**10.1 Character Testing**

- There are several functions that answer questions about characters.
- Each has one char parameter.
- Each returns true or false.

#### Functions:

- `isalpha` true if argument is a letter, false otherwise.
- `isalnum` true if argument is a letter or digit, false otherwise.
- `isdigit` true if argument is a digit 0-9, false otherwise.
- `islower` true if argument is lowercase, false otherwise.
- `isprint` true if argument is printable, false otherwise.
- `ispunct` true if argument is a punctuation character, false otherwise.
- `isupper` true if argument is an uppercase letter, false otherwise.
- `isspace` true if argument is a whitespace character, false otherwise.

**Example**

```c
char ch1 = 'H';
char ch2 = 'e';
char ch3 = '!
```

```
char c1 = toupper(ch1);  // assigns 'H'
char c2 = toupper(ch2); // assigns 'E'
char c3 = toupper(ch3);  // assigns '!
```

**10.2 Character Case Conversion**

- Functions:
  - `toupper`: if char argument is lowercase letter, return uppercase equivalent; otherwise, return input unchanged.
  - `tolower`: if char argument is uppercase letter, return lowercase equivalent; otherwise, return input unchanged.

```c
char ch1 = 'H';
char ch2 = 'e';
char ch3 = '!
```

```
char c1 = toupper(ch1);  // assigns 'H'
char c2 = toupper(ch2); // assigns 'E'
char c3 = toupper(ch3);  // assigns '!
```
Review of the Internal Storage of C-Strings

- **C-string**: sequence of characters stored in adjacent memory locations and terminated by NULL character
- **String literal (string constant)**: sequence of characters enclosed in double quotes " ":
  
  ```
  "Hi there!"
  
  Hi there!\0
  ```

10.3 Review of the Internal Storage of C-Strings

- Array of `char` can be used to define storage for string:
  
  ```
  const int SIZE = 20;
  char city[SIZE];
  ```

- Leave room for NULL at end

- Can enter a value using `cin` or `>>`
  
  – Input is whitespace-terminated
  
  – No check to see if enough space

- For input containing whitespace, and to control amount of input, use `cin.getline()`

- Can write entire array using `cout`
  
  ```
  cout<<city;
  ```

10.4 Library Functions for Working with C-Strings

- These functions require c-strings as parameters (actually the address of a c-string.) This can be accomplished by:
  
  – sending the name of an array holding a c-string
  
  – a char pointer (we'll learn about these later)

- Some of these c-string functions lead to potential security hazards and have been deprecated.

- These will produce a warning in the Visual Studio environment

- Although the textbook uses the deprecated versions of these functions, we will use the newer, more secure versions

Library Functions for Working with C-Strings

**Functions:**

- `strlen(str)`: returns length of C-string
  
  ```
  char city[SIZE] = "Missoula";
  cout << strlen(city); // prints 8
  ```

- `strcat_s(str1, str2)`: appends `str2` to the end of `str1`
  
  ```
  const int SIZE = 100;
  char location[SIZE] = "Missoula, ";
  char state[3] = "MT";
  strcat_s(location, state);
  // location now has "Missoula, MT"
  ```

- `strcpy_s(str1, str2)`: copies `str2` to `str1`
  
  ```
  const int SIZE = 100;
  char fname[SIZE] = "Maureen", name[SIZE];
  strcpy_s(name,fname);
  ```

C-Strings

Ht there!
C-string Inside a C-string

Function:

– `strcmp(str1, str2)`: compares `str1` with `str2`.
  - Returns 0 if they are the same.
  - Returns a positive number if `str1` > `str2`.
  - Returns a negative number if `str1` < `str2`.

```c
char str1[] = "Hello";
char str2[] = "Hella";
if (strcmp(str1, str2) > 0)
    cout << str1 << " is greater than " << str2 << endl;
else if (strcmp(str1, str2) < 0)
    cout << str1 << " is shorter than " << str2 << endl;
else
    cout << str1 << " is the same as " << str2 << endl;
```

The C++ String Class

• C-string has some limitations:
  - You have to know how big to make it when you declare it.
  - You have to manage bounds checking.
  - You have to use the available functions to perform common tasks.

  – `strcpy_s`
  – `strcmp`
  – `strlen`
  – `strcat_s`

• **The C++ String Class**

  • Special data type supports working with strings

  ```
  #include <string>
  ```

  • Can define string variables in programs:
    ```
    string firstName, lastName;
    ```

  • Can receive values with assignment operator:
    ```
    firstName = "George";
    lastName = "Washington";
    ```

  • Can be displayed via `cout` operators:
    ```
    cout << firstName << " " << lastName;
    ```

  Example:

  ```
  cin >> firstName >> lastName;
  cout << "Enter your first name: ";
  cin >> firstName;
  ```

  The C++ String Class

  • Use `cin >>` to read an item into a string:
  
  • Can be displayed via `cout`:
    ```
    firstName = "George";
    ```

  • Setting `firstName`, `lastName`:
    ```
    firstName = "George";
    ```

  • Can define string variables in programs:
    ```
    string firstName, lastName;
    ```

