CS 31: Introduction To Computer Science I
Howard A. Stahl

Agenda

• CStrings
• cstring Library Functions
• cstdlib Library Functions
• cctype Library Functions
• CStrings As Function Arguments
• string Class

Text Data

• C++ Provides Two Core String Datatypes
• C++ Provides “C-Strings”
  – String Datatype From The C Language Point-Of-View
  – A Null-Terminated Array of char
• C++ Provides The string Class To Manipulate Text Data
  – You Must #include <string>
CStrings

• CStrings Are Arrays Of Characters
• CStrings Are How C Programs Manipulate Text
• We’ve Already Been Using CStrings
  – The Literal “Hello World” Is Stored As A CString

• CStrings Are, By Convention, Always Null Terminated
  – NULL is a special symbol: ‘\0’
• You Must Have Enough Space In Your Array For This Extra Character!
• CString Arrays Are Partially Filled Arrays
• CStrings Are Arrays With An Extra Letter

• Example:
  char s[ 10 ];
CStrings

• Example:
  ```c
  char s[10];
  ```

CStrings

• Example:
  ```c
  char t[5] = "Hi";
  ```
CStrings

- Example:
  ```c
  char t[5] = "Hi";
  ```

CStrings

- Example:
  ```c
  char u[] = "Howie";
  ```
CStrings

- Example:
  ```
  char u[] = "Howie!";
  ```

CStrings Observations

- CStrings Are Implemented As Arrays Of Char
- Unfortunately, CStrings Are Used In A Different Way And Do Not Support Your Intuition, As You Will See
  - the sentinel \0 is vital to cstrings working right!
- CStrings Are Tricky!

CStrings CounterIntuition #1

- Example:
  ```
  char v[] = "abc";
  char w[] = {'a', 'b', 'c'};
  ```
CStrings CounterIntuition #1

• Example:
  ```
  char v[] = "abc";
  char w[] = {'a', 'b', 'c'};
  ```

These Are Not Equivalent Arrays!

CStrings CounterIntuition #2

• Example:
  ```
  char x[5];
  x = "Foo";
  ```

CStrings Cannot Be Assigned A Value
CString CounterIntuition #2

- Example:
  ```cpp
cchar x[5];
x = "Foo";
```

CString Is An Array

- You Can Use An Index To Walk The Array
- Example:
  ```cpp
cchar lots_of_x[] = "Hello World";
int index = 0;
while (lots_of_x[ index ] != '\0') {
  lots_of_x[ index ] = 'x';
  index++;
}
```

CString Is An Array

- Be Very Careful Never To Write Over The Ending NULL Character
  - If You Do, You MUST Put A NULL Character Onto The End Of Your Array
  - Otherwise, You Will Get Unpredictable Results
Time For Our First Demo!

- CString.cpp

(See Handout For Example 1)

Summarizing Our First Demo!

- Always Remember About \0 When Working With CStrings
- You Cannot Use Assignment With CStrings
- CStrings Can Be Declared `const` If Appropriate

<string> Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcpy(target_str, src_str)</td>
<td>Copies the string in <code>src_str</code> to <code>target_str</code> to the end.</td>
<td>Does not check to make sure <code>target_str</code> is large enough to hold the value of <code>src_str</code>.</td>
</tr>
<tr>
<td>strncpy(target_str, src_str, max_len)</td>
<td>The same as the two argument <code>strcpy</code> except that no more than <code>max_len</code> characters are copied.</td>
<td>Doesn't check to make sure that <code>target_str</code> is large enough to hold the result of the <code>strncpy</code> operation.</td>
</tr>
<tr>
<td>wcscpy(target_str, wcsstr, wcslen)</td>
<td>Copies the string in <code>wcsstr</code> to <code>target_str</code> to the end of.</td>
<td>Doesn't check to make sure that <code>target_str</code> is large enough to hold the result of the comparison.</td>
</tr>
</tbody>
</table>
**cstring Library**

- The *cstring* System Library Provides Access To Various C Library String Manipulation Routines

