Characters and Strings

James Daly
CSE 232
Michigan State University
Characters

- Example 6.1
Characters are complicated

- In the old days, there was a very simple character set, ASCII, which represented the basic English language characters
- Essentially what the standard `char` type represents
- Indicate with single quotes

```cpp
char my_char = 'a';
```
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx Oct</th>
<th>Ch</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Ch</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Ch</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Ch</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32 20 040</td>
<td>#32: Space</td>
<td>64 40 100</td>
<td>#64: 8</td>
<td>96 60 140</td>
<td>#96: `</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33 21 041</td>
<td>#33: !</td>
<td>65 41 101</td>
<td>#65: A</td>
<td>97 61 141</td>
<td>#97: a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34 22 042</td>
<td>#34: &quot;</td>
<td>66 42 102</td>
<td>#66: B</td>
<td>98 62 142</td>
<td>#98: b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35 23 043</td>
<td>#35: #</td>
<td>67 43 103</td>
<td>#67: C</td>
<td>99 63 143</td>
<td>#99: c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36 24 044</td>
<td>#36: $</td>
<td>68 44 104</td>
<td>#68: D</td>
<td>100 64 144</td>
<td>#100: d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37 25 045</td>
<td>#37: %</td>
<td>69 45 105</td>
<td>#69: E</td>
<td>101 65 145</td>
<td>#101: e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38 26 046</td>
<td>#38: &amp;</td>
<td>70 46 106</td>
<td>#70: F</td>
<td>102 66 146</td>
<td>#102: f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39 27 047</td>
<td>#39: '</td>
<td>71 47 107</td>
<td>#71: G</td>
<td>103 67 147</td>
<td>#103: g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40 28 050</td>
<td>#40: (</td>
<td>72 48 110</td>
<td>#72: H</td>
<td>104 68 150</td>
<td>#104: h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>41 29 051</td>
<td>#41: )</td>
<td>73 49 111</td>
<td>#73: I</td>
<td>105 69 151</td>
<td>#105: i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
<td>42 2A 052</td>
<td>#42: *</td>
<td>74 4A 112</td>
<td>#74: J</td>
<td>106 6A 152</td>
<td>#106: j</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43 2B 053</td>
<td>#43: +</td>
<td>75 4B 113</td>
<td>#75: K</td>
<td>107 6B 153</td>
<td>#107: k</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>014</td>
<td>FF (NP form feed, new page)</td>
<td>44 2C 054</td>
<td>#44: ,</td>
<td>76 4C 114</td>
<td>#76: L</td>
<td>108 6C 154</td>
<td>#108: l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45 2D 055</td>
<td>#45: -</td>
<td>77 4D 115</td>
<td>#77: M</td>
<td>109 6D 155</td>
<td>#109: m</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46 2E 056</td>
<td>#46: .</td>
<td>78 4E 116</td>
<td>#78: N</td>
<td>110 6E 156</td>
<td>#110: n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47 2F 057</td>
<td>#47: /</td>
<td>79 4F 117</td>
<td>#79: O</td>
<td>111 6F 157</td>
<td>#111: o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48 30 060</td>
<td>#48: 0</td>
<td>80 50 120</td>
<td>#80: P</td>
<td>112 70 160</td>
<td>#112: p</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49 31 061</td>
<td>#49: 1</td>
<td>81 51 121</td>
<td>#81: Q</td>
<td>113 71 161</td>
<td>#113: q</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50 32 062</td>
<td>#50: 2</td>
<td>82 52 122</td>
<td>#82: R</td>
<td>114 72 162</td>
<td>#114: r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51 33 063</td>
<td>#51: 3</td>
<td>83 53 123</td>
<td>#83: S</td>
<td>115 73 163</td>
<td>#115: s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52 34 064</td>
<td>#52: 4</td>
<td>84 54 124</td>
<td>#84: T</td>
<td>116 74 164</td>
<td>#116: t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53 35 065</td>
<td>#53: 5</td>
<td>85 55 125</td>
<td>#85: U</td>
<td>117 75 165</td>
<td>#117: u</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54 36 066</td>
<td>#54: 6</td>
<td>86 56 126</td>
<td>#86: V</td>
<td>118 76 166</td>
<td>#118: v</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55 37 067</td>
<td>#55: 7</td>
<td>87 57 127</td>
<td>#87: W</td>
<td>119 77 167</td>
<td>#119: w</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56 38 070</td>
<td>#56: 8</td>
<td>88 58 130</td>
<td>#88: X</td>
<td>120 78 170</td>
<td>#120: x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57 39 071</td>
<td>#57: 9</td>
<td>89 59 131</td>
<td>#89: Y</td>
<td>121 79 171</td>
<td>#121: y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58 3A 072</td>
<td>#58: :</td>
<td>90 5A 132</td>
<td>#90: Z</td>
<td>122 7A 172</td>
<td>#122: z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59 3B 073</td>
<td>#59: ;</td>
<td>91 5B 133</td>
<td>#91: [</td>
<td>123 7B 173</td>
<td>#123: {</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60 3C 074</td>
<td>#60: &lt;</td>
<td>92 5C 134</td>
<td>#92: \</td>
<td>124 7C 174</td>
<td>#124:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61 3D 075</td>
<td>#61: =</td>
<td>93 5D 135</td>
<td>#93: ]</td>
<td>125 7D 175</td>
<td>#125: }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62 3E 076</td>
<td>#62: &gt;</td>
<td>94 5E 136</td>
<td>#94: ^</td>
<td>126 7E 176</td>
<td>#126: ~</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63 3F 077</td>
<td>#63: ?</td>
<td>95 5F 137</td>
<td>#95: `</td>
<td>127 7F 177</td>
<td>#127: DEL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
The world is not just English

