Chars
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

```c
char my_char = 'a';
```
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td> </td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>@</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>!</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>&quot;</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>#</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>$</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>%</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>&amp;</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>'</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>(</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>)</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>*</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>+</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>,</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>-</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>.</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>/</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>1</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>2</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>3</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>4</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>5</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>6</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>7</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAN</td>
<td>56</td>
<td>38</td>
<td>068</td>
<td>8</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>9</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>032</td>
<td>SUB</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>:</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>033</td>
<td>ESC</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>;</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>[</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>034</td>
<td>FS</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>&lt;</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>035</td>
<td>GS</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>=</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>]</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>036</td>
<td>RS</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>&gt;</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>^</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>037</td>
<td>US</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>?</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>137</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
After a long history, 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
Which of the following should be on the exam?

- `char`
- `wchar_t`
- `char16_t`
- `char32_t`
We’ll worry about this later

- This is just a complicated topic and we’ll not worry about it here
  - 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 3.3: cctype Functions

<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>

- chars and strings
Strings
strings

- our first STL container
Standard Template Library (STL)

Containers
- string
- deque
- vector
- map

Iterators

Generic Algorithms
- find
- count
- remove
- sort
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>`
What library is std::string provided in?

- `#include <string>`
- STL
- The Standard Library
- I don't know
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 5 characters

```c++
my_str  t i g e r
   0 1 2 3 4
```
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: t i g e r

str2: t i g e r
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 aaaaa
```
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 members and Subscripts

- To access individual characters in a string, use the `.at` member function
  - Index starts at 0

```cpp
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
text 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"
```
More String Methods
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: r
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;
}
```
or ...

```cpp
string::size_type i = 0; ...
```
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).
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)

```
my_str = Mary Jones likes cats
```
String input

- Example 6.4
for-each loop

- Example 6.5
for-each loop (range-based for 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 determines 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"
```
Another Initializer

- You can do this at the initializer stage

```c
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

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

- [https://en.cppreference.com/w/cpp/string/basic_string](https://en.cppreference.com/w/cpp/string/basic_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