C-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.

- A **C-string** is a sequence of characters stored in consecutive memory locations and terminated by a **null character** ('\0').

- **A string literal** is a sequence of characters enclosed by **double quotation marks.** (Note: a string literal automatically inserts the null character.) A string literal is a **string constant**.

- **Declaring C-string Variables:**
  A C-string is declared like an array of characters. You must leave room for the null character.

```
char name[21];          // this can hold a string up to length 20
```

- **Initializing a C-string:**

```
char first[10] = {'t','a','b','l','e','\0'};
char second[10] = "table";
```

- The **length** of a C-string is the number of characters stored in the string up to, but not including, the null character.

- The name of a C-string can be viewed as a **constant pointer** to a string.

- **Variable Pointer to a C-string:**

```
char *sptr;
```

- Legal Characters in a string literal:
  Each occurrence of double quotation marks or a backslash (or any other special character) must be preceded by the escape character (\) to tell the compiler that this is a character and not a control character.

```
char quotes[20] = "the \"king\" of rock";
char filename[20] = "c:\\hwork\\prob9.c";
```

● **Sample Program:**

```
//This program demonstrates that string literals
//are pointers to char
#include <iostream>
using namespace std;

int main()
{
   // Define variables that are pointers to char.
   char *p, *q;

   // Assign string literals to the pointers to char.
   p = "Hello ";
   q = "Bailey";

   // Print the pointers as C-strings.
   cout << p << q << endl;

   // Print the pointers as C-strings and as addresses.
   cout << p  << " is stored at " << int(p) << endl;
   cout << q  << " is stored at " << int(q) << endl;

    // A string literal can be treated as a pointer!
   cout << int ("Another string literal") << endl;

   return 0;
}
```

- **Initializing a String within the Code:**

```
char city[15];

city[0] = 'L';
city[1] = '.';
city[2] = 'A';
city[3] = '.';
city[4] = '\0';
```

- **A C-string as an Array of char:**

```
char c,d;
char str[5] = "wing";
char item[10] = "compater";

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

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

● **Sample Program:**

```
// This program cycles through a character array, displaying
// each element until a null terminator is encountered.
#include <iostream>
using namespace std;

int main()
{
   const int LENGTH = 80;   // Maximum length for string
   char line[LENGTH];       // Array of char

   // Read a string into the character array.
   cout << "Enter a sentence of no more than "
        << LENGTH-1 << " characters:\n";
   cin.getline(line, LENGTH);

   cout << endl << "The sentence you entered is:\n";

   // Loop through the array printing each character.
   for(int index = 0; line[index] != '\0'; index++)
   {
      cout << line[index];
   }
   cout << endl;

   return 0;
}
```

Dynamic Memory Allocation for C-Strings

- **Sample Program:**

```
// This program illustrates dynamic allocation
// of storage for C-strings.
#include <iostream>
using namespace std;

int main()
{
   const int NAME_LENGTH = 50;  // Maximum length
   char *pname;                 // Address of array

   // Allocate the array.
   pname = new char[NAME_LENGTH];

   // Read a string.
   cout << "Enter your name: ";
   cin.getline(pname, NAME_LENGTH);

   // Display the string.
   cout << endl << "Hello " << pname << endl;

   return 0;
}
```

C-String Library Functions

- **To use the C-string library functions:**
  #include <cstring> //or #include <string>

- **Assigning a Value to a C-string Variable - strcpy():**
  strcpy(*char *dest_str, char *source_str*);

  - copies the source_str to the dest_str.

- **Examples:**

```
char first[16];
char last[16];

strcpy(first,"Wayne Smith");
strcpy(last,first);

strcpy(last,&first[6]);
strcpy(last,first+6);
```

  Note: strcpy continues to copy until the first null character.

- **Determining the length of a C-string - strlen():**
  length = strlen(*char *str*);

  - returns the current length of str as an integer.

- **The sizeof Operator:**

```
size = sizeof item;           // returns size in bytes
   or
size = sizeof(item);          // returns size in bytes
   or
size = sizeof(typename);      // returns number of bytes
                              // allocated to that type
```

- **Examples:**