- A `char` is only 8 bits (1 byte) so it can only represent 256 characters
- Not enough to deal with the world’s character sets
- Unicode is a way to represent these character sets, but it is complicated
utf8

- After a long story, a committee created a Unicode standard called utf8
  - ASCII stuff unchanged
  - Variable size byte values to store an essentially infinite number of characters
New char types

- C++ allows for new char types
  - wchar_t: older, implementation dependent
  - char16_t and char32_t: C++11 for unicode
We’ll worry about this later

- This is just a complicated topic and we’ll worry about it later
  - Plenty of other problems in C++
Character Operations

- Example 6.2
Character Functions

- Page 92 of the book
- These are all tests of various kinds you can place on a character
  - Most are Booleans
#include<cctype>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum(c)</td>
<td>True if c is a letter or a digit.</td>
</tr>
<tr>
<td>isalpha(c)</td>
<td>True if c is a letter.</td>
</tr>
<tr>
<td>iscntrl(c)</td>
<td>True if c is a control character.</td>
</tr>
<tr>
<td>isdigit(c)</td>
<td>True if c is a digit.</td>
</tr>
<tr>
<td>isgraph(c)</td>
<td>True if c is not a space but is printable.</td>
</tr>
<tr>
<td>islower(c)</td>
<td>True if c is a lowercase letter.</td>
</tr>
<tr>
<td>isprint(c)</td>
<td>True if c is a printable character (i.e., a space or a character that has a visible representation).</td>
</tr>
<tr>
<td>ispunct(c)</td>
<td>True if c is a punctuation character (i.e., a character that is not a control character, a digit, a letter, or a printable whitespace).</td>
</tr>
<tr>
<td>isspace(c)</td>
<td>True if c is whitespace (i.e., a space, tab, vertical tab, return, newline, or formfeed).</td>
</tr>
<tr>
<td>isupper(c)</td>
<td>True if c is an uppercase letter.</td>
</tr>
<tr>
<td>isxdigit(c)</td>
<td>True if c is a hexadecimal digit.</td>
</tr>
<tr>
<td>tolower(c)</td>
<td>If c is an uppercase letter, returns its lowercase equivalent; otherwise returns c unchanged.</td>
</tr>
<tr>
<td>toupper(c)</td>
<td>If c is a lowercase letter, returns its uppercase equivalent; otherwise returns c unchanged.</td>
</tr>
</tbody>
</table>
strings

