

Strings

- **Problem:**

A direct-mail advertising agency has decided to personalize its sweepstakes offers. It has prepared a basic letter with a particular customer's name, address, spouse's name, and other personal information. The company would like a computer program to make the appropriate changes to address each customer individually. As a first test, the company would like the program to make one set of changes: to replace all occurrences of

1. Smith by Johnson
2. Mr. by Ms.
3. 421 Main St. by 18 Windy Lane
4. wife by husband
5. Susan by Robert
6. her by his

Here is the basic letter:

Congratulations, Mr. Smith! The Smith family of 421 Main St. may have already won a new One-million-dollar house!! Your neighbors at 421 Main St. will be so surprised when you, Mr. Smith, move into your new house with your wife, Susan! And her eyes will light up with joy at her fabulous new home! Enter the sweepstakes now, and you and the Smith family may win it all!.

Write a C++ program that reads in the text line by line, displays each line as it is read in, makes the designated changes, and displays the revised text.

The String Class

- C++ has a **string class**. This class allows one to declare variables of data type **string** and provides **methods** (or member functions) for manipulating strings. (To access the string methods, one must `#include <string.h>`)

- **Declaring and Using String Variables:**

```
string str1 = "Hello";  
string str2 = "Computer Science is fun!";  
  
cout << str1 << " , " << str2 << endl;
```

This statement prints:
Hello, Computer Science is fun!

- **Assigning a Value to a String Variables:**

```
str1 = "Today";           // can use the assignment statement  
str2 = str1;  
  
cin >> str1;             // can use cin to read in a string value
```

Note: cin will read in characters until the first whitespace character (space, TAB, or ENTER) is encountered.

Reading in a String Using the Function `getline()`:

- **Function Syntax:**

`getline(input_stream, string_variable, delimiting_character);`

`getline()` reads characters (including whitespace characters) from the *input_stream* into the specified *string_variable* until the *delimiting_character* is encountered. The third parameter is optional. If it is omitted, the default is the newline character (Enter).

- **Example:**

```
string str1, str2, str3;
```

```
getline(cin, str1, '.');    //user types: Hello there.
```

```
                           // str1 = Hello there
```

```
                           // (Note: no period)
```

```
getline(cin, str2, '\n');  //user types: Great Idea!
```

```
                           // str2 = Great Idea!
```

```
                           // (Note: with exclamation point)
```

```
getline(cin, str3);       //reads cin until Enter pressed
```

String Operations

- **Concatenation (+ operator):**

```
string str1, str2, str3;
```

```
str1 = "yesterday";
```

```
str2 = "morning";
```

```
str3 = str1 + " " + str2; //str3 == "yesterday morning"
```

```
cout << str3 << endl; //prints "yesterday morning"
```

```
string str4 = "empty train";
```

```
str4 = str4 + 's'; //str4 == "empty trains"
```

could have used:

```
str4 += 's'; //str4 == "empty trains"
```

- **Note:** the use of an operator (like +) for multiple uses is called **operator overloading**.

- **Comparing Strings:**

Strings can be compared by using the standard relational operators. Strings are compared character by character according to their ASCII codes. Two strings are equal if and only if they have the exact same number of characters and match position by position.

(Note: 'A' < 'Z' < 'a' < 'z'.

```
string str1 = "cat", str2 = "dog";
```

```
if (str1 == str2)
```

```
    cout << "alike" << endl;
```

```
else
```

```
    cout << "different" << endl; //prints "different"
```

Note:

```
"Cat" < "Dog" < "cat" < "cats" < "dog"
```

Methods of the String Class

To use a method of a class, the syntax is:

object.method();

e.g.;

```
cout.precision();  
cin.eof();
```

- **The Methods `length()` and `size()`:**

To find the numbers of characters currently contained in a string use either the method `length()` or `size()`. (They both do the same thing.)

```
string state = "New York";  
string city = "Cincinnati";  
int len, numchars;
```

```
len = state.length();           //len = 8  
numchars = city.size();        //numchars = 10
```

- **The Data Type `size_type`:**

Officially, `length()` and `size()` return a value of data type `size_type` (which is an unsigned integer). To use this data type, it must be preceded by **`string::`**.

```
string state = "New York";  
string city = "Cincinnati";
```

```
string::size_type len, numchars;
```

```
len = state.length();           //len = 8  
numchars = city.size();        //numchars = 10
```