```
const int MAX = 30;
int length,size;
char dest[25];
char source[MAX];

cin.getline(source, MAX);
length = strlen(source)
size = sizeof dest;
if (length < size)
   strcpy(dest, source);
else
   cout << "won't fit" << endl;
```

- **Comparing Strings - strcmp():**

  result = strcmp(*char *first_str, char *second_str*);

  - strings are compared according to their ASCII values.
  - strings are compared character by character until either a null character or a difference in characters is found.
  - returns an integer result of the comparison:

    result > 0 if first_str > second_str
    result = 0 if first_str = = second_str
    result < 0 if first_str < second_str
  - A < Z < a < z
  - " " < "Car" < "Cat" < "car" < "cat" < "cats" < "cub"

- **Examples:**

```
int result;
char first[15] = "cat";
char second[15] = "car";

result = strcmp("cat","car");      // result > 0
result = strcmp("big","little");   // result < 0
result = strcmp("ABC","abc");      // result < 0
result = strcmp(" ab","ab");       // result < 0
result = strcmp("pre","prefix");   // result < 0
result = strcmp("potato","pot");   // result > 0
result = strcmp("cat","cat");      // result = = 0

result = strcmp(first,second);     // result > 0
result = strcmp(first,"catalog");  // result < 0

cin.getline(first,15);
cin.getline(second,15);
if (strcmp(first, second) = = 0)
   cout < < "they are equal" < endl;
```

- **Concatenating Two Strings - strcat():**
  strcat(*char *first_str, char *second_str*);

  - This functions concatenates the second string onto the end of the first string.

- **Example:**

```
char bigstr[1024];
char dest[30] = "computer";
char second[15] = " programming";

strcat(dest,second);        // dest = "computer programming"
strcpy(bigstr,dest);        // bigstr = "computer programming"
strcat(bigstr," is fun and very demanding");

// bigstr = "computer programming is fun and very demanding"
```

- **Example:**

```
char dest[30] = "computer";
char second[15] = " programming";

if (strlen(dest) + strlen(second) < sizeof(dest))
   strcat(dest, second);
else
   cout << "error: can't concatenate - dest too small\n";
```

Substring Functions

- **Comparing Substrings - strncmp():**
  result = strncmp(*char *addr_1, char *addr_2, numchars*);

  - Compares up to **numchars** from two strings, starting at the addresses specified (**addr_1 and addr_2**).

  - Returns an integer representing the relationship between the strings. (As per strcmp().)

  - Strings are compared character by character until numchars are compared or either a null character or a difference in characters is found.

- **Example:**

```
char first[30] = "strong";
char second[10] = "stopper";
int numchars;

if (strncmp(first,second,4) == 0)
   cout << "first four characters are alike\n";
else if (strncmp(first,second,4) < 0)
   cout << "first four characters of first string are less\n";
else
   cout << "first four characters of first string are more\n";

if (strncmp(&first[2],"ron",3) == 0)
   cout << "ron is found in position 2\n";
else
   cout << "ron is not found in position 2\n";
```

- **Copying a Substring - strncpy():**
  strncpy(*char *dest_str, char *source_str, numchars*);

  - copies numchars from the source_str to the dest_str.

- **Example:**

```
char dest[10];
char source[20] = "computers are fun";
char result[18] = "I have a cat";
char insert[10] = "big ";
int len;

strncpy(dest, source+3, 3);
dest[3] = '\0';                         // dest = "put"
cout << dest << endl;

len = strlen(insert);
strcpy(result+9+len,result+9);   // make room for insertion
strncpy(result+9,insert,len);    // result = "I have a big cat"
```

Arrays of C-Strings

- **Declaring an Array of C-Strings:**
  char *identifier*[*ARRAYSIZE*][*STRINGSIZE*];

- **Example:**

```
char   months[12][10]   =   {"January","February","March",
          "April","May","June","July","August","September",
          "October","November","December"};
int i;

for (i = 0; i < 12; i++)
   cout << months[i] << endl;
```

- **Example:**

```
char str[10][20];
int i=0;

while (cin.getline(str[i], 19) && i < 10) {
   cout << str[i] << endl;
   i++;
}
```

- **Example:**

```
char animal[3][12];
int I, result;

strcpy(animal[0],"giraffe");
strcpy(animal[1],"tiger");
strcpy(animal[2],"rhinoceros");
for (i = 0; i <= 2; i ++) {
   result = strcmp(animal[i],"tiger");
   if (result == 0) {
      << "tiger was found in position " << I << endl;
      break;
   }
}
```

- **Example:**