- our first STL container
Standard Template Library (STL)
More STL

- Containers
  - Data structures to hold other data with various capabilities/efficiencies
    - Most are templated
- Generic Algorithms
  - Algorithms for common tasks that work with container contents (mostly)
- Iterators
  - A kind of pointer, allowing access to containers independent of type
String Class Library

- A string is an STL class used to represent a **sequence of characters**
  - An STL sequence, but not templated as it can only hold characters
  - Templated containers can hold any type
- As with other classes we have seen, there is a representation for the string objects and a set of operations
- **Use** `#include<string>`
Objects and Methods

- A string is a C++ object. The word object has special meaning in programming, but there are two we care about for the moment.
  - What data it stores
  - What methods we can call
First Strings

- Example 6.3
Declaring Strings

- `string my_str;`
  - Creates a string object and initializes it to the empty string ""
- `const string my_str = "tiger";`
  - Creates a string object with 6 characters
Internal Structure

- Each element in a string is a single character
  - `char my_char = 'a';`
- In this case, a string is a sequence of `char` type elements
- Thus a variable of type `string` can hold a large number of individual characters
Copy Assignment

- Declaration
  - `string str1, str2 = “tiger”;`
- Assignment
  - `str1 = str2;`
- Makes a copy of `str2` so

```
str1: tiger
str2: tiger
```
Other ways to initialize a string

- `{ }` contains universal initializer, a list of elements to go in the string
- Since strings hold characters, we list individual characters

```cpp
string first{‘H’, ’o’, ’m’, ’e’, ’r’};
cout << first << endl;
// prints Homer
```
More initializers

- Can create copies of an individual character in a string
  - First arg is the count
  - Second arg is the characters

```cpp
string a_5(5, 'a');
cout << a_5 << endl;
// prints aaaa
```
More initializers

- Copy construction is technically different from assignment, but it does the same kind of thing

```cpp
string first = "Homer";
string second = first;
cout << second << endl;
```

prints Homer

It’s a *copy* of the original
We worry about copying

- If we copy a long string (say a copy of Shakespeare as a string) we do a lot of work
  - We have to make memory (which the string class does) to hold it
  - We have to use the CPU to move all that data around
- We will discuss this more
Methods, like functions

- A method is a function that is:
  - called in the context of a particular instance of an object
  - uses the dot notation for the call
Example methods size() and length()

- `string my_str = “tiger”;`
- `size()` method returns the number of characters in the string
- `cout << my_str.size();`
- Will output the integer 5
- `.length()` is the same as `.size()`
Data member: Subscript

- To access individual characters in a string, use the `.at` member function
  - Index starts at 0
- `string my_str = "tiger";`
  - `my_str`  
    - `t i g e r`
    - 0 1 2 3 4
- `cout << my_str.at(2);`
- Outputs the character `g`
[] instead of .at

- You can also use the subscript operator [].

```cpp
string my_string;
my_string = "hello";
cout << my_string[4] // output is 'o'
```
[] vs .at

- There is one important difference:
- If you access a non-existent index
  - .at will throw an error
  - [] will not (it will do something weird, but not throw an error)
Starting at 0

- On of the most important things to remember about strings (or any sequence in C+) is that they start at 0
  - Same as in Python and Java
- You will save yourself grievous headaches if you remember this!
Can assign values

- You can assign using the .at or [] operator

```cpp
string my_str;
my_str = "hello";
my_str[0] = 'j';
// string is now jello
my_str.at(0) = 'h';
// back to hello
```
Subscript Assignment

```cpp
string my_str = "tiger";
my_str.at(2) = 'm';
cout << my_str;
```
- Outputs “timer”
Assign Method

- You can also use the `assign` method and get `substring` assignment

```cpp
string a_str;
a_str = "myTry";
string next_str;
next_str.assign(a_str, 2, string::npos);
// next_str becomes "Try"
```
string::npos