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>MEANING</th>
<th>ARGUMENTS</th>
<th>RETURNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcpy(dest, src)</td>
<td>dest = src</td>
<td>cstring, cstring</td>
<td>void</td>
</tr>
<tr>
<td>strcat(dest, src)</td>
<td>dest = dest + src</td>
<td>cstring, cstring</td>
<td>void</td>
</tr>
<tr>
<td>strlen(src)</td>
<td>length of cstring src</td>
<td>cstring</td>
<td>int</td>
</tr>
<tr>
<td>strcmp(s1, s2)</td>
<td>compares s1 and s2</td>
<td>cstring, cstring</td>
<td>int</td>
</tr>
</tbody>
</table>

See Appendix 4 For A Fuller List

**cstring System Library**

- `strlen(char s[])` :: int
  - returns length of `s` NOT including `\0`
- `strcmp(char s[], char t[])` :: int
  - returns 0 if `s` equals `t` lexicographically
  - returns <0 if `s` is lexicographically less than `t`
  - returns >0 if `s` is lexicographically greater than `t`
  - NOTE: 0 equates to `false` in expressions
cstdlib System Library

• The cstdlib System Library Provides Access To Various Conversion Routines

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>MEANING</th>
<th>ARGUMENTS</th>
<th>RETURNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>atoi( src )</td>
<td>ascii to integer</td>
<td>cstring</td>
<td>int</td>
</tr>
<tr>
<td>atof( src )</td>
<td>ascii to floating-point</td>
<td>cstring</td>
<td>double</td>
</tr>
<tr>
<td>atol( src )</td>
<td>ascii to long</td>
<td>cstring</td>
<td>long</td>
</tr>
</tbody>
</table>

See Appendix 4 For A Fuller List

CString Functions: strlen()

• "String Length"
• Often Useful To Know A String’s Length

```c
char myString[10] = "dobedo";
cout << strlen(myString);
```

• Returns Number Of Characters Not Including NULL
• Result here: 6

CString Functions: strcat()

• "String concatenate"
• Essentially += On Two Pieces Of Text

```c
char stringVar[20] = "The rain";
strcat(stringVar, "in Spain");
```

• Result: stringVar = "The rain in Spain"
• Be Careful! Have Enough Space For The Result And An Ending NULL!
• Add spaces as needed!
### <cctype> System Library

- Another Useful Library Of Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Argument</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>toupper(c)</td>
<td>char</td>
<td>int</td>
</tr>
<tr>
<td>tolower(c)</td>
<td>char</td>
<td>int</td>
</tr>
<tr>
<td>isupper(c)</td>
<td>char</td>
<td>bool</td>
</tr>
<tr>
<td>islower(c)</td>
<td>char</td>
<td>bool</td>
</tr>
<tr>
<td>isalpha(c)</td>
<td>char</td>
<td>bool</td>
</tr>
<tr>
<td>isdigit(c)</td>
<td>char</td>
<td>bool</td>
</tr>
<tr>
<td>isspace(c)</td>
<td>char</td>
<td>bool</td>
</tr>
</tbody>
</table>

---

Example:

```cpp
char c = toupper('a');
cout << c;
```

Although function returns `int`, it can be cast back into a `char`, as we expect

- Alternatively,

```cpp
cout << char( toupper('a') );
```
CStrings As Function Parameters

- Under The Covers, A CString Is An Array
- CStrings Passed To A Function Can Be Changed By The Function And The Caller Will See The Result
- If It Is A Read-Only Argument, Mark The CString With const
- Size Companion Parameter Only Needed If The String Will Get Changed

CString Input

- We Have Two Choices:
  - Use >> With cin Which Eats Leading Whitespace And Reads Just A Single Word
  - Use cin.getline Which Is The Version That Supports CString Data

CString Input

- cin.getline( char *, int )
char a[80];
cout << "Enter input: ";
cin.getline(a, 80);
cout << a << endl;
- Result: Enter input: Do be do to you!
           Do be do to you!
CStrings Are Nasty!

- Example:
  ```c
  char var[3];
  strcpy( var, "A Nice Long String" );
  ```

You Must Use `strcpy` To Do Assignments

Completely Legal And Yet Disasterous!
CStrings Are Nasty!