```
/* Function classify:
 *    finds monthname in array months and classifies its
 *    position into one of four seasons
 */
char *classify(char *monthname)
{
   char months[12][10] = {"January","February","March",
         "April","May","June","July","August","September",
         "October","November","December"};

   int i,found=0;

   for (i = 0; i <= 11 && !found; i++)
      if (strcmp(monthname,months[i]) == 0) found = 1;
   if (!found) return ("error");
   switch (i-1) {
      case 11:
      case 0:
      case 1:
         return ("winter");
      case 2:
      case 3:
      case 4:
         return ("spring");
      case 5:
      case 6:
      case 7:
         return ("summer");
      case 8:
      case 9:
      case 10:
         return ("autumn");
   }
}
```

- **Searching for a Substring - strstr():**
  strPtr = strstr(*char *string1, char *string2*);

  - Searches for the first occurrence of string2 within string1.

  - If found, returns a pointer to the position; otherwise, it returns the NULL pointer.

```
// This program uses the strstr function to search an array
// of strings for a name.
#include <iostream>
#include <string>                 // For using strstr
using namespace std;

int main()
{
   const int N_ITEMS = 5,        // Maximum number of items
             S_LENGTH = 31;   // maximum length of description

   // Array of product descriptions. - NOTE: 2 dimensional array
   char prods[5][S_LENGTH] = {
                              "TV327  31 inch Television",
                              "CD257  CD Player",
                              "TA677  Answering Machine",
                              "CS109  Car Stereo",
                              "PC955  Personal Computer"};


   char lookUp[S_LENGTH];       // For user input
   char *strPtr = NULL;          // Result from strstr

   // Get user input.
   cout << "\tProduct Database\n\n";
   cout << "Enter a product number to search for: ";
   cin.getline(lookUp, S_LENGTH);

   // Search for the string.
   int index = 0;
   while(index < N_ITEMS)
   {
      strPtr = strstr(prods[index], lookUp);
      if (strPtr != NULL)
         break;
      index++;
   }

   // Output the result of the search.
   if (strPtr == NULL)
      cout << "No matching product was found.\n";
   else
      cout << prods[index] << endl;

   return 0;
}
```

String/Numeric Conversion Functions

- **To use these library functions:**
  #include <cstdlib>

- **Selected Conversion Functions:**

<u>Function</u>	<u>Description</u>
atoi(numericStr)	converts a numeric string to an int
Example: num = atoi("4567");	
atol(numericStr)	converts a numeric string to a long integer
Example: num = atoi("500000");	
atof(numericStr)	converts a numeric string to a double
Example: num = atoi("3.14159");	
itoa(intVal, char *str, base)	converts an integer to a string
Example: itoa(185, myString, 16); //myString = “AF”	

  - Note: The “base” specifies the number system that the converted integer should be expressed in.

Character Testing

- **To use these library functions:**
  #include <cctype>

- **Selected Character Testing Functions:**

Function	Checks
isalpha(ch)	is the parameter alphabetic (A..Z or a..z)
isdigit(ch)	is the parameter a digit (0..9)
isalnum(ch)	is the parameter alphabetic or a digit
isspace(ch)	is the parameter a space (' ')
ispunct(ch)	is the parameter a punctuation mark
islower(ch)	is the parameter a lowercase alphabetic (a..z)
isupper(ch)	is the parameter an uppercase alphabetic (A..Z)
isprintr(ch)	is the parameter a printable character

  - All these functions return 1 if true and 0 if false.

- **Example:**

```
int ch;
int alpha=0,digit=0,space=0,punct=0;

while (cin >> ch) {
   if (isalpha(ch))
      alpha++;
   else if (isdigit(ch))
      digit++;
   else if (isspace(ch))
      space++;
   else if (ispunct(ch))
      punct++;
}
```

Character Case Conversion

- **To use these library functions:**
  #include <cctype>

- **The Functions toupper() & tolower():**
  chout = toupper(*ch*);

  chout = tolower(*ch*);

  - These functions force the case of the input character.
  - These functions have a return type if int.

- **Example:**

```
int ch;

do {
   ...
   cout << "Do you want to continue? (y/n): ";
   cin >> ch;
} while ((toupper(ch) == 'Y');
```

- **Example:**

```
char str[15];
int i=0;

strcpy(str,"alPHABetic");
while (str[i] != '\0') {
   str[i] = tolower(str[i]);
   i++;
}
```

Writing C-String Functions

- **The Function length():**
  - Write a function that returns the length of a string.

