C101 Topics

Adapted from Dr. Raman Adaikkalavan, Lecture CS, IUSB
Based on the first class discussion, the following topics will be reviewed:

- enum
- continue
- static
- namespace
- #define, #ifdef, #ifndef, #endif
- strings
- external, function overloading - will be covered later
Enumeration Data Type

- List of constants of type **int**
  - Typical use: case labels in switch, ...
  - Unlimited number of constants
  - Use as labels and avoid using in arithmetic operations
  - Always increases by one from left to right

```c
enum Direction {NORTH = 0, SOUTH = 1, EAST = 2, WEST = 3};
enum Direction {NORTH, SOUTH, EAST, WEST};
enum MyVal {ONE = 12, TWO, THREE = -2, FOUR};

switch (value) {
    case NORTH:
    case SOUTH:
        ...
}
Continue

• Used to skip statements without executing

```cpp
int count (0);
for (count = 0; count < 5; count++) {
    if (3 == count) {
        cout << count << endl;
        break; //break and get out of the loop
    }
    else {
        continue; //ignore all the below statements
        //and go to the loop statement again
        cout << count << endl;
    }
}
```
//create a miniworld with its own functions and variables
namespace raman {
    double userInput = 0;
    int count();
}
namespace jim {
    int count();
}
using namespace raman;
using namespace jim;

//cout << count() << endl; // ambiguous
cout << raman::count() << endl;
cout << jim::count() << endl;
#define, #ifdef, #ifndef, #endif

- define_0.cpp
C Style Strings
C-String Storage

• `#include <cstring>

• A standard array:
  `char s[10];`

  – If `s` contains string "Hi Mom!", it is stored as

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>i</td>
<td>M</td>
<td>o</td>
<td>m</td>
<td>!</td>
<td>\0</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
C-String Input Example

```cpp
char a[80], b[80];
cout << "Enter input: ";
cin >> a >> b;
cout << a << b << "END OF OUTPUT\n";
```

- Dialogue offered:
  - Enter input: **One two three four!**
    - **OnetwoEND OF OUTPUT**
  - Note: Underlined portion typed at keyboard

- C-string *a* receives: “one"
- C-string *b* receives: “two"
C-String Line Input

• Can receive entire line into c-string

• Use `getline()`, a predefined member function:
  ```
  char a[80];
cout << "Enter input: ";
cin.getline(a, 80);
cout << a << "END OF OUTPUT\n";
  ```

  Dialogue:
  Enter input: Do be do to you!
  One two three four!END OF OUTPUT
More getline()

• Can specify the length to receive:

