

## 10

# Characters and Strings

---

**PURPOSE**

1. To demonstrate the unique characteristics of character data
2. To view strings as an array of characters
3. To show how to input and output strings
4. To work with string functions

---

**PROCEDURE**

1. Students should read the Pre-lab Reading Assignment before coming to the lab.
2. Students should complete the Pre-lab Writing Assignment before coming to lab.
3. In the lab, students should complete labs assigned to them by the instructor.

Contents	Pre-requisites	Approximate completion time	Page number	Check when done
Pre-lab Reading Assignment		20 min.	176	
Pre-lab Writing Assignment	Pre-lab reading	10 min.	186	
<b>LESSON 10A</b>				
<b>Lab 10.1</b>				
Character Testing and String Validation	Pre-lab reading	15 min.	187	
<b>Lab 10.2</b>				
Case Conversion	Basic fundamental instructions	5 min.	190	
<b>Lab 10.3</b>				
Using <code>getline()</code> & <code>get()</code>	Basic knowledge of character arrays	30 min.	192	
<b>LESSON 10B</b>				
<b>Lab 10.4</b>				
String Functions— <code>strcat</code>	Basic knowledge of character arrays	15 min.	193	
<b>Lab 10.5</b>				
Student Generated Code Assignments	Basic knowledge of character arrays	35 min.	193	

## PRE-LAB READING ASSIGNMENT

---

### Character Functions

C++ provides numerous *functions* for character testing. These functions will test a single character and return either a non-zero value (true) or zero (false). For example, `isdigit` tests a character to see if it is one of the digits between 0 and 9. So `isdigit(7)` returns a non-zero value whereas `isdigit(y)` and `isdigit($)` both return 0. We will not list all the character functions here. A complete list may be found in the text. The following program demonstrates some of the others. Note that the `cctype` header file must be included to use the character functions.

#### *Sample Program 10.1:*

---

```
// This program utilizes several functions for character testing

#include <iostream>
#include <cctype>
using namespace std;

int main()
{
    char input;

    cout << "Please Enter Any Character:" << endl;
    cin >> input;
    cout << "The character entered is " << input << endl << endl;
    cout << "The ASCII code for " << input << " is " << int(input)
        << endl;

    if (isalpha(input))    // tests to see if character is a letter
    {
        cout << "The character is a letter" << endl;

        if (islower(input)) // tests to see if letter is lower case
            cout << "The letter is lower case" << endl;

        if (isupper(input)) // tests to see if letter is upper case
            cout << "The letter is upper case" << endl;
    }

    else if (isdigit(input)) // tests to see if character is a digit
        cout << "The character you entered is a digit" << endl;

    else
        cout << "The character entered is not a letter nor a digit"
            << endl;

    return 0;
}
```

---

In Lab 10.1 you will see a more practical application of character testing functions.

## Character Case Conversion

The C++ library provides the `toupper` and `tolower` functions for converting the case of a character. `toupper` returns the uppercase equivalent for a letter and `tolower` returns the lower case equivalent. For example, `cout << tolower('F');` causes an `f` to be displayed on the screen. If the letter is already lowercase, then `tolower` will return the value unchanged. Likewise, any non-letter argument is returned unchanged by `tolower`. It should be clear to you now what `toupper` does to a given character.

While the `toupper` and `tolower` functions are conceptually quite simple, they may not appear to be very useful. However, the following program shows that they do have beneficial applications.

### *Sample Program 10.2:*

---

```
// This program shows how the toupper and tolower functions can be
// applied in a C++ program

#include <iostream>
#include <cctype>
#include <iomanip>
using namespace std;

int main()
{
    int week, total, dollars;
    float average;
    char choice;

    cout << showpoint << fixed << setprecision(2);

    do
    {
        total = 0;
        for(week = 1; week <= 4; week++)
        {
            cout << "How much (to the nearest dollar) did you"
                << " spend on food during week " << week
                << " ?:" << endl;

            cin >> dollars;

            total = total + dollars;
        }
        average = total / 4.0;

        cout << "Your weekly food bill over the chosen month is $"
            << average << endl << endl;
    }
    {
        cout << "Would you like to find the average for "
            << "another month?";
    }
}
```

*continues*

```

        cout << endl << "Enter Y or N" << endl;
        cin >> choice;
    } while(toupper(choice) != 'Y' && toupper(choice) != 'N');

} while (toupper(choice) == 'Y');

return 0;
}