- **A String as an Array of characters:**

A string can be viewed as a sequence of elements within an array of characters.

```
char c,d;
string str = "wing";
string item = "compater";

string::size_type len = str.length();    //len == 4

c = str[3];           // c == 'g'
d = str[0];           // d == 'w'

item[4] = 'u';        // item == "computer"

For (int i=0; i < len ; i+ +)
    cout << str[i] << endl;
```

This prints:

```
w
i
n
g
```

- **The Empty (or Null) String:**

A string of length zero is called the empty (or null) string.

```
string name = "Lenny";

string::size_type len = name.size();    //len == 5

name.clear();                           //name == ""
len = name.size();                       //len == 0

name = "";                               //name is empty
len = name.size();                       //len == 0
```

- **The Method find():**

Used to search for the presence of a character or substring within a string.

- **Syntax:**

position = source.find(string_to_find, start_position);

- **Examples:**

```
string str = "this orchestra is fabulous";  
string::size_type pos, start_position = 0;  
string string_to_find = "orchestra";
```

```
pos = str.find(string_to_find, start_position);    //pos = 5
```

```
pos = str.find("band",start_position);    //pos = string::npos
```

```
pos = str.find("is",0);                    //pos = 2
```

```
pos = str.find('c',0);                    //pos = 7
```

```
pos = str.find("is",5);                   //pos = 15
```

- **Note:**

The value **npos** is the maximum number of characters that a string can hold. Therefore, it is one greater than the largest possible character position. The exact value of **npos** is irrelevant. What matters is that it represents a value that cannot be an index into the string.

```
int count = 0;
```

```
pos = str.find('s' ,0);  
while(pos != str.npos)  
{  
    count + + ;  
    pos = str.find('s' ,pos + 1);  
}
```

```
cout << "s was found " << count << " times." << endl;
```

- **The Method `replace()`:**

Used to replace part of a string with another string.

- **Syntax:**

```
source.replace(start_position, numchars, new_string);
```

- **Examples:**

```
string message = "what a nice day!";  
string str = "good";
```

```
message.replace(7, 4, "good"); //message = what a good day!
```

```
message.replace(7, str.length(), str); //same result
```

```
message.replace(7, 3, "bad"); //message = what a badd day!
```

```
message.replace(7, 6, "rotten"); //message = what a rottenay!
```

- **The Methods `erase()` and `insert()`:**

Respectively used to erase part of a string and to insert a new string into an existing string.

- **Syntax:**

```
source.erase(start_position, numchars);
```

```
source.insert(start_position, string_to_insert);
```

- **Examples:**

```
string message = "what a nice day!";  
string oldstr = "nice";  
string newstr = "rotten";
```

```
int pos = message.find(oldstr, 0); //find position of oldstr
```

```
message.erase(pos, oldstr.length()); //erase oldstr
```

```
message.insert(pos, newstr); //insert newstr
```

```
//now message = what a rotten day!
```

- **The Method substr():**
Used to extract part of a string.

- **Syntax:**

dest_string = *source.substr(start_position, numchars)*;

If you omit the second parameter (numchars), substr() will extract all the characters from the start_position until the end of the string.

- **Examples:**

```
string message = "what a nice day!";
```

```
int pos = message.find("nice", 0);
```

```
string str1 = str.substr(pos);    //str1 = nice day!
```

```
string str2 = str.substr(pos, 4); //str2 = nice
```

Strings and Functions

Sending a string to a function works the same as with any other data type. The formal parameter must be of type **string**. If the string is to be changed within the function, it must be sent as a reference parameter using the **&** operator (**string &**). It is also possible to use **string** as a return data type from a function.