  • Special data type supports working with strings

  ```
  #include <string>
  ```

  • Can receive values with assignment operator:
    ```
    firstName = "George";
    ```

  • Can be displayed via `cout` operators:
    ```
    cout << firstName << " " << lastName;
    ```

  Example:

  ```
  cin >> firstName >> lastName;
  cout << "Enter your first name: ";
  cin >> firstName;
  ```

  The C++ String Class

  • Use `cin >>` to read an item into a string:
  
  • Can be displayed via `cout`:
    ```
    firstName = "George";
    ```

  • Setting `firstName`, `lastName`:
    ```
    firstName = "George";
    ```

  • Can define string variables in programs:
    ```
    string firstName, lastName;
    ```

  • Special data type supports working with strings

  ```
  #include <string>
  ```

  • Can receive values with assignment operator:
    ```
    firstName = "George";
    ```

  • Can be displayed via `cout` operators:
    ```
    cout << firstName << " " << lastName;
    ```

  Example:

  ```
  cin >> firstName >> lastName;
  cout << "Enter your first name: ";
  cin >> firstName;
  ```

  The C++ String Class

  • Use `cin >>` to read an item into a string:
  
  • Can be displayed via `cout`:
    ```
    firstName = "George";
    ```

  • Setting `firstName`, `lastName`:
    ```
    firstName = "George";
    ```

  • Can define string variables in programs:
    ```
    string firstName, lastName;
    ```

  • Special data type supports working with strings

  ```
  #include <string>
  ```

  • Can receive values with assignment operator:
    ```
    firstName = "George";
    ```

  • Can be displayed via `cout` operators:
    ```
    cout << firstName << " " << lastName;
    ```

  Example:

  ```
  cin >> firstName >> lastName;
  cout << "Enter your first name: ";
  cin >> firstName;
  ```
There are multiple ways to define a string:

- ```string name;``` defines an empty string object
- ```string myname("Chris");``` defines a string and initializes it
- ```string yourname(myname);``` defines a string and initializes it
- ```string aname(myname, 3);``` defines a string and initializes it with first 3 characters of myname
- ```string verb(myname, 3, 2);``` defines a string and initializes it with 2 characters from myname starting at position 3
- ```string noname('A', 5);``` defines string and initializes it to 5 'A's

Other Definitions of C++

**String Operators**

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;</td>
<td>extracts characters from stream up to whitespace, insert into string</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>inserts string into stream</td>
</tr>
<tr>
<td>=</td>
<td>assigns string on right to string object on left</td>
</tr>
<tr>
<td>+=</td>
<td>appends string on right to end of contents on left</td>
</tr>
<tr>
<td>+</td>
<td>concatenates two strings</td>
</tr>
<tr>
<td>[ ]</td>
<td>references character in string using array notation</td>
</tr>
</tbody>
</table>

- > | >= | < | <= | == | !=

```
class string {  
    friend istream & operator>>(istream &istr, string &s);  
    friend ostream & operator<<(ostream &ostream, const string &s);  
    string(string &s) {  
        this->s = s.s;  
        this->length = s.length;  
    }  
};  
```
Strings:

- Arrays of strings:

```
string word1, word2, phrase;
```

```
cin >> word1; // user enters "Hot Tamale"
// word1 has "Hot"
phrase = word1 + word2; // phrase has "Hot Dog"
phrase += " on a bun";
```

```
for (int i = 0; i < 16; i++)
    cout << phrase[i]; // displays "Hot Dog on a bun"
```

**Member Functions**

- **Assignment:** assign, copy, data
- **Modification:** append, clear, erase, insert, replace, swap
- **Space Management:** capacity, empty, length, resize, size
- **Substrings:** find, substr
- **Comparison:** compare

```
string word1, word2, phrase;
word2.assign(" Dog");
phrase.append(word1);
phrase.append(word2); // phrase has "Hot Dog"
phrase.append(" with mustard relish", 13); // phrase has "Hot Dog with mustard"
phrase.insert(8, "on a bun ");
```

```
cout << phrase << endl; // displays "Hot Dog on a bun with mustard"
```

Arrays of strings

```
string titles[size];
```

**Operators**

```
foreach (const char *word) in string...
```

**See Table 10-7 for a list of functions**

- **Comparison:** compare
- **Substrings:** find, substr
- **Length:** length, size
- **Space Management:** capacity, empty, length, resize, size
- **Substrings:** find, substr
- **Comparison:** compare

**Member Functions**

- **Assignment:** assign, copy, data
- **Modification:** append, clear, erase, insert, replace, swap
- **Space Management:** capacity, empty, length, resize, size
- **Substrings:** find, substr
- **Comparison:** compare

```
Program 10-13
```
Passing strings to Functions

• In the prototype:
  
  ```cpp
  void function(string);
  ```

• In the call:
  
  ```cpp
  function(myString);
  ```

• In the definition
  
  ```cpp
  void function(string title)
  {
      ...
  }
  ```

Example

```cpp
string Name;

Example
```