```

This program prompts the user to input weekly food costs, to the nearest dollar (an integer) for a four-week period. The average weekly total for that month is output. Then the user is asked whether they want to repeat the calculation for a different month. The flow of this program is controlled by a do-while loop. The condition `toupper(choice) == 'Y'` allows the user to enter 'Y' or 'y' for yes. This makes the program more user friendly than if we just allowed 'Y'. Note the second do-while loop near the end of the program. This loop also utilizes `toupper`. Can you determine the purpose of this second loop? How would the execution of the program be affected if we removed this loop (but left in the lines between the curly brackets)?

## String Constants

We have already talked about the character data type which includes letters, digits, and other special symbols such as \$ and @. Often we need to put characters together to form strings. For example, the price "\$1.99" and the phrase "one for the road!" are both strings of characters. The phrase contains blank space characters in addition to letters and an exclamation mark. In C++ a string is treated as a sequence of characters stored in consecutive memory locations. The end of the string in memory is marked by the null character `\0`. Do not confuse the null character with a sequence of two characters (i.e., `\` and `0`). The null character is actually an escape sequence. Its ASCII code is 0. For example, the phrase above is stored in computer memory as

o	n	e		f	o	r		t	h	e		r	o	a	d	!	\0
---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	---	---	----

A **string constant** is a string enclosed in double quotation marks. For example,

```

"Learn C++"
"What time is it?"
"Code Word 7dF#c&Q"

```

are all string constants. When they are stored in the computer's memory, the null character is automatically appended. The string "Please enter a digit" is stored as

P	l	e	a	s	e		e	n	t	e	r		a		d	i	g	i	t	\0
---	---	---	---	---	---	--	---	---	---	---	---	--	---	--	---	---	---	---	---	----

When a string constant is used in C++, it is the memory address that is actually accessed. In the statement

```
cout << "Please enter a digit";
```

the memory address is passed to the `cout` object. `cout` then displays the consecutive characters until the null character is reached.

## Storing Strings in Arrays

Often we need to access parts of a string rather than the whole string. For instance, we may want to alter characters in a string or even compare two strings. If this is the case, then a string constant is not what we need. Rather, a character array is the appropriate choice. When using character arrays, enough space to hold the null character must be allocated. For example:

```
char last[10];
```

This code defines a 10-element character array called `last`. However, this array can hold no more than 9 non-null characters since a space is reserved for the null character. Consider the following:

```
char last[10];
cout << "Please enter your last name using no more than 9 letters";
cin >> last;
```

If the user enters `Symon`, then the following will be the contents of the `last` array:

S	y	m	o	n	\0
---	---	---	---	---	----

Recall that the computer actually sees `last` as the beginning address of the array. There is a problem that can arise when using the `cin` object on a character array. `cin` does not “know” that `last` has only 10 elements. If the user enters `Newmanouskous` after the prompt, then `cin` will write past the end of the array. We can get around this problem by using the `getline` function. If we use

```
cin.getline(last,10)
```

then the computer knows that the maximum length of the string, including the null character, is 10. Consequently, `cin` will read until the user hits `ENTER` or until 9 characters have been read, whichever occurs first. Once the string is in the array, it can be processed character by character. In this next section we will see a program that uses `cin.getline()`.

## Library Functions for Strings

The C++ library provides many functions for testing and manipulating strings. For example, to determine the length of a given string one can use the `strlen` function. The syntax is shown in the following code:

```
char line[40] = "A New Day";
int length;
length = strlen(line);
```

Here `strlen(line)` returns the length of the string including white spaces but not the null character at the end. So the value of `length` is 9. Note this is smaller than the size of the actual array holding the string.

To see why we even need a function such as `strlen`, consider the problem of reading in a string and then writing it backwards. If we only allowed strings of a fixed size, say length 29 for example, then the task would be easy. We simply read the string into an array of size 30 or more. Then write the 28<sup>th</sup> entry followed by the 27<sup>th</sup> entry and so on, until we reach the 0<sup>th</sup> entry. However, what if we wish to allow the user to input strings of different lengths? Now it is unclear where the end of the string is. Of course, we could search the array until we find

the null character and then figure out what position it is in. But this is precisely what the `strlen` function does for us. Sample Program 10.3 is a complete program that performs the desired task.