- **Examples:**

```
//prototypes
void change(string &);
string donotappend(string);

string str1 = "cat";
string str2 = "cow";
string str3;

change(str1);                //after call str1 = cats

str3 = donotchange(str2);    //after call str3 = cows
...                          //          str2 = cow

void change(string &str)
{
    str += 's';
    return;
}

string donotchange(string str)
{
    str = str + 's';
    return(str);
}
```

Strings and File I/O

```
/* program to illustrate strings and File I/O */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void change(string &str);      //prototype

int main()
{
    string line;

    // open input file
    ifstream infile("c:\\mypgms\\prob8in.txt");
//    ifstream infile("con");      //un-comment for debugging

    // open output file
    ofstream outfile("c:\\mypgms\\prob8out.txt");
//    ofstream outfile("con");    //un-comment for debugging

    while (getline(infile, line))
    {
        change(line);
        outfile << line << endl;
    }

    infile.close();            //close input file
    outfile.close();          //close output file
    return 0;
}

void change(string &str)
{
    str += 's';
    return;
}
```

File I/O using open() and Global File Reference

```
/* program to illustrate strings and File I/O */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
void change(string &str);          //prototype

ifstream infile;                   //global declarations
ofstream outfile;

int main()
{
    string line;

    // open input file
    infile.open("c:\\mypgms\\prob8in.txt");
//    infile.open("con");          //un-comment for debugging

    // open output file
    outfile.open("c:\\mypgms\\prob8out.txt");
//    outfile.open("con");        //un-comment for debugging

    while (getline(infile, line))
    {
        change(line);
        outfile << line << endl;
    }

    infile.close();                 //close input file
    outfile.close();                //close output file
    return 0;
}

void change(string &str)
{
    str += 's';
    return;
}
```

Strings and File I/O - redirecting cin and cout

```
/* program to illustrate strings and File I/O */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
void change(string &str);      //prototype

int main()
{
    string line;

    //un-comment to redirect cin to a file
    ifstream cin("c:\\mypgms\\prob8in.txt");

    //un-comment to redirect cout to a file
    ofstream cout("c:\\mypgms\\prob8out.txt");

    while (getline(cin, line))
    {
        change(line);
        cout << line << endl;
    }

    return 0;
}

void change(string &str)
{
    str += 's';
    return;
}
```

Returning to Our Problem

- **Pseudocode:**

*while there is a line of the letter to read
read in a line of the original letter
print the original line
replace the old strings in the line by the new ones
print the new line*

- **Main Program:**

```
/* program to read a letter and replace all occurrences of old
 * strings with new strings.
 */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

//Function prototypes
void change(string &line);
... //more prototypes to be added

int main()
{
    string line;

    while (getline(cin, line))
    {
        cout << line << endl; //output original line
        change(line); //make changes
        cout << line << endl; //output revised line
    }

    return 0;
}
```

- **Pseudocode for change():**

read in a set of replacements (oldstr & newstr)
while there are data values
make changes in line
read in a set of replacements (oldstr & newstr)

- **Revised Pseudocode for change():**

*call **getchanges()** to read in a set of replacements*
while there are data values
*call **change_one_line()** to make changes in line*
*call **getchanges()** to read in a set of replacements*

- **Function change():**

```
/* Function change()
 * Input:
 *   line - a reference to the string to be processed
 * Process:
 *   repeatedly call getchanges() to get a new set of
 *   replacement strings (oldstr & newstr), until there
 *   are no more sets of changes.
 * Output:
 *   line - the processed string
 */
void change(string &line)
{
    string oldstr, newstr;

    while (getchanges(oldstr, newstr))
    {
        change_one_line(line, oldstr, newstr);
    }
    return;
}
```

● **The Function `getchanges()`:**

```
/* Function getchanges():
 * Input:
 *   oldstr - a reference to the old string to search for
 *   newstr - a reference to the new replacement string
 * Process:
 *   reads in a pair of strings (oldstr & newstr) if they exist
 * Output:
 *   oldstr - the old string to search for
 *   newstr - the new replacement string
 *   if they exist, returns true; else, returns false
 */
bool getchanges(string &oldstr, string &newstr)
{
    if (getline(cin, oldstr))
    {
        getline(cin, newstr);
        return true;
    }
    return false;
}
```

