STRINGS

The `string` Data Type

- To use the `string` class, a program must contain the `#include <string>` directive
- You can use the `string` class to create `string` variables or `string` named constants

The `string` Data Type

- Example

```
string zipCode = "";
```

  - Declare and Initializes a `string` variable named `zipCode`
- Example

```
const string VALID_LENGTH = "Valid Length";
```

  - Declares and Initializes a `string` named constant called `VALID_LENGTH`
- Example

```
const string COMPANY_NAME = "ABC Company";
```

  - Declares and Initializes a `string` named constant called `COMPANY_NAME`

The `getline` Function

- The `getline` function obtains string data from the keyboard and stores it in a `string` variable

- Syntax is:

```
getline(cin, stringVariableName, [delimiterCharacter]);
```

- Three actual arguments (first two required):
  - `cin` argument refers to computer keyboard
  - *stringVariableName* argument is name of a `string` variable in which to store input
  - Optional *delimiterCharacter* indicates character that immediately follows last character of string

The `getline` Function (delimiter character)

- Function will continue to read characters entered at keyboard until it encounters a delimiter character
- Default delimiter character is newline character { "\n" }
- When the function encounters a delimiter character, it discards the character—process called **consuming the character**

  - Backslash is called an escape character
  - Backslash and character following it are called an escape sequence

The “Creative” Sales Program

```cpp
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

int main (){

    const double RATE = .1;
    string name = "";
    int sales = 0;
    double bonus = 0.0;

    cout << "\nSalesperson's name: ";
    getline (cin, name);
    cout << "Sales: $";
    cin >> sales;

    bonus = sales * RATE;

    cout << fixed << setprecision(2);
    cout << "Bonus for " << name << ": $" << bonus << endl << endl;

}
```

- Program that enters a salesperson’s name and sales amount; It calculates the salesperson’s bonus by multiplying the sales amount by 10%.

```
Salesperson's name: Jessica L. Sizemore
Sales: $256445
Bonus for Jessica L. Sizemore: $25644.50
```

The `ignore` Function

- The **`ignore` function** instructs the computer to read and ignore characters stored in the `cin` object by consuming (discarding) them

- Syntax is:

```
cin.ignore([numberOfCharacters][, delimiterCharacter]);
```

- Has two actual arguments, both optional:
  - `numberOfCharacters` argument is maximum number of characters function should consume (default is 1)
  - `delimiterCharacter` argument stops `ignore` function from reading and discarding any more characters when consumed

The `ignore` Function

- Example One

```
cin.ignore();
```

  - Reads and consumers one character; also written as `cin.ignore(1)`

- Example Two

```
cin.ignore(5);
```

  - Reads and consumes five characters

- Example Three

```
cin.ignore(100, '\n')
```

  - Reads and consumes until either 100 characters or newline characters are consumed

- Example Four

```
Cin.ignore(25, '#');
```

  - Reads and consumes until either 25 characters or the # character is consumed

Number of Characters Contained in a `string` Variable

- You use `string` class's **`length` function** to determine the number of characters in a string variable

- Syntax is:

  `string``.length()`

  - Returns number of characters contained in `string`

- Example

```
string name = "Nancy Haberdeen";
cout << name.length() << endl;
```

  - Displays the number 15 on the screen

Number of Characters Contained in a `string` Variable

```
5   int main () {
6
7       const string VALID_MSG = "Valid Length";
8       const string INVALID_MSG = "Invalid Length";
9       string exmpString = "";
10
11      cout << "Enter a string of 5 characters: ";
12      cin >> exmpLength;
13
14      if (exmpString.length() == 5){
15          cout << VALID_MSG << endl;
16      } else {
17          cout << INVALID_MSG << endl;
18      }
```

- Compares the number of characters stored in the `exmpLength` variable with the number 5 and then displays an appropriate message

Accessing the Characters Contained in a `string` Variable

- The **`substr` function** allows you to access any number of characters contained in a `string` variable by returning the specified characters

- Syntax is:

```
string.substr(subscript[, count])
```

- Has two arguments (first is required):
  - `subscript` argument represents subscript of first character you want to access in `string`
  - `count` argument is number of characters to return after character specified by `subscript`

Accessing the Characters Contained in a `string` Variable

- If you omit `count` argument, function returns all characters from `subscript` position through end of `string`
- Each character has a unique `subscript` that indicates character's position in the `string`

Accessing the Characters Contained in a `string` Variable

```
 6  int main () {
 7
 8      string name = "Jack Blackfeather";
 9      string first = "";
10      string last = "";
11      first = name.substr(0, 4);
12      last = name.substr(5);
13  }
```

- Assigns “Jake” to the `first` variable and “Blackfeather” to the `last` variable

Accessing the Characters Contained in a `string` Variable

```
15      string sales = "";
16      cout << "Enter the sales: ";
17      getline(cin, sales);
18
19      if (sales.substr(0, 1) == "$"){
20
21          sales = sales.substr(1);
22
23      }
```