*Sample Program 10.3:*

---

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char line[50];
    int length, count = 0;

    cout << "Enter a sentence of no more than 49 characters:\n";
    cin.getline(line, 50);

    length = strlen(line);    // strlen returns the length of the
                             // string currently stored in line

    cout << "The sentence entered read backwards is:\n";

    for(count = length-1; count >= 0; count--)
    {
        cout << line[count];
    }

    cout << endl;
    return 0;
}
```

---

*Sample Run 1:*

```
Enter a sentence of no more than 49 characters:
luaP deiruB I
The sentence you entered printed backwards is:
I Buried Paul
```

*Sample Run 2:*

```
Enter a sentence of no more than 49 characters:
This sentence is too long to hold a mere 49 characters!
The sentence you entered printed backwards is:
arabc 94 erem a dlhc ot gnol oot si ecnetnes sihT
```

Another useful function for strings is `strcat`, which concatenates two strings. `strcat(string1, string2)` attaches the contents of `string2` to the end of `string1`. The programmer must make sure that the array containing `string1` is large enough to hold the concatenation of the two strings plus the null character.

Consider the following code:

```
char string1[25] = "Total Eclipse ";    // note the space after the second
                                       // word - strcat does not insert a
char string2[11] = "of the Sun";      // space. The programmer must do this.

cout << string1 << endl;
cout << string2 << endl;

strcat(string1, string2);

cout << string1 << endl;
```

These statements produce the following output:

```
Total Eclipse of
the Sun
Total Eclipse of the Sun
```

What would have happened if we had defined `string1` to be a character array of size 20?

There are several other string functions such as `strcpy` (copies the second string to the first string), `strcmp` (compares two strings to see if they are the same or, if not, which string is alphabetically greater than the other), and `strstr` (looks for the occurrence of a string inside of another string). Note that C-string functions require the `cstring` header file. For more details on these string functions and the others, see the text.

## The get and ignore functions

There are several ways of inputting strings. We could use the standard `>>` extraction operator for a character array or string class object. However, we know that using `cin >>` skips any leading whitespace (blanks, newlines). It will also stop at the first trailing whitespace character. So, for example, the name “John Wayne” cannot be read as a single string using `cin >>` because of a blank space between the first and last names. We have already seen the `getline` function which does allow blank spaces to be read and stored. In this section we will introduce the `get` and `ignore` functions, which are also useful for string processing.

The `get` function reads in the next character in the input stream, including whitespace. The syntax is

```
cin.get(ch);
```

Once this function call is made, the next character in the input stream is stored in the variable `ch`. So if we want to input

```
$ X
```

we can use the following:

```
cin.get(firstChar);
cin.get(ch);
cin.get(secondChar);
```

where `firstChar`, `ch`, and `secondChar` are all character variables. Note that after the second call to the `get` function, the blank character is stored in the variable `ch`.

The `get` function, like the `getline` function, can also be used to read strings. In this case we need two parameters:

```
cin.get(strName, numChar+1);
```

Here `strName` is a string variable and the integer expression `numChar+1` gives the number of characters that may be read into `strName`.

Both the `getline` and the `get` functions do not skip leading whitespace characters. The `get` statement above brings in the next input characters until it either has read `numChar+1` characters or it reaches the newline character `\n`. However, the newline character is not stored in `strName`. The null character is then appended to the end of the string. Since the newline character is not **consumed** (not read by the `get` function), it remains part of the input characters yet to be read.

*Example:*

```
char strName[21];
cin.get(strName, 21);
```

Now suppose we input

**John Wayne**

Then “John Wayne” is stored in `strName`. Next input

**My favorite westerns star John Wayne**

In this case the string “My favorite westerns” is stored in `strName`.

We often work with records from a file that contain character data followed by numeric data. Look at the following data which has a name, hours worked, and pay rate for each record stored on a separate line.

*Pay Roll Data*

John Brown	7	12.50
Mary Lou Smith	12	15.70
Dominic DeFino	8	15.50

Since names often have imbedded blank spaces, we can use the `get` function to read them. We then use an integer variable to store the number of hours and a floating point variable to store the pay rate. At the end of each line is the `'\n'` character. Note that the end of line character is not consumed by reading the pay rate and, in fact, is the next character to be read when reading the second name from the file. This creates problems. Whenever we need to read through characters in the input stream without storing them, we can use the `ignore` function. This function has two arguments, the first is an integer expression and the second is a character expression. For example, the call