- The :: is the scope resolution operator.
- It gives you access to functions and variables that are defined as part of a class.
- `string::npos` is the name of a variable within the string class.
- It stands for “no position”, a position not found in the string.
Character Processing

```cpp
string my_str = "tiger";
for (int i = 0; i < my_str.size(); i++) {
    cout << i << ": " << my_str[i] << endl;
}
```

Output: 0: t 1: i 2: g 3: e 4: 4
not int, string::size_type

- Every STL container has a size_type.
- For strings it is string::size_type.
- You shouldn’t use use int

```cpp
string my_str = "tiger";
for (decltype(my_str.size()) i = 0; i < my_str.size(); i++)
{
    cout << i << ": " << my_str[i] << endl;
}
```

Whatever size returns is size_type
size_types are unsigned

- As for all unsigned types, you can get some strange behavior if you go below 0.
- Watch for that (try it, see what it prints).
String input

- Example 6.4
Some regular functions: I/O

- Input operator `>>` is overloaded:
  ```cpp
  string my_str;
  cin >> mystr;
  ```
- Reads first word in `istream` up to whitespace
- If input is "fred", `my_str` is "fred"
- If input is "mary jones", `my_str` is only "mary"
More I/O, full line input

- To read a whole line of text (up to a newline character, ‘\n’) use
  - `getline(cin, my_str);`
- If input is “Mary Jones likes cats\n” then `my_str` is “Mary Jones likes cats”
  - ‘\n’ not included (is discarded)

```cpp
my_str  Mary Jones likes cats
```
for-each loop

- Example 6.5
for-each loop

- Much better loop
  - Similar to the for-loop in Python
  - Is a C++11 thing

```cpp
string my_str = "tiger";
for (auto chr : my_str)
    cout << chr << ", ";
```

C++ can determine the type of each element so we just `auto` the type
String Comparison

- Beginning at character 0 (leftmost), compare each character until a difference is found.
- The ASCII values of those different characters determine the comparison value.
- E.g. “aardvark” < “ant” since the second characters ‘a’ < ‘n’ because 97 < 110.
String Ops

- Example 6.6
Concatenation

- Concatenation appends one string to another.

```cpp
string result;
string tig = "tiger";
string ant = "ant";
result = tig + ant;
cout << result;
```

- Output is "tigerant"
Substrings

- The method is `substr`

```cpp
string my_str = "abc123";
mystr.substr(0, 4) // Starts at 0, length 4
  "abc1"
```

- If length is past end or no length argument, assume to the end

```cpp
my_str.substr(1, 100)
my_str.substr(1);
my_str.substr(1, string::npos)
  "bc123"
```

```cpp
Same thing
```
Another Initializer

- You can do this at the initializer stage
  
  ```
  string last = "Simpson";
  string sub_last(last, 3, 2);
  ```

- copy from last
  - start at index 3
  - length of 2
  - Prints ps
Constructors

- Methods / functions called in the context of initializing a newly declared variable are called constructors.
- Can have multiple based on arguments.
- All the initializers we’ve seen are constructors.
- We will write our own for our new classes later.
Some general seq ops

```cpp
string my_str = "abc";
// push_back: append 1 element to end
my_str.push_back('d'); // "abcd"
// append string at end
my_str.insert(my_str.size(), "efgh");
```
More String operations

- Table 9.13 on page 363
- www.cplusplus.com/reference/string/string
String find function

- Example 6.7
find function

- `find` finds the first occurrence of char in a string, starting at the start position.

```cpp
string my_str = "hello world"
string::size_type pos = 0;
pos = my_str.find('e', pos);
// pos gets set to 1
// doesn't exist? return string::npos
```
Lots of find functions

- Look at Table 9.14 (page 365). Works for characters and strings
  - `s.rfind(arg)`: finds the last of arg in s
  - `s.find_first_of(arg)`: first of any of the args in s
  - `s.find_last_of(arg)`: find last of any of the args in s
  - `s.find_first_not_of(args)`: find first of any char in s that is not in arg
  - `s.find_last_not_of(args)`: find last of any char in s that is not in arg
Lychrel Number

- Example 6.8