```
/* function to find and return length of a string */
int length(char *str)
{
    int i = 0;

    while (str[i] != '\0')
        i++;
    return(i);
}
```

- **The Function countchar():**
  - Write a function that counts how many times a particular character appears within a string.

```
/* function to find number of occurrences of a particular
 * character within a string
 */
int countchar(char str[], char let)
{
    int i=0, count=0;

    while (str[i] != '\0') {
        if (str[i] == let)
            count++;
        i++;
    }
    return(count);
}
```

- **The Function findchar():**
  - Write a function to determine the position of a particular character within a string.

```
/* function to return the position of a particular character
 * within a string. Returns -1 if not found.
 */
int findchar(char *str, char let)
{
   int i=0;

   while (str[i] != '\0') {
      if (str[i] == let)
         return(i);
      i++;
   }
   return(-1);
}
```

- **Alternate Code:**

```
int findchar(char *str, char let)
{
   int i=0, found=0;

   while (*(str+i) != '\0' && !found)
      if (*(str+i) == let)
         found = 1;
      else
         i++;
   if (!found)
      i = -1;
   return(i);
}
```

Functions that Return a Value of char *

- **The Function classify():**

```
/* classifies monthname into one of four seasons */
char *classify(char *monthname)
{
   if (strcmp(monthname, "December") == 0 ||
      strcmp(monthname, "January") == 0 ||
      strcmp(monthname, "February") == 0)
      return("winter");
   else if (strcmp(monthname, "March") == 0 ||
      strcmp(monthname, "April") == 0 ||
      strcmp(monthname, "May") == 0)
      return("spring");
   else if (strcmp(monthname, "June") == 0 ||
      strcmp(monthname, "July") == 0 ||
      strcmp(monthname, "August") == 0)
      return("summer");
   else if (strcmp(monthname, "September") == 0 ||
      strcmp(monthname, "October") == 0 ||
      strcmp(monthname, "November") == 0)
      return("fall");
   else
      return("error");
}
```

- **Calling Program:**

```
char *season;
char month[10];

cin.getline(month, 9);
season = classify(month);
if (strcmp(season,"error") != 0)
   cout << month “ is in the “ << season << endl;
else
   cout << month << " is not a valid month" << endl;
```

The C++ string Class

- The **string** class offers several advantages over C-strings:
  - There is a large body of member functions
  - Overloaded operators to simplify expressions

- **string Class Constructors:**

Constructor	**Description**
string str1;	Defines an empty string object str1
string str1("Jonn Doe");	Defines a string object str1 initialized to John Doe
string str1(str2);	Defines a string object str1 which is a copy of str2
string str1(str2, 5);	Defines a string object str1 which is initialized to the first 5 characters of the character array str2
string str1(str2, 1, 7);	Defines a string object str1 which is initialized with a substring of 7 characters from starting at index 1
string str1('z', 10);	Defines a string object str1 which is initialized to 10 'z' characters

- **string Class Overloaded Operators:**

Operator	**Description**
>>	Extraction operator
<<	Insertion operator
=	Assignment operator
+=	Append operator
+	Concatenation operator
[]	Array notation operator
<	Relational operators
>	
<=	
>=	
==	
!=	

- **string Class Member Functions: (SEE TEXTBOOK)**

Member Function	**Description**
str1.append(str2);	Appends str2 to str1
str1.append(str2,x,n);	Append n characters from index x
str1.append(str2,n);	Append first n characters of str2
str1.append(n,'z');	Append n copies of 'z'
str1.assign(str2);	Assign str2 to str1 (i.e., str1 = str2)
str1.assign(str2,x,n);	Assign n characters from index x
str1.assign(str2,n);	Assign first n characters of str2
str1.assign(n,'z');	Assign n copies of 'z' to str1
str1.at(x);	
str1.begin();	
str1.capacity();	
str1.clear();	
str1.compare(str2);	
str1.compare(x,n,str2);	
str1.copy(str2,x,n);	
str1.c_str();	
str1.data();	
str1.empty();	
str1.end();	
str1.erase(x,n);	
str1.find(str2,x);	
str1.find('z',x);	
str1.insert(x,str2);	
str1.insert(x,n,'z');	
str1.length();	
str1.replace(x,n,str2);	
str1.resize(n,'z');	
str1.size();	
str1.substr(x,n);	
str1.swap(str2);	

Creating Your Own string Class

- **Class Specification File:**