```
cin.ignore(80, '\n');
```

says to skip over the next 80 input characters but stop if a newline character is read. The newline character is consumed by the `ignore` function. This use of `ignore` is often employed to find the end of the current input line.



The following program will read the sample pay roll data from a file called `payRoll.dat` and show the result to the screen. Note that the input file must have names that are no longer than 15 characters and the first 15 positions of each line are reserved for the name. The numeric data must be after the 15th position in each line.

*Sample Program 10.4:*

---

```
#include <fstream>
#include <iostream>
using namespace std;

const int MAXNAME = 15;

int main()
{
    ifstream inData;

    inData.open("payRoll.dat");
    char name[MAXNAME+1];
    int hoursWorked;
    float payRate;

    inData.get(name,MAXNAME+1); // prime the read
    while (inData)
    {
        inData >> hoursWorked;
        inData >> payRate;

        cout << name << endl;
        cout << "Hours Worked " << hoursWorked << endl;
        cout << "Pay Rate " << payRate << " per hour"
            << endl << endl;

        inData.ignore(80,'\n');
        // This will ignore up to 80 characters but will
        // stop (ignoring) when it reads the \n which is
        // consumed.

        inData.get(name,MAXNAME+1);

    }

    return 0;
}
```

---

*Summary of types of input for strings:*

```

cin >> strName;           // skips leading whitespace. Stops at the first
                          // trailing whitespace (which is not consumed)

cin.get(strName, 21);    // does not skip leading whitespace
                          // stops when either 20 characters are read or
                          // '\n' is encountered (which is not consumed)

cin.ignore(200, '\n');   // ignores at most 200 characters but stops if
                          // newline (which is consumed) is encountered

```

**Pointers and Strings**

Pointers can be very useful for writing string processing functions. If one needs to process a certain string, the beginning address can be passed with a pointer variable. The length of the string does not even need to be known since the computer will start processing using the address and continue through the string until the null character is encountered.

Sample Program 10.5 below reads in a string of no more than 50 characters and then counts the number of letters, digits, and whitespace characters in the string. Notice the use of the pointer `strPtr`, which points to the string being processed. The three functions `countLetters`, `countDigits`, and `countWhiteSpace` all perform basically the same task—the while loop is executed until `strPtr` points to the null character marking the end of the string. In the `countLetters` function, characters are tested to see if they are letters. The `if(isalpha(*strPtr))` statement determines if the character pointed at by `strPtr` is a letter. If so, then the counter `occurs` is incremented by one. After the character has been tested, `strPtr` is incremented by one to test the next character. The other two functions are analogous.

*Sample Program 10.5:*


---

```

#include <iostream>
#include <cctype>

using namespace std;

//function prototypes
int countLetters(char*);
int countDigits(char*);
int countWhiteSpace(char*);

int main()
{
    int numLetters, numDigits, numWhiteSpace;
    char inputString[51];

    cout <<"Enter a string of no more than 50 characters: "
         << endl << endl;

```

```

cin.getline(inputString,51);

numLetters = countLetters(inputString);
numDigits = countDigits(inputString);
numWhiteSpace = countWhiteSpace(inputString);

cout << "The number of letters in the entered string is "
      << numLetters << endl;
cout << "The number of digits in the entered string is "
      << numDigits << endl;
cout << "The number of white spaces in the entered string is "
      << numWhiteSpace << endl;

return 0;
}
//*****
//          countLetters
//
// task:          This function counts the number of letters
//                (both capital and lower case) in the string
// data in:       pointer that points to an array of characters
// data returned: number of letters in the array of characters
//
//*****

int countLetters(char *strPtr)
{
    int occurs = 0;
    while(*strPtr != '\0')    // loop is executed as long as
                            // the pointer strPtr does not point
                            // to the null character which
                            // marks the end of the string
    {
        if (isalpha(*strPtr)) // isalpha determines if
                              // the character is a letter
            occurs++;
        strPtr++;
    }
    return occurs;
}
//*****
//          countDigits
//
// task:          This function counts the number of digits
//                in the string
// data in:       pointer that points to an array of characters
// data returned: number of digits in the array of characters
//
//*****

```

*continues*