```cpp
char shortString[5];
cout << "Enter input: ";
cin.getline(shortString, 5);
cout << shortString << "END OF OUTPUT\n";
```

– Results:

Enter input: dobedowap
dobeEND OF OUTPUT

– Forces **FOUR** characters only be read
  • Recall the need for **null** (‘\0’) character!
### Display 9.1  Some Predefined C-String Functions in `<cstring>`

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>CAUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(Target_String_Var, Src_String)</code></td>
<td>Copies the C-string value <code>Src_String</code> into the C-string variable <code>Target_String_Var</code>.</td>
<td>Does not check to make sure <code>Target_String_Var</code> is large enough to hold the value <code>Src_String</code>.</td>
</tr>
<tr>
<td><code>strcpy(Target_String_Var, Src_String, Limit)</code></td>
<td>The same as the two-argument <code>strcpy</code> except that at most <code>Limit</code> characters are copied.</td>
<td>If <code>Limit</code> is chosen carefully, this is safer than the two-argument version of <code>strcpy</code>. Not implemented in all versions of C++.</td>
</tr>
<tr>
<td><code>strcat(Target_String_Var, Src_String)</code></td>
<td>Concatenates the C-string value <code>Src_String</code> onto the end of the C-string in the C-string variable <code>Target_String_Var</code>.</td>
<td>Does not check to see that <code>Target_String_Var</code> is large enough to hold the result of the concatenation.</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>CAUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcat(Target_String_Var, Src_String, Limit)</td>
<td>The same as the two argument strcat except that at most Limit characters are appended.</td>
<td>If Limit is chosen carefully, this is safer than the two-argument version of strcat. Not implemented in all versions of C++.</td>
</tr>
<tr>
<td>strlen(Src_String)</td>
<td>Returns an integer equal to the length of Src_String. (The null character, \0, is not counted in the length.)</td>
<td></td>
</tr>
<tr>
<td>strcmp(String_1, String_2)</td>
<td>Returns 0 if String_1 and String_2 are the same. Returns a value &lt; 0 if String_1 is less than String_2. Returns a value &gt; 0 if String_1 is greater than String_2 (that is, returns a nonzero value if String_1 and String_2 are different). The order is lexicographic.</td>
<td>If String_1 equals String_2, this function returns 0, which converts to false. Note that this is the reverse of what you might expect it to return when the strings are equal.</td>
</tr>
<tr>
<td>strcmp(String_1, String_2, Limit)</td>
<td>The same as the two-argument strcat except that at most Limit characters are compared.</td>
<td>If Limit is chosen carefully, this is safer than the two-argument version of strcmp. Not implemented in all versions of C++.</td>
</tr>
</tbody>
</table>
C-string Functions: strlen()

• "String length"

• Often useful to know string length:
  char myString[10] = "dobedo";
  cout << strlen(myString);  
  – Returns number of characters
    • Not including null
  – Result here:
    6
C-string Functions: strcat()

• strcat()

• "String concatenate":
  char stringVar[20] = "The rain";
  strcat(stringVar, "in Spain");
  – Note result:
    stringVar now contains "The rainin Spain"
  – Be careful!
  – Incorporate spaces as needed!
Character I/O

• Input and output data
  – ALL treated as character data
  – e.g., number 10 outputted as "1" and "0"
  – Conversion done automatically
    • Uses low-level utilities

• Can use same low-level utilities ourselves as well
Member Function get()

• Reads one char at a time

• Member function of cin object:
  char nextSymbol;
  cin.get(nextSymbol);
  – Reads next char & puts in variable nextSymbol
  – Argument must be char type
    • Not "string"! 
Member Function put()

- Outputs one character at a time
- Member function of cout object:

Examples:
```cpp
cout.put('a');  // Outputs letter 'a' to screen
