>Aberdeen Proving Ground, MD 21005-5066</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARL-TN-545</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DISTRIBUTION/AVAILABILITY STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release; distribution is unlimited.</td>
</tr>
</tbody>
</table>

### Abstract

Text files are often used to store tabulated data and user-modifiable inputs for scientific modeling. This report presents a set of functions, written in C++, that can be used to read, write, and parse text files.

<table>
<thead>
<tr>
<th>15. SUBJECT TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++, read, write, parse, text files</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16. SECURITY CLASSIFICATION OF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. REPORT: Unclassified</td>
</tr>
<tr>
<td>b. ABSTRACT: Unclassified</td>
</tr>
<tr>
<td>c. THIS PAGE: Unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17. LIMITATION OF ABSTRACT</th>
<th>18. NUMBER OF PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>UU</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19a. NAME OF RESPONSIBLE PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robert J. Yager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19b. TELEPHONE NUMBER (Include area code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>410-278-6689</td>
</tr>
</tbody>
</table>
Contents

Acknowledgments iv

1. Introduction 1

2. Reading and Writing Text Files 1
   2.1 ReadTextFile() Function .................................................................1
   2.2 WriteTextFile() Function ...............................................................2
   2.3 ReadTextFile()/WriteTextFile() Example .......................................2

3. Comment Removal 3
   3.1 RemoveLineComments() Function ................................................3
   3.2 RemoveBlockComments() Function ...............................................4
   3.3 RemoveLineComments()/RemoveBlockComments() Example ..........4

4. Parsing Character Arrays 5
   4.1 Parse() Function ...........................................................................5
   4.2 Parse2D() Function ....................................................................6
   4.3 Parse()/Parse2D() Example ........................................................8

5. Summary 9

Distribution List 11
Acknowledgments

The author would like to thank Dr. Benjamin Breech of the U.S. Army Research Laboratory’s Weapons and Materials Research Directorate. Dr. Breech provided technical and editorial recommendations that improved the quality of this report.
1. Introduction

Text files are often used to store tabulated data and user-modifiable inputs for scientific modeling. This report presents a set of functions, written in C++, that can be used to read, write, and parse text files. A summary sheet is provided at the end of this report. It presents the ylo namespace, which contains the six functions that are described in this report.

2. Reading and Writing Text Files

2.1 ReadTextFile() Function

The ReadTextFile() function can be used to create a character array that contains all of the information from a text file.

Note that the ReadTextFile() function uses the “new” command to allocate memory for the character array that is pointed to by the return value. Thus, to avoid memory leaks, each use of the ReadTextFile() function should be accompanied by a use of the “delete[]” operator.

ReadTextFile() Code

```c
inline char* ReadTextFile(const char* filename){//<=========READS A TEXT FILE INTO A CHARACTER ARRAY
    FILE* f=fopen(filename,"rb");//........binary is necessary to get n right
    if(!f)printf("\nCan't open \"%s\".\n",filename),exit(1);//........is f open?
    size_t n;/*<-*/fseek(f,0,SEEK_END),n=ftell(f),rewind(f);///<.get size of file
    char* b=new char[n+1];/*<-*/fread(b,1,n,f),fclose(f),b[n]='$';//.....read f
    return b;//.....................note that b points to newly allocated memory
}~/~/YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

ReadTextFile() Parameters

- **filename** specifies the name of a text file.

ReadTextFile() Return Value

The ReadTextFile() function returns a pointer to the beginning of a character array that stores all of the information from the input file.
If the file that is specified by `filename` cannot be opened, the `ReadTextFile()` function calls the `exit()` function with status code 1. Inability to open a file is typically the result of an incorrectly specified filename or path.

### 2.2 WriteTextFile() Function

The `WriteTextFile()` function can be used to write a character array to a text file.

**WriteTextFile() Parameters**
- **filename**: specifies the name of the text file that will be written.
- **text**: points to a character array that contains the text that will be written to the text file.
- **mode**: specifies how the text will be written to the output file. Use “w” to overwrite an existing file. Use “a” to append to the end of an existing file. In either case, if a file with the same name as the output filename doesn’t already exist, a new file will be created. The default value is “w.”

**WriteTextFile() Return Value**

The `WriteTextFile()` function returns the number of characters that were successfully written to the output file.

If the file that is specified by `filename` cannot be opened, the `WriteTextFile()` function calls the `exit()` function with status code 1. Inability to open a file is often the result of an incorrectly specified filename or path. However, it can also be the result of a file being marked as read only, as may be the case if a file is open in another program.

### 2.3 ReadTextFile()/WriteTextFile() Example

The following example first creates a file named “example.txt,” then reads the newly created file and displays its contents:
3. Comment Removal

3.1 RemoveLineComments() Function

The RemoveLineComments() function can be used to overwrite line comments in a character array. A line comment is a comment that begins with some identifying set of characters and continues to the end of the line. Memory that is occupied by line comments isn’t actually freed by the RemoveLineComments() function. Instead, all of the line-comment characters are replaced with a user-specified character.

RemoveLineComments() Code