```

int countDigits(char *strPtr)
{
    int occurs = 0;
    while(*strPtr != '\0')
    {
        if (isdigit(*strPtr)) // isdigit determines if
                               // the character is a digit
            occurs++;
        strPtr++;
    }
    return occurs;
}

//*****
//                               countWhiteSpace
//
// task:           This function counts the number of whitespace
//                characters in the string
// data in:        pointer that points to an array of characters
// data returned:  number of whitespaces in the array of
//                characters
//
//*****

int countWhiteSpace(char *strPtr) // this function counts the
                                // number of whitespace characters.
                                // These include, space, newline,
                                // vertical tab, and tab
{
    int occurs = 0;
    while(*strPtr != '\0')
    {
        if (isspace(*strPtr)) // isspace determines if
                               // the character is a
                               // whitespace character
            occurs++;
        strPtr++;
    }
    return occurs;
}

```

---

## PRE-LAB WRITING ASSIGNMENT

---

### Fill-in-the-Blank Questions

1. The code `cout << toupper('b');` causes a \_\_\_\_\_ to be displayed on the screen.
2. The data type returned by `isalpha('g')` is \_\_\_\_\_.

3. After the assignment statement `result = isdigit('$')`, `result` has the value \_\_\_\_\_.
4. The code `cout << tolower('#');` causes a \_\_\_\_\_ to be displayed on the screen.
5. The end of a string is marked in computer memory by the \_\_\_\_\_.
6. In `cin.getline(name, 25)`, the 25 indicates that the user can input at most \_\_\_\_\_ characters into `name`.
7. Consider the following:

```
char message[35] = "Like tears in the rain";
int length;
length = strlen(message);
```

Then the value of `length` is \_\_\_\_\_.

8. Consider the code
 

```
char string1[30] = "In the Garden";
char string2[15] = "of Eden";
strcat(string1, string2);
cout << string1;
```

The output for this is \_\_\_\_\_.
9. The \_\_\_\_\_ header file must be included to access the `islower` and `isspace` character functions.
10. In C++, a string constant must be enclosed in \_\_\_\_\_ whereas a character constant must be enclosed in \_\_\_\_\_.

## LESSON 10

---

### LAB 10.1 Character Testing and String Validation

The American Equities investment company offers a wide range of investment opportunities ranging from mutual funds to bonds. Investors can check the value of their portfolio from the American Equities' web page. Information about personal portfolios is protected via encryption and can only be accessed using a password. The American Equities company requires that a password consist of 8 characters, 5 of which must be letters and the other 3 digits. The letters and digits can be arranged in any order. For example,

```
rt56AA7q
123actyN
1Lo0Dwa9
myNUM741
```

are all valid passwords. However, the following are all invalid:

```
the476NEw // It contains more than 8 characters (also more than 5
           // letters)
be68moon // It contains less than 3 digits.
$retrn99 // It contains only 2 digits and has an invalid character ('$')
```

American Equities needs a program for their web page that determines whether or not an entered password is valid. The program `american_equities.cpp` from the Lab 10 folder performs this task. The code is the following:

---

```
// This program tests a password for the American Equities
// web page to see if the format is correct

// Place Your Name Here

#include <iostream>
#include <cctype>
#include <cstring>

using namespace std;

//function prototypes

bool testPassWord(char[]);
int countLetters(char*);
int countDigits(char*);

int main()
{
    char passWord[20];

    cout << "Enter a password consisting of exactly 5 "
         << "letters and 3 digits:" << endl;
    cin.getline(passWord,20);

    if (testPassWord(passWord))
        cout << "Please wait - your password is being verified" << endl;
    else
    {
        cout << "Invalid password. Please enter a password "
             << "with exactly 5 letters and 3 digits" << endl;
        cout << "For example, my37RuN9 is valid" << endl;
    }

    // Fill in the code that will call countLetters and
    // countDigits and will print to the screen both the number of
    // letters and digits contained in the password.

    return 0;
}
```

```

/*****
//
//          testPassWord
//
// task:      determines if the word in the
//            character array passed to it, contains
//            exactly 5 letters and 3 digits.
// data in:   a word contained in a character array
// data returned: true if the word contains 5 letters & 3
//            digits, false otherwise
//
/*****
bool testPassWord(char custPass[])
{
    int numLetters, numDigits, length;

    length = strlen(custPass);
    numLetters = countLetters(custPass);
    numDigits = countDigits(custPass);
    if (numLetters == 5 && numDigits == 3 && length == 8 )
        return true;
    else
        return false;
}