```
#ifndef MYSTRING_H
#define MYSTRING_H

#include <iostream>
#include <cstring>          // For string library functions
#include <cstdlib>          // For exit() function
using namespace std;

// The following declarations are needed by some compilers
class MyString;             // Forward declaration
ostream &operator<<(ostream &, MyString &);
istream &operator>>(istream &, MyString &);

// MyString class: An abstract data type
// for handling strings
class MyString
{
  private:
    char *str;
    int len;
  public:
    // Default constructor.
    MyString()
    {
        str = 0;
        len = 0;
    }

    // Convert and copy constructors.
    MyString(char *);
    MyString(MyString &);

    // Destructor.
    ~MyString()
    {
        if (len != 0)
            delete [] str;
        str = 0;
        len = 0;
    }
    // Various member functions and operators.
    int length() { return len; }
    char *getValue() { return str; };
    MyString operator+=(MyString &);
    MyString operator+=(const char *);
    MyString operator=(MyString &);
    MyString operator=(const char *);
    bool operator==(MyString &);
    bool operator==(const char *);
    bool operator!=(MyString &);
    bool operator!=(const char *);
    bool operator>(MyString &);
    bool operator>(const char *);
    bool operator<(MyString &);
    bool operator<(const char *);
    bool operator>=(MyString &);
    bool operator>=(const char*);
    bool operator<=(MyString &);
    bool operator<=(const char *);

    // Overload insertion and extraction operators.
    friend ostream &operator<<(ostream &, MyString &);
    friend istream &operator>>(istream &, MyString &);
};
#endif
```

● **Class Implementation File:**

```
#include "mystring.h"

//*********************************************
// Constructor to initialize the str member with a string constant.
//*********************************************
MyString::MyString(char *sptr)
{
   len = strlen(sptr);
   str = new char[len + 1];
   strcpy(str, sptr);
}

//*********************************************
// Copy constructor.
//*********************************************
MyString::MyString(MyString &right)
{
   str = new char[right.length() + 1];
   strcpy(str, right.getValue());
   len = right.length();
}

//*********************************************
// Overloaded = operator.
// Called when operand on the right is another MyString object.
// Returns the calling object.
//*********************************************
MyString MyString::operator=(MyString &right)
{
   if (len != 0)
      delete [] str;
   str = new char[right.length() + 1];
   strcpy(str, right.getValue());
   len = right.length();
   return *this;
}
```

```
//*********************************************
// Overloaded = operator.
// Called when operand on the right is a string.
// Returns the calling object.
//*********************************************
MyString MyString::operator=(const char *right)
{
   if (len != 0)
      delete [] str;
   len = strlen(right);
   str = new char[len + 1];
   strcpy(str, right);
   return *this;
}

//*******************************************
// Overloaded += operator.
// Called when operand on the right is another MyString object.
// Concatenates the str member of right to the str member of the
// calling object.
// Returns the calling object.
//*******************************************
MyString MyString::operator+=(MyString &right)
{
   char *temp = str;

   str = new char[strlen(str) + right.length() + 1];
   strcpy(str, temp);
   strcat(str, right.getValue());
   if (len != 0)
      delete [] temp;
   len = strlen(str);
   return *this;
}
```

```
//*******************************************
// Overloaded += operator.
// Called when operand on the right is a string. Concatenates the
// string right to the str member of the calling object.
// Returns the  the calling object.
//********************************************
MyString MyString::operator+=(const char *right)
{
   char *temp = str;

   str = new char[strlen(str) + strlen(right) + 1];
   strcpy(str, temp);
   strcat(str, right);
   if (len != 0)
      delete [] temp;
   return *this;
}

//**********************************************
// Overloaded == operator.
// Called when the operand on the right is a MyString object.
// Returns true if right.str is the same as str.
//**********************************************
bool MyString::operator==(MyString &right)
{
   return !strcmp(str, right.getValue());
}
//**********************************************
// Overloaded == operator.
// Called when the operand on the right is a string.
// Returns true if right is the same as str.
//**********************************************
bool MyString::operator==(const char *right)
{
   return !strcmp(str, right);
}
```