• Example:
  char var[3];
  strcpy( var, "A Nice Long String" );

It Is Always The Programmer’s Responsibility To Ensure That The Destination Is Large Enough For The String Being Placed There

CStrings As Function Arguments

• Like Other Array Variables, CStrings Can Be Passed To Functions
• Generally, It’s A Good Idea To Provide An Argument That Specifies The Maximum String Size Allowed
• Read-Only Functions Can Use The ‘0’ Sentinel To Determine The End Of The String

Understanding CString Parameters

• Read-Only Functions Can Work Off The ‘0’ Sentinel Embedded In A CString
• Functions Updating A CString Variable Should Be Provided A Maximum Size Allowed Value As A Parameter
• Use strcmp Carefully!
  – return 0 when the two cstrings are equal
  – ~0 equates to false
Stream Input

• By Default, Stream Insertion Operator Eats Whitespace
• But Whitespace Is Meaningful To Strings
• To Read Character Data, Use getline Function

• Example:
  ```
  istream::getline(char s[],int i)
  reads up to i-1 chars into s, stops at new-line
  ```
  ```
  const int LINESIZE=80;
  char line1[LINESIZE];
  char line2[LINESIZE];
  cin.getline( line1, LINESIZE );
  cin.getline( line2, LINESIZE );
  ```

Stream Input

• Unfortunately, Things Are More Involved...
• When Intermixing >> With getline, Note That cin Eats Only Opening Whitespace
Stream Input

- Unfortunately, Things Are More Involved...
- When Intermixing >> With getline, Note That cin Eats Only Opening Whitespace

```cpp
int number;
char str[80];

cin >> number;
cin.getline( str, 80 );
```

Stream Input

- Unfortunately, Things Are More Involved...
- When Intermixing >> With getline, Note That cin Eats Only Opening Whitespace

```cpp
int number;
char str[80];

cin >> number;
cin.getline( str, 80 );
```

Stream Input

- Unfortunately, Things Are More Involved...
- When Intermixing >> With getline, Note That cin Eats Only Opening Whitespace

```cpp
int number;
char str[80];

cin >> number;
cin.getline( str, 80 );
```
• Unfortunately, Things Are More Involved...
• When Intermixing `>>` With `getline`, Note That `cin` Eats Only Opening Whitespace

```cpp
int number;
char str[80];

// Stream Input

```
Stream Input

- Unfortunately, Things Are More Involved...
- When Intermixing >> With getline, Note That cin Eats Only Opening Whitespace
- Use istream::ignore( int, char )

```cpp
int number, 30
char str[80];

cin >> number;
cin.ignore( 10000, '\n' );
cin.getline( str, 80 );
```

string Class

- A Standard Class From The string Library
- + Operator Performs Concatenation
- Default And String Argument Constructor
- Allows For Character Access Using [] Indexing
- << And >> Are Overloaded As You Expect
- All Boolean Operators Work As You Expect
String Class

- Example:

```cpp
#include <string>

string name, dog("dog"), hotdog;
cin >> name;
hotdog = "hot " + dog;
for (int i=0; i < name.length(); ++i)
    cout << name[i] << ";
```

[ ] operator does not perform bounds checking

Consider using `at()` member.
**getline Function For string Objects**

- **getline() Function For string Objects Is A Normal Function**
  - not a member of istream

```cpp
string& getline( istream& input,
    string& str,
    char delimiter = '\n'
);
```

**string Member Functions**

- **string Class Is Pretty Functional!**

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>MEANING</th>
<th>ARGUMENTS</th>
<th>RETURNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring</td>
<td>substring starting at pos for length len</td>
<td>int, int</td>
<td>string</td>
</tr>
<tr>
<td>empty()</td>
<td>tests for empty</td>
<td></td>
<td>boolean</td>
</tr>
<tr>
<td>insert(pos, str)</td>
<td>insert string at pos</td>
<td>int, string</td>
<td>void</td>
</tr>
<tr>
<td>remove(pos, len)</td>
<td>remove starting at pos for length len</td>
<td>int, int</td>
<td>void</td>
</tr>
<tr>
<td>find(str)</td>
<td>find first occurrence of str in instance</td>
<td>string</td>
<td>int</td>
</tr>
</tbody>
</table>

See Page 650 For A Fuller List
Time For Our Next Demo!

- String.cpp

(See Handout For Example 2)

Summarizing Our Second Demo!

- `string` Class Is Much Nicer To Deal With!
- `string` Class Follows Our Intuition
- Remember That Boolean Operators Work Different With CString And `string` Objects!

Summary

- CStrings
- `cstring` Library Functions
- `cstdlib` Library Functions
- `cctype` Library Functions
- CStrings As Function Arguments
- `string` Class