● **The Function `change_one_line()`:**

```
/* Function change_one_line():
 * Input:
 *   line - a reference to the string to be processed
 *   oldstr - the old string to search for
 *   newstr - the new replacement string
 * Process:
 *   searches line for position of oldstr
 *   if oldstr exists, erases oldstr and inserts newstr
 *   returns when all replacements have been made
 * Output:
 *   line - the processed string
 */
void change_one_line(string &line, string oldstr, string newstr)
{
    string::size_type pos;

    pos = line.find(oldstr, 0);
    while (pos != line.npos)
    {
        line.erase(pos, oldstr.length());
        line.insert(pos, newstr);
        pos = line.find(oldstr, pos + newstr.length());
    }
    return;
}
```

● Revised Main Program:

```
/* program to read a letter and replace all occurrences of old
 * strings with new strings.
 */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

//Function prototypes
void change(string &line);
bool getchanges(string &oldstr, string &newstr);
void change_one_line(string &line, string oldstr, string newstr);

int main()
{
    string line;

    while (getline(cin, line))
    {
        cout << line << endl;           //output original line
        change(line);                   //make changes
        cout << line << endl;           //output revised line
    }

    return 0;
}
```

● Revised Program Using File I/O:

```
/* program to read a letter and replace all occurrences of old
 * strings with new strings.
 */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

//Function prototypes
void change(string &line);
bool getchanges(ifstream &cfile, string &oldstr, string &newstr);
void change_one_line(string &line, string oldstr, string newstr);

int main()
{
    string line;

    // open input file
    ifstream infile("input.txt"); //comment-out for debugging
//    ifstream infile("con");      //un-comment for debugging

    // open output file
//    ofstream outfile("output.txt"); //comment-out for debugging
    ofstream outfile("con");      //un-comment for debugging

    while (getline(infile, line))
    {
//        outfile << line << endl;    //output original line
        change(line);                //make changes
        outfile << line << endl;    //output revised line
    }

    infile.close();                //close input file
    outfile.close();               //close output file

    return 0;
}
```

● Revised Function change():

```
/* Function change()
 * Input:
 *   line - a reference to the string to be processed
 * Process:
 *   repeatedly call getchanges() to get a new set of
 *   replacement strings (oldstr & newstr), until there
 *   are no more sets of changes.
 * Output:
 *   line - the processed string
 */
void change(string &line)
{
    string oldstr, newstr;

    // open changes file
    ifstream changesfile("changes.txt"); //comment for debug
// ifstream changesfile("con"); //un-comment for debugging

    while (getchanges(changesfile, oldstr, newstr))
    {
        change_one_line(line, oldstr, newstr);
    }

    changesfile.close();
    return;
}
```

● **The Revised Function `getchanges()`:**

```
/* Function getchanges():
 * Input:
 *   cfile - a reference to the file that contains the
 *             replacement pairs
 *   oldstr - a reference to the old string to search for
 *   newstr - a reference to the new replacement string
 * Process:
 *   reads in a pair of strings (oldstr & newstr) if they exist
 * Output:
 *   oldstr - the old string to search for
 *   newstr - the new replacement string
 *   if they exist, returns true; else, returns false
 */
bool getchanges(ifstream &cfile,string &oldstr,string &newstr)
{
    if (getline(cfile, oldstr)
        {
            getline(cfile, newstr);
            return true;
        }
    return false;
}
```

● **Revised Function change() - less modular version:**

```
/* Function change()
 * Input:
 *   line - a reference to the string to be processed
 * Process:
 *   repeatedly get a new set of replacement strings
 *   (oldstr & newstr), until there are no more sets of
 *   changes. Search line for position of oldstr.
 *   if oldstr exists, erase oldstr and insert newstr.
 *   Returns when all replacements have been made
 * Output:
 *   line - the processed string
 */
void change(string &line)
{
    string oldstr, newstr;
    string::size_type pos;

    // open changes file
    ifstream changesfile("changes.txt"); //comment for debug
// ifstream changesfile("con"); //un-comment for debugging

    while (getline(changesfile, oldstr))
    {
        getline(changesfile, newstr);
        pos = line.find(oldstr, 0);
        while (pos != line.npos)
        {
            line.erase(pos, oldstr.length());
            line.insert(pos, newstr);
            pos = line.find(oldstr, pos + newstr.length());
        }
    }

    changesfile.close();
    return;
}
```