- If the string stored in the `sales` variable begins with $ the code assigns the variable's contents, excluding the $ to the variable

Accessing the Characters Contained in a `string` Variable

```
26      string rate = "";
27      cout << "Enter the rate: ";
28      getline(cin, rate);
29
30      if (rate.substr(rate.length() - 1, 1) == "%") {
31
32          rate = rate.substr(0, rate.length() - 1);
33
34      }
```

- If the string stored in the `rates` variable ends with the % the code assigns the variable's contents, excluding the % to the variable

Searching the Contents of a `string` Variable

- You use the **`find` function** to search contents of a `string` variable to determine whether it contains a specific sequence of characters

- Syntax is:

```
string.find(searchString, subscript)
```

  - searchString argument is a string for which you are searching within string
  - subscript argument specifies starting position for the search

Searching the Contents of a `string` Variable

- The `find` function performs a case-sensitive search (uppercase and lowercase letters are not equivalent)
  - When `searchString` is contained within `string`, function returns an integer that indicates beginning position of `searchString` within `string`
  - Function returns -1 when `searchString` is not contained within `string`

Rearranged Name Program

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main () {
7
8      string firstLast = "";
9      string first = "";
10     string last = "";
11     int spaceLocation = 0;
12
13     cout << "Name (first last): ";
14     getline(cin, firstLast);
15
16     spaceLocation = firstLast.find(" ", 0);
17     first = firstLast.substr(0, spaceLocation);
18     last = firstLast.substr(spaceLocation + 1);
19
20     cout << last << ", " << first << endl;
21 }
```

```
Name (first last): Jessica Sizemore
Sizemore, Jessica
```

- Gets first and last name
- Locate space, then pulls out first and last names
- Displays rearranged name

Removing Characters from a `string` Variable

- You can use the **`erase` function** to remove one or more characters from a string variable

- Syntax is:

```
string.erase(subscript[, count]);
```

  - `subscript` is position of first character you want to remove
  - Optional `count` argument is an integer that specifies number of characters you want removed
  - If you omit `count`, function removes all characters from `subscript` through end of string

Removing Characters from a `string` Variable

```
 8
 9      string place = "Salem, Oregon";
10      place.erase(0, 7);
11
12  /*--------------------------------------*/
13
14      string place = "Salem, Oregon";
15      place.erase(5);
16
```

- Removes the first seven characters from the `place` variables, changing the variable's contents to "Oregon"
- Removes all of the characters rom the `places` variable, beginning with the character whose subscript is 5, changing the variable's contents to "Salem"

Replacing Characters in a `string` Variable

- The **`replace` function** replaces one sequence of characters in a string variable with another

- Syntax is:

```
string.replace(subscript, count, replacementString);
```

  - `subscript` argument specifies where to begin replacing characters in `string`
  - `count` argument indicates number of characters to replace
  - `replacementString` argument contains replacement characters

Replacing Characters in a `string` Variable

```
18
19      string phone = "1-800-111-0000";
20      phone.replace(2, 3, "877");
21
22  /*--------------------------------*/
23
24      string name = "Karena Wilson";
25      name.replace(7, 6, "Farley");
26
```

- Replaces three characters in the `phone` variable, beginning with the character whose subscript is 2 with “877”; changes are made to be “1-877-111-0000’
- Replace six characters in thee `name` variable, beginning with the characters whose subscript is 7, with “Farley”; changes are made to be “Karena Farley”

Inserting Characters within a `string` Variable

- You can use the **`insert` function** to insert characters into a `string` variable

- Syntax is:

```
string.insert(subscript, insertString);
```

  - `subscript` specifies where in `string` you want characters inserted
  - `insertString` specifies characters to be inserted

Inserting Characters within a `string` Variable

```
29
30      string SSN = "111220000";
31      SSN.insert(3, "-");
32      SSN.insert(6, "-");
33
34  /*-------------------------------------*/
35
36      string name = "Harold Cruthers";
37      name.insert(7, "G. ");
38
```

- Insert how " – " in the `SSN`, one after the third number and another after the fifth number; changes to be "111-22-0000"
- Inserts "G. " between the first and last names stores in the `name` variable; changes to be "Harold G. Cruthers"

Concatenating Strings

- **String concatenation** refers to the process of connecting (linking) strings together
- You concatenate strings using the **concatenation operator** (+ sign)

Concatenating Strings

```
40
41      string first = "Sydney";
42      string last = "Holmes";
43      string full = " ";
44      full = first + " " + last;
45
46  /*--------------------------------------*/
47
48      string sentence = "How are you"
49      sentence = sentence + "?";
50
```

- Concatenates the contents of the `first` variable, a space, and the contents of the `last` variable; results are assigns to the `full` variable as “Sydney Holmes”
- Concatenates the contents of the sentence variable and a question mark and the assigns the result “ How are you?”

Work Cited

- Diane Zax, "An Introduction to Programming with C++, Sixth Edition",
  - Chapter 13 – String.
- Towson University, Professor Robert Eyer, COSC 175,
  - Chapter 13 Lecture Slides..