```
//*******************************************
// Overloaded != operator.
// Called when the operand on the right is a MyString object.
// Returns true if right.str is not equal to str.
//*******************************************
bool MyString::operator!=(MyString &right)
{
   return strcmp(str, right.getValue());
}

//*******************************************
// Overloaded != operator.
// Called when the operand on the right is a string.
// Returns true if right is not equal to str.
//*******************************************
bool MyString::operator!=(const char *right)
{
   return strcmp(str, right);
}

//*******************************************
// Overloaded != operator.
// Called when the operand on the right is a string.
// Returns true if str is greater than right.getValue.
//*******************************************
bool MyString::operator>(MyString &right)
{
   if (strcmp(str, right.getValue()) > 0)
      return true;
   else
      return false;
}
```

```
//**********************************************
// Overloaded > operator.
// Called when the operand on the right is a string.
// Returns true if str is greater than right.
//**********************************************
bool MyString::operator>(const char *right)
{
   if (strcmp(str, right) > 0)
      return true;
   else
      return false;
}

//**********************************************
// Overloaded < operator.
// Called when the operand on the right is a MyString object.
// Returns true if str is less than right.getValue.
//**********************************************
bool MyString::operator<(MyString &right)
{
   if (strcmp(str, right.getValue()) < 0)
      return true;
   else
      return false;
}

//**********************************************
// Overloaded < operator.
// Called when the operand on the right is a string.
// Returns true if str is less than right.
//**********************************************
bool MyString::operator<(const char *right)
{
   if (strcmp(str, right) < 0)
      return true;
   else
      return false;
}
```

```
//*********************************************************
// Overloaded >= operator.
// Called when the operand on the right is a MyString object.
// Returns true if str is greater than or equal to right.getValue.
//*********************************************************
bool MyString::operator>=(MyString &right)
{
   if (strcmp(str, right.getValue()) >= 0)
      return true;
   else
      return false;
}

//*********************************************************
// Overloaded >= operator.
// Called when the operand on the right is a string.
// Returns true if str is greater than or equal to right.
//*********************************************************
bool MyString::operator>=(const char *right)
{
   if (strcmp(str, right) >= 0)
      return true;
   else
      return false;
}

//*********************************************************
// Overloaded <= operator.
// Called when the operand on the right is a MyString object.
// Returns true if right.str is less than or equal  to str.
//*********************************************************
bool MyString::operator<=(MyString &right)
{
   if (strcmp(str, right.getValue()) <= 0)
      return true;
   else
      return false;
}
```

```
//*********************************************
// Overloaded <= operator.
// Called when the operand on the right is a string.
// Returns true if str is less than or equal to right.
//*********************************************
bool MyString::operator<=(const char *right)
{
   if (strcmp(str, right) <= 0)
      return true;
   else
      return false;
}

//*********************************************
// Overloaded stream insertion operator (<<).
//*********************************************
ostream &operator<<(ostream &strm, MyString &obj)
{
   strm << obj.str;
   return strm;
}

//*********************************************
// Overloaded stream extraction operator (>>).
//*********************************************
istream &operator>>(istream &strm, MyString &obj)
{
   // Buffer to read string in.
   const int MAX_LEN = 256;
   char buffer[MAX_LEN];

   // Read the string.
   strm.getline(buffer, MAX_LEN);

   // Invoke the convert constructor.
   obj = buffer;
   return strm;
}
```

● **Demo Program:**

```
// This program demonstrates the MyString class. Be
// sure to compile this program with mystring.cpp.
#include <iostream>
#include "mystring.h"
using namespace std;

int main()
{
   MyString object1("This"), object2("is");
   MyString object3("a test.");
   MyString object4 = object1;  // Call copy constructor.
   MyString object5("is only a test.");
   char string1[] = "a test.";

   cout << "object1: " << object1 << endl;
   cout << "object2: " << object2 << endl;
   cout << "object3: " << object3 << endl;
   cout << "object4: " << object4 << endl;
   cout << "object5: " << object5 << endl;

   cout << "string1: " << string1 << endl;

   object1 += " ";
   object1 += object2;
   object1 += " ";
   object1 += object3;
   object1 += " ";
   object1 += object4;
   object1 += " ";
   object1 += object5;
   cout << "object1: " << object1 << endl;

   return 0;
}
```