● **Revised Program Using File I/O - Using Global File Reference:**

```
/* program to read a letter and replace all occurrences of old
 * strings with new strings.
 */
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

//Function prototypes
void change(string &line);
bool getchanges(string &oldstr, string &newstr);
void change_one_line(string &line, string oldstr, string newstr);

//global file reference
ifstream changesfile;

int main()
{
    string line;

    // open input file
    ifstream infile("input.txt"); //comment-out for debugging
//    ifstream infile("con"); //un-comment for debugging

    // open output file
//    ofstream outfile("output.txt"); //comment-out for debugging
    ofstream outfile("con"); //un-comment for debugging

    while (getline(infile, line))
    {
//        outfile << line << endl; //output original line
        change(line); //make changes
        outfile << line << endl; //output revised line
    }

    infile.close(); //close input file
    outfile.close(); //close output file
    return 0;
}
```

● **Revised Function change() - using global file reference:**

```
/* Function change()
 * Input:
 *   line - a reference to the string to be processed
 * Globals:
 *   changesfile - global reference to the file that contains
 *   the replacement pairs
 * Process:
 *   repeatedly call getchanges() to get a new set of
 *   replacement strings (oldstr & newstr), until there
 *   are no more sets of changes.
 * Output:
 *   line - the processed string
 */
void change(string &line)
{
    string oldstr, newstr;

    // open changes file
    changesfile.open("changes.txt"); //comment for debug
// changesfile.open("con"); //un-comment for debugging
    changesfile.clear(); // reset EOF flag

    while (getchanges(oldstr, newstr))
    {
        change_one_line(line, oldstr, newstr);
    }

    changesfile.close();
    return;
}
```

● **The Revised Function getchanges() - global file reference:**

```
/* Function getchanges():
 * Input:
 *     oldstr - a reference to the old string to search for
 *     newstr - a reference to the new replacement string
 * Globals:
 *     changesfile - global reference to the file that contains
 *                 the replacement pairs
 * Process:
 *     reads in a pair of strings (oldstr & newstr) if they exist
 * Output:
 *     oldstr - the old string to search for
 *     newstr - the new replacement string
 *     if they exist, returns true; else, returns false
 */
bool getchanges(string &oldstr,string &newstr)
{
    if (getline(changesfile, oldstr))
    {
        getline(changesfile, newstr);
        return true;
    }
    return false;
}
```

Using an Array of Strings

```
/* Program to illustrate an array of strings */
#include <iostream>
#include <string>
using namespace std;

const int SIZE = 5;

int main()
{
    string str[SIZE];

    str[0] = "Jones";
    str[1] = "Harrow";
    str[2] = "Tenenbaum";
    str[3] = "Arnow";
    str[4] = "Raphan";

    //print the original array
    cout << "Original Array" << endl;
    for (int i = 0; i < SIZE; i++)
        cout << str[i] << endl;

    return 0;
}
```

Passing an Array of Strings to a Function

```
/*Program to illustrate passing an array of strings to a function */
#include <iostream>
#include <string>
using namespace std;

const int SIZE = 5;

//function prototype
void printthearray(string str[]);

int main()
{
    string str[SIZE];

    str[0] = "Jones";
    str[1] = "Harrow";
    str[2] = "Tenenbaum";
    str[3] = "Arnow";
    str[4] = "Raphan";

    // print the array
    printthearray(str);

    return 0;
}

void printthearray(string str[])
{
    //print the array
    cout << "The Array of Strings" << endl;
    for (int i = 0; i < SIZE; i++)
        cout << str[i] << endl;
    return;
}
```

Referencing Single Characters in Elements of a String Array

```
/*Program to illustrate single character access*/
#include <iostream>
#include <string>
using namespace std;

const int SIZE = 5;

//function prototype
void printthearray(string str[]);

int main()
{
    string str[SIZE];

    str[0] = "Jones";
    str[1] = "Harrow";
    str[2] = "Tenenbaum";
    str[3] = "Arnow";
    str[4] = "Raphan";

    // use two-dimensional array access
    cout << str[2][6];           //prints 'b'
    cout << str[0][0];           //prints 'J'

    str[3][2] = 'r';             //str[3] == Arrow

    for (int i=0; i < str[1].length(); i++)
        cout << str[1][i];       //prints "Harrow"

    return 0;
}
```