// the next 2 functions are from Sample Program 10.5
/*****
//
//          countLetters
//
// task:      counts the number of letters (both
//            capital and lower case)in the string
// data in:   a string
// data returned: the number of letters in the string
//
/*****
int countLetters(char *strPtr)
{
    int occurs = 0;

    while(*strPtr != '\0')

    {
        if (isalpha(*strPtr))
            occurs++;
        strPtr++;
    }

    return occurs;
}

```

*continues*

```

//*****
//                                countDigits
//
// task:                counts the number of digits in the string
// data in:              a string
// data returned:       the number of digits in the string
//
//*****
int countDigits(char *strPtr)
{
    int occurs = 0;

    while(*strPtr != '\0')
    {
        if (isdigit(*strPtr)) // isdigit determines if
                               // the character is a digit
            occurs++;
        strPtr++;
    }

    return occurs;
}

```

---

*Exercise 1:* Fill in the code in bold and then run the program several times with both valid and invalid passwords. Read through the program and make sure you understand the logic of the code.

*Exercise 2:* Alter the program so that a valid password consists of 10 characters, 6 of which must be digits and the other 4 letters.

*Exercise 3:* Adjust your program from Exercise 2 so that only lower case letters are allowed for valid passwords.

---

## LAB 10.2 Case Conversion

Bring in `case_convert.cpp` from the Lab 10 folder. Note that this is Sample Program 10.2. The code is the following:

---

```

// This program shows how the toupper and tolower functions can be
// applied in a C++ program

// PLACE YOUR NAME HERE

#include <iostream>
#include <cctype>
#include <iomanip>
using namespace std;

int main()
{
    int week, total, dollars;
    float average;
    char choice;

```



```

cout << showpoint << fixed << setprecision(2);

do
{
    total = 0;
    for(week = 1; week <= 4; week++)
    {
        cout << "How much (to the nearest dollar) did you"
            << " spend on food during week " << week
            << " ?:" << endl;
        cin >> dollars;

        total = total + dollars;
    }
    average = total / 4.0;

    cout << "Your weekly food bill over the chosen month is $"
        << average << endl << endl;
    do
    {
        cout << "Would you like to find the average for "
            << "another month?";
        cout << endl << "Enter Y or N" << endl;
        cin >> choice;
    } while(toupper(choice) != 'Y' && toupper(choice) != 'N');

    } while (toupper(choice) == 'Y');

    return 0;
}

```

---

*Exercise 1:* Run the program several times with various inputs.

*Exercise 2:* Notice the following do-while loop which appears near the end of the program:

```

do
{
    cout << "Would you like to find the average for another month?";
    cout << endl << "Enter Y or N" << endl;
    cin >> choice;
} while(toupper(choice) != 'Y' && toupper(choice) != 'N');

```

How would the execution of the program be different if we removed this loop? Try removing the loop but leave the following lines in the program:

```

cout << "Would you like to find the average for another month?";
cout << endl << "Enter Y or N" << endl;
cin >> choice;

```

Record what happens when you run the new version of the program.

*Exercise 3:* Alter program `case_convert.cpp` so that it performs the same task but uses `tolower` rather than `toupper`.

**LAB 10.3 Using `getline()` & `get()`**

*Exercise 1:* Write a short program called `readata.cpp` that defines a character array `last` which contains 10 characters. Prompt the user to enter their last name using no more than 9 characters. The program should then read the name into `last` and then output the name back to the screen with an appropriate message. Do not use the `getline()` or `get` functions!

*Exercise 2:* Once the program in Exercise 1 is complete, run the program and enter the name **Newmanouskous** at the prompt. What, if anything, happens? (Note that the results could vary depending on your system).

*Exercise 3:* Re-write the program above using the `getline()` function (and only allowing 9 characters to be input). As before, use the character array `last` consisting of 10 elements. Run your new program and enter **Newmanouskous** at the prompt. What is the output?

*Exercise 4:* Bring in program `grades.cpp` and `grades.txt` from the Lab 10 folder. Fill in the code in bold so that the data is properly read from `grades.txt`. and the desired output to the screen is as follows:

OUTPUT TO SCREEN		DATA FILE	
<b>Adara Starr</b>	<b>has a(n) 94 average</b>	<b>Adara Starr</b>	<b>94</b>
<b>David Starr</b>	<b>has a(n) 91 average</b>	<b>David Starr</b>	<b>91</b>
<b>Sophia Starr</b>	<b>has a(n) 94 average</b>	<b>Sophia Starr</b>	<b>94</b>
<b>Maria Starr</b>	<b>has a(n) 91 average</b>	<b>Maria Starr</b>	<b>91</b>
<b>Danielle DeFino</b>	<b>has a(n) 94 average</b>	<b>Danielle DeFino</b>	<b>94</b>
<b>Dominic DeFino</b>	<b>has a(n) 98 average</b>	<b>Dominic DeFino</b>	<b>98</b>
<b>McKenna DeFino</b>	<b>has a(n) 92 average</b>	<b>McKenna DeFino</b>	<b>92</b>
<b>Taylor McIntire</b>	<b>has a(n) 99 average</b>	<b>Taylor McIntire</b>	<b>99</b>
<b>Torrie McIntire</b>	<b>has a(n) 91 average</b>	<b>Torrie McIntire</b>	<b>91</b>
<b>Emily Garrett</b>	<b>has a(n) 97 average</b>	<b>Emily Garrett</b>	<b>97</b>
<b>Lauren Garrett</b>	<b>has a(n) 92 average</b>	<b>Lauren Garrett</b>	<b>92</b>
<b>Marlene Starr</b>	<b>has a(n) 83 average</b>	<b>Marlene Starr</b>	<b>83</b>
<b>Donald DeFino</b>	<b>has a(n) 73 average</b>	<b>Donald DeFino</b>	<b>73</b>

The code of `grades.cpp` is as follows:

---

```
#include <fstream>
#include <iostream>
using namespace std;

// PLACE YOUR NAME HERE

const int MAXNAME = 20;

int main()
{
    ifstream inData;
    inData.open("grades.txt");

    char name[MAXNAME + 1]; // holds student name
    float average;          // holds student average
```

```

inData.get(name,MAXNAME+1);
while (inData)
{
    inData >> average;
    // Fill in the code to print out name and
    // student average

    // Fill in the code to complete the while
    // loop so that the rest of the student
    // names and average are read in properly
}

return 0;
}

```

---

#### LAB 10.4 String Functions—`strcat`

Consider the following code:

```

char string1[25] ="Total Eclipse ";
char string2[11] = "of the Sun";
cout << string1 << endl;
cout << string2 << endl;
strcat(string1,string2);
cout << string1 << endl;

```

*Exercise 1:* Write a complete program including the above code that outputs the concatenation of `string1` and `string2`. Run the program and record the result.

*Exercise 2:* Alter the program in Exercise 1 so that `string1` contains 20 characters rather than 25. Run the program. What happens?

#### LAB 10.5 Student Generated Code Assignments

*Exercise 1:* A **palindrome** is a string of characters that reads the same forwards as backwards. For example, the following are both palindromes:

1457887541      madam

Write a program that prompts the user to input a string of a size 50 characters or less. Your program should then determine whether or not the entered string is a palindrome. A message should be displayed to the user informing them whether or not their string is a palindrome.

*Exercise 2:* The `strcmp(string1,string2)` function compares `string1` to `string2`. It is a value returning function that returns a negative integer if `string1 < string2`, 0 if `string1 == string2`, and a positive integer if `string1 > string2`. Write a program that reads two names (last name first followed by a comma followed by the first name) and then prints them in alphabetical order. The two names should be stored in separate character arrays holding a maximum of 25 characters each. Use the `strcmp()` function to make the comparison of the two names. Remember that 'a' < 'b', 'b' < 'c', etc. Be sure to include the proper header file to use `strcmp()`.

*Sample Run 1:*

**Please input the first name**

Brown, George

**Please input the second name**

Adams, Sally

**The names are as follows:**

**Adams, Sally**

**Brown, George**

*Sample Run 2:*

**Please input the first name**

Brown, George

**Please input the second name**

Brown, George

**The names are as follows:**

**Brown, George**

**Brown, George**

**The names are the same**

*Exercise 3:* (Optional) Write a program that determines how many consonants are in an entered string of 50 characters or less. Output the entered string and the number of consonants in the string.