char myString[10] = "Hello";
cout.put(myString[1]);  // Outputs letter 'e' to screen
```
More Member Functions

• putback()
  – Once read, might need to "put back"
  – `cin.putback(lastChar);`

• peek()
  – Returns next char, but leaves it there
  – `peekChar = cin.peek();`

• ignore()
  – Skip input, up to designated character
  – `cin.ignore(1000, '\n');`
    • Skips at most 1000 characters until '\n'
### Display 9.3 Some Functions in `<cctype>` (1 of 3)

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DESCRIPTION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>toupper(CharExp)</td>
<td>Returns the uppercase version of CharExp (as a value of type int).</td>
<td>char c = toupper('a');</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cout &lt;&lt; c;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outputs: A</td>
</tr>
<tr>
<td>tolower(CharExp)</td>
<td>Returns the lowercase version of CharExp (as a value of type int).</td>
<td>char c = tolower('A');</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cout &lt;&lt; c;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outputs: a</td>
</tr>
<tr>
<td>isupper(CharExp)</td>
<td>Returns true provided CharExp is an uppercase letter; otherwise, returns false.</td>
<td>if (isupper(c))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cout &lt;&lt; &quot;Is uppercase.&quot;;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cout &lt;&lt; &quot;Is not uppercase.&quot;;</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>DESCRIPTION</td>
<td>EXAMPLE</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| islower(Char_Exp) | Returns true provided Char_Exp is a lowercase letter; otherwise, returns false. | `char c = 'a';
if (islower(c))
    cout << c << " is lowercase."
Outputs: a is lowercase. |
| isalpha(Char_Exp) | Returns true provided Char_Exp is a letter of the alphabet; otherwise, returns false. | `char c = '$';
if (isalpha(c))
    cout << "Is a letter."
else
    cout << "Is not a letter."
Outputs: Is not a letter. |
| isdigit(Char_Exp) | Returns true provided Char_Exp is one of the digits '0' through '9'; otherwise, returns false. | `if (isdigit('3'))
    cout << "It's a digit."
else
    cout << "It's not a digit."
Outputs: It's a digit. |
| isalnum(Char_Exp) | Returns true provided Char_Exp is either a letter or a digit; otherwise, returns false. | `if (isalnum('3') && isalnum('a'))
    cout << "Both alphanumeric."
else
    cout << "One or more are not."
Outputs: Both alphanumeric. |
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Code snippet</th>
</tr>
</thead>
</table>
| `isspace(Char_Exp)` | Returns true provided `Char_Exp` is a whitespace character, such as the blank or newline character; otherwise, returns false. | //Skips over one "word" and sets c //equal to the first whitespace //character after the "word":
```cpp
do {
    cin.get(c);
} while (! isspace(c));
``` |
| `ispunct(Char_Exp)` | Returns true provided `Char_Exp` is a printing character other than whitespace, a digit, or a letter; otherwise, returns false. | if (ispunct('?'))
    cout << "Is punctuation.";
else
    cout << "Not punctuation."; |
| `isprint(Char_Exp)` | Returns true provided `Char_Exp` is a printing character; otherwise, returns false. | |
| `isgraph(Char_Exp)` | Returns true provided `Char_Exp` is a printing character other than whitespace; otherwise, returns false. | |
| `isctrl(Char_Exp)` | Returns true provided `Char_Exp` is a control character; otherwise, returns false. | |
C++ Strings
Standard Class \texttt{string}

\begin{verbatim}
#include <string>
using namespace std;

• String variables and expressions
• Can assign, compare, add:

\texttt{string s1, s2, s3;}
\texttt{s3 = s1 + s2; // Concatenation}
\texttt{s3 = "Hello Mom!" // Assignment}
\end{verbatim}

– Note c-string "Hello Mom!" automatically converted to string type!
Display 9.4
Program Using the Class string