```cpp
inline char* RemoveLineComments(char* text, const char* start="#", char c=' ')
{
    char* s=text;
    while(s=strstr(s,start))memset(s,c,strcspn(s,"


---

RemoveLineComments() Parameters

- **text**: text points to a character array that contains the text that will have its line comments overwritten. Typically, the character array is created using the ReadTextFile() function.
- **start**: start is used to identify the beginning of a line comment. The default value is “#.”
- **c**: c specifies the character that will be used to replace the characters in the line comment. The default value is a space.
RemoveLineComments() Return Value
The RemoveLineComments() function returns the input pointer `text`. Although the pointer `text` is unmodified, the character array that `text` points to is modified.

3.2 RemoveBlockComments() Function
The RemoveBlockComments() function can be used to overwrite block comments in a character array. A block comment is a comment that begins with some identifying set of characters and ends with a different set of identifying characters. Block comments may or may not span multiple lines. Note that the memory that is occupied by block comments isn’t actually freed by the RemoveBlockComments() function. Instead, all of the block-comment characters are replaced with a user-specified character.

RemoveBlockComments() Code

```
inline char* RemoveBlockComments(char* text, char* start, char* end, char c)
{
    char* s = text;
    while (s = strstr(s, start)) {
        int t = strstr(s, end) - s + strlen(end);
        if (t > 0) t = 0;
        memset(s, c, t);
    }
    return text;
}
```

RemoveBlockComments() Parameters
- `text`: points to a character array that contains the text that will have its block comments overwritten. Typically, the character array is created using the ReadTextFile() function.
- `start`: indicates the beginning of a block comment. The default value is “/*.”
- `end`: indicates the end of a block comment. The default value is “*/.”
- `c`: specifies the character that will be used to replace the characters in the line comment. The default value is a space.

RemoveBlockComments() Return Value
The RemoveBlockComments() function returns the input pointer `text`. Although the pointer `text` is unmodified, the character array that `text` points to is modified.

3.3 RemoveLineComments()/RemoveBlockComments() Example
The following example begins by creating a character array from a text file that contains both line comments and block comments. Then, both the RemoveLineComments() and the
RemoveBlockComments() functions are used to remove comments from the character array. The contents of the character array are displayed at each step.

```c
#include "y_io.h"//.................................<cstdio>
int main(){
    char* s=yIo::ReadTextFile("comment_example.txt");
    printf("ORIGINAL TEXT:
%s

",s);
    printf("LINE COMMENTS REMOVED:
%s

",yIo::RemoveLineComments(s));
    printf("LINE COMMENTS AND BLOCK COMMENTS REMOVED:
%s

",
    yIo::RemoveBlockComments(s));
    delete[] s;
}~/~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~
```

comment_example.txt:
```
#sample comment
Hello/*sample comment*/ World!
```

OUTPUT:
```
ORIGINAL TEXT:
#sample comment
Hello/*sample comment*/ World!

LINE COMMENTS REMOVED:
Hello/*sample comment*/ World!

LINE COMMENTS AND BLOCK COMMENTS REMOVED:
Hello                   World!
```

4. Parsing Character Arrays

4.1 Parse() Function

The Parse() function can be used to separate the text contained in a character array into a set of smaller character arrays called tokens. Tokens are separated by user-defined delimiting characters. Common delimiters are commas, tabs, and spaces.

The Parse() function works by searching for tokens in the character array pointed to by the input variable text. When a token is found, the delimiting character that immediately follows the token is replaced by a NULL character. A pointer to the beginning of the token is stored in a vector.
Parse() Code

```cpp
inline vector<char*> Parse( // <======================================= 1D PARSER
    char* text, // <---------------------------- THE CHARACTER ARRAY THAT WILL BE PARSED
    const char* delimiters=" ,\t\n\f\r"){ // <---- CHARACTERS THAT SEPARATE TOKENS
    vector<char*> S(1,text=strtok(text,delimiters));//....output array of tokens
    while(text=strtok(NULL,delimiters))S.push_back(text);
    return S;
} // ~~~~YAGENAUT@GMAIL.COM ~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

Parse() Parameters

text  text points to the character array that will be parsed. Typically, the character array is created using the ReadTextFile() function.

delimiters  delimiters specifies a set of characters that separate tokens. Any set of consecutive delimiter characters will act as a single delimiter. For example, suppose that the text string “8,,9” was parsed using the Parse() function and with delimiters set to “,”. The Parse() function will treat the set of three consecutive commas as a single delimiter. Thus, only two tokens will be found. By default, spaces, commas, tabs, line feeds, form feeds, and carriage returns are all treated as delimiters.

Parse() Return Value

The Parse() function returns a vector of pointers. Each pointer points to the beginning of a token. The tokens are stored in the character array that was originally pointed to by the text input parameter.

4.2 Parse2D() Function

The Parse2D() function can be used to separate the text contained in a character array into a set of smaller character arrays called tokens. Tokens are separated by two types of user-defined delimiting characters. The first set of delimiters separates tokens within a row of data. Common examples are spaces, commas, and tabs. The second set of delimiters separate data rows. Common examples are line feeds, form feeds, and carriage returns.

The Parse2D() function works by searching for row ends in the character array pointed to by the input variable text. When a row end is found, the delimiting character that immediately follows the row is replaced by a NULL character. The Parse() (not 2D) function is then used to parse the row.
Parse2D() Code

```cpp
inline vector<vector<char*>> Parse2D(const char* text,//<==========2D PARSER (CALLS 1D PARSER)
    const char* delimiters=" ,",//<---------------------THE CHARACTER ARRAY THAT WILL BE PARSED
    const char* ends="\n\f\r"){//<---------------------PRIMARY (COLUMN) DELIMITERS
    vector<vector<char*>> S;//............................output array of tokens
    char* b,* c=new char[strlen(delimiters)+strlen(ends)+1];
    strcpy(c,delimiters),strcat(c,ends);
    while(*(b=text+strspn(text,c))){//............................find row start
        text=b+strcspn(b,ends),*text=0,text++;//......................find row end
        S.push_back(Parse(b,delimiters));}//.............................parse row
    delete[] c;
    return S;//...the number of columns per row may not be the same for all rows
}
```

Parse2D() Parameters

- **text**: points to the character array that will be parsed. Typically, the character array is created using the ReadTextFile() function.

- **delimiters**: specifies a set of characters that separate tokens within the same row. Any set of consecutive delimiting characters act as a single delimiter. Default values are spaces, commas, and tabs.

- **ends**: specifies a set of characters that separate rows of tokens. Default values are line feeds, form feeds, and carriage returns.

Parse2D() Return Value

The Parse2D() function returns a vector of vectors of pointers. Each pointer points to the beginning of a token. The tokens are stored in the character array that was originally pointed to by the `text` input parameter.
4.3 Parse()/Parse2D() Example

The following example begins by creating a character array from a text file that contains a table of numbers. The example parses the character array first using the Parse() function, then using the Parse2D function. Between the two parsings, the character array is deleted, then recreated.

```c
#include "y_io.h" //.............................................<cstdio>,<vector>
int main()
{
    char* s=yIo::ReadTextFile("parse_example.txt");
    printf("ORIGINAL TEXT:
%s
",s);
    std::vector<char*> A=yIo::Parse(s);
    printf("PARSED TEXT:
");
    for(int i=0;i<9;++i)printf("%s , ",A[i]);
    delete[] s;
    s=yIo::ReadTextFile("parse_example.txt");
    std::vector<std::vector<char*> > B=yIo::Parse2D(s);
    printf("2D-PARSED TEXT:
");
    for(int i=0;i<3;++i)for(int j=0;j<3;++j)
        printf("%s , %s",B[i][j],j==2?" ":"
");
    delete[] s;
//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
}
```

parse_sample.txt:

```
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0
```

OUTPUT:

```
ORIGINAL TEXT:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0
PARSED TEXT:
1.0 , 2.0 , 3.0 , 4.0 , 5.0 , 6.0 , 7.0 , 8.0 , 9.0
2D-PARSED TEXT:
1.0 , 2.0 , 3.0
4.0 , 5.0 , 6.0
7.0 , 8.0 , 9.0
```
5. Summary

A summary sheet is provided at the end of this report. It presents the yIo namespace, which contains the six functions that are described in detail in this report. Also presented is an example that can be used to test the performance of the functions contained in the yIo namespace.
<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEFENSE TECHNICAL (PDF)</td>
<td>1</td>
<td>RDRL WML</td>
</tr>
<tr>
<td></td>
<td>INFORMATION CTR DTIC OCA</td>
<td></td>
<td>M ZOLTOSKI</td>
</tr>
<tr>
<td>1</td>
<td>DIRECTOR (PDF)</td>
<td>1</td>
<td>RDRL WML A</td>
</tr>
<tr>
<td></td>
<td>US ARMY RESEARCH LAB IMAL HRA</td>
<td></td>
<td>B BREECH</td>
</tr>
<tr>
<td>1</td>
<td>DIRECTOR (PDF)</td>
<td>1</td>
<td>P BUTLER</td>
</tr>
<tr>
<td></td>
<td>US ARMY RESEARCH LAB RDRL CIO LL</td>
<td></td>
<td>B FLANDERS</td>
</tr>
<tr>
<td>1</td>
<td>GOVT PRINTG OFC (PDF)</td>
<td></td>
<td>W OBERLE</td>
</tr>
<tr>
<td></td>
<td>A MALHOTRA</td>
<td></td>
<td>C PATTERSON</td>
</tr>
<tr>
<td>2</td>
<td>ARDEC (PDF)</td>
<td>1</td>
<td>R PEARSON</td>
</tr>
<tr>
<td></td>
<td>RDA REIS AM D ERICSON C DOUROS</td>
<td></td>
<td>L STROHM</td>
</tr>
<tr>
<td>1</td>
<td>AMRDEC (PDF)</td>
<td>1</td>
<td>A THOMPSON</td>
</tr>
<tr>
<td></td>
<td>RDMR SSL T G WIGGS</td>
<td></td>
<td>R YAGER (1 PDF, 5 HC)</td>
</tr>
<tr>
<td>2</td>
<td>DRTA (PDF)</td>
<td></td>
<td>DLYON</td>
</tr>
<tr>
<td></td>
<td>RDMR SSL T Y SOHN J BELL</td>
<td></td>
<td>RDRL WML B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N TRIVEDI</td>
</tr>
<tr>
<td>1</td>
<td>AFRL/RWW (PDF)</td>
<td></td>
<td>RDRL WML C</td>
</tr>
<tr>
<td></td>
<td>RDMR SSL T C EWING</td>
<td></td>
<td>S AUBERT</td>
</tr>
<tr>
<td>1</td>
<td>NAWCAD (PDF)</td>
<td></td>
<td>RDRL WML D</td>
</tr>
<tr>
<td></td>
<td>RDMR SSL T E LYNCH</td>
<td></td>
<td>R BEYER</td>
</tr>
<tr>
<td>1</td>
<td>NSWCDD (PDF)</td>
<td></td>
<td>RDRL WML E</td>
</tr>
<tr>
<td></td>
<td>RDMR SSL T K LEWIS</td>
<td></td>
<td>P WEINACHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RDRL WML F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DLYON</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RDRL WML G</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>J SOUTHW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RDRL WML H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>J NEWILL</td>
</tr>
</tbody>
</table>

**ABERDEEN PROVING GROUND**

<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>DIR USARL (5 HC, 25 PDF)</td>
</tr>
<tr>
<td></td>
<td>P GILILCH</td>
</tr>
<tr>
<td></td>
<td>L HALL</td>
</tr>
<tr>
<td></td>
<td>T MYERS</td>
</tr>
<tr>
<td></td>
<td>C KENNEDY</td>
</tr>
<tr>
<td></td>
<td>N MOHOLKAR</td>
</tr>
<tr>
<td></td>
<td>K RAFAELS</td>
</tr>
<tr>
<td></td>
<td>RDRL WM</td>
</tr>
<tr>
<td></td>
<td>P BAKER</td>
</tr>
</tbody>
</table>
INTENTIONALLY LEFT BLANK.