```cpp
// Demonstrates the standard class string.
#include <iostream>
#include <string>
using namespace std;

int main() {
    string phrase;
    string adjective("fried"), noun("ants");
    string wish = "Bon appetite!";

    phrase = "I love " + adjective + " " + noun + "!";
    cout << phrase << endl
        << wish << endl;

    return 0;
}
```

**SAMPLE DIALOGUE**
I love fried ants!
Bon appetit!
I/O with Class string

• Just like other types!

• `string s1, s2; cin >> s1; cin >> s2;`

• Results:
  User types in:
  May the hair on your toes grow long and curly!

• Extraction still ignores whitespace:
  s1 receives value "May"
  s2 receives value "the"
getline() with Class string

• For complete lines:
  string line;
  cout << "Enter a line of input: ";
  getline(cin, line);
  cout << line << "END OF INPUT";

• Dialogue produced:
  Enter a line of input: Do be do to you!
  Do be do to you!END OF INPUT

  – Similar to c-string’s usage of getline()
Other getline() Versions

• Can specify "delimiter" character:
  ```cpp
  string line;
  cout << "Enter input: ";
  getline(cin, line, '?');
  ```
  Receiving input until "?" encountered

• getline() actually returns reference
  ```cpp
  string s1, s2;
  getline(cin, s1) >> s2;
  ```
  Results in: (cin) >> s2;
Pitfall: Mixing Input Methods

• Be careful mixing cin >> var and getline

  - int n;
    string line;
    cin >> n;
    getline(cin, line);

  - If input is: 42
    Hello hitchhiker.

    • Variable n set to 42
    • line set to empty string!

  - cin >> n skipped leading whitespace, leaving ‘\n’ on stream for getline()!
Class string Processing

• The string class has the same operations as c-strings

• And more!
  – Over 100 members of standard string class

• Some member functions:
  – .length()
    • Returns length of string variable
  – .at(i)
    • Returns reference to char at position i
    • \texttt{str[2]} is equivalent to \texttt{str.at(2)}
### Member Functions of the Standard Class `string`

<table>
<thead>
<tr>
<th>Example</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructors</strong></td>
<td></td>
</tr>
<tr>
<td><code>string str;</code></td>
<td>Default constructor; creates empty string object <code>str</code></td>
</tr>
<tr>
<td><code>string str(&quot;string&quot;);</code></td>
<td>Creates a string object with data &quot;string&quot;.</td>
</tr>
<tr>
<td><code>string str(aString);</code></td>
<td>Creates a string object <code>str</code> that is a copy of <code>aString</code>. <code>aString</code> is an object of the class <code>string</code>.</td>
</tr>
<tr>
<td><strong>Element access</strong></td>
<td></td>
</tr>
<tr>
<td><code>str[i]</code></td>
<td>Returns read/write reference to character in <code>str</code> at index <code>i</code>.</td>
</tr>
<tr>
<td><code>str.at(i)</code></td>
<td>Returns read/write reference to character in <code>str</code> at index <code>i</code>.</td>
</tr>
<tr>
<td><code>str.substr(position, length)</code></td>
<td>Returns the substring of the calling object starting at <code>position</code> and having <code>length</code> characters.</td>
</tr>
<tr>
<td><strong>Assignment/Modifiers</strong></td>
<td></td>
</tr>
<tr>
<td><code>str1 = str2;</code></td>
<td>Allocates space and initializes it to <code>str2</code>'s data, releases memory allocated for <code>str1</code>, and sets <code>str1</code>'s size to that of <code>str2</code>.</td>
</tr>
<tr>
<td><code>str1 += str2;</code></td>
<td>Character data of <code>str2</code> is concatenated to the end of <code>str1</code>; the size is set appropriately.</td>
</tr>
<tr>
<td><code>str.empty()</code></td>
<td>Returns true if <code>str</code> is an empty string; returns false otherwise.</td>
</tr>
</tbody>
</table>
Display 9.7  Member Functions of the Standard Class string

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str1 + str2</code></td>
<td>Returns a string that has <code>str2</code>’s data concatenated to the end of <code>str1</code>’s data. The size is set appropriately.</td>
</tr>
<tr>
<td><code>str.insert(pos, str2)</code></td>
<td>Inserts <code>str2</code> into <code>str</code> beginning at position <code>pos</code>.</td>
</tr>
<tr>
<td><code>str.remove(pos, length)</code></td>
<td>Removes substring of size <code>length</code>, starting at position <code>pos</code>.</td>
</tr>
</tbody>
</table>

**Comparisons**

<table>
<thead>
<tr>
<th><code>str1 == str2</code></th>
<th><code>str1 != str2</code></th>
<th>Compare for equality or inequality; returns a Boolean value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str1 &lt; str2</code></td>
<td><code>str1 &gt; str2</code></td>
<td>Four comparisons. All are lexicographical comparisons.</td>
</tr>
<tr>
<td><code>str1 &lt;= str2</code></td>
<td><code>str1 &gt;= str2</code></td>
<td>Returns index of the first occurrence of <code>str1</code> in <code>str</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>str.find(str1)</code></th>
<th>Returns index of the first occurrence of string <code>str1</code> in <code>str</code>; the search starts at position <code>pos</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str.find(str1, pos)</code></td>
<td></td>
</tr>
<tr>
<td><code>str.find_first_of(str1, pos)</code></td>
<td>Returns the index of the first instance in <code>str</code> of any character in <code>str1</code>, starting the search at position <code>pos</code>.</td>
</tr>
<tr>
<td><code>str.find_first_not_of</code></td>
<td>Returns the index of the first instance in <code>str</code> of any character not in <code>str1</code>, starting search at position <code>pos</code>.</td>
</tr>
</tbody>
</table>
C-string and C++ string Conversions

• Automatic type conversions
  
  – From c-string to string object:
    char aCString[] = "My C-string";
    string stringVar;
    stringVar = aCString;
    • Perfectly legal and appropriate!
  
  – aCString = stringVar;
    • ILLEGAL!
    • *Cannot auto-convert to c-string*

  – Must use explicit conversion:
    strcpy(aCString, stringVar.c_str());
String Summary

• C-string variable is "array of characters"
  – With addition of null character, ‘\0’

• C-strings act like arrays
  – Cannot assign, compare like simple variables

• Libraries <cctype> & <string> have useful manipulating functions

• cin.get() reads next single character

• getline() versions allow full line reading

• Class string objects are better-behaved than c-strings