Arrays and Strings
Arrays and Pointers

• In C, array names are just aliases that can be used as pointers
  ```
  int  y[] = {2, 3, 4, 5};  // these two are
  int *y   = {2, 3, 4, 5};  // roughly equivalent
  ```

• Indexing and dereferencing pointers are equivalent
  Side note: you can do arithmetic with pointers!
  ```
  *y  ≡  y[0]      *(y+1)  ≡  y[1]
  ```
Arrays and Pointers

- Pointer types are important!
  - If \( x \) is an \texttt{int8_t*}, \( x[3] \) accesses element at byte offset \( 3 \times 1 = 3 \)
  - If \( y \) is an \texttt{int32_t*}, \( y[3] \) accesses element at byte offset \( 3 \times 4 = 12 \)
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;

What are the values of x and y at the end?
Pointers

```c
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```

1

1
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
Pointers

```c
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;

What about this?

p++;
*p = 9;
Arrays and Pointers

- The same is (roughly) true for C "strings" (arrays of chars)

```c
char  text[] = "hello";  // read-write
char *text     = "hello";  // read-only
```

![Diagram of array and pointer representation of "hello"]
C Strings

- C strings are a sequence of ASCII chars terminated with null char (‘\0’)
  - Declare and initialize (static/stack, no explicit size needed):
    - `char *name = "John Smith";`
    - `char name[] = "John Smith";`
  - Declare only (static/stack, size needed):
    - `char name[11];`
  - Declare only (heap, size needed):
    - `char *name = (char*) malloc (sizeof(char) * 11);`

- Useful functions (need to `#include <string.h>`)  
  - Find length: `strlen`
  - Copy string or convert / format data into string: `snprintf`
  - Convert to long / float: `strtol` / `strtof`
  - Compare strings: `strncmp`
  - Search for substring: `strstr`
Information = Bits + Context

### ASCII TABLE

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BELL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
</tbody>
</table>

Public domain. https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg
• In Java: `dest = str1 + str2;  // copy strings`  
  – What does this code do in C?  

• Need to copy all characters from one string to another  
  – First for `str1` and then for `str2`
Old solution: `strcpy`

- Basically:
  ```c
  void strcpy (char *dest, char *src)
  {
    for (int i = 0; src[i] != '\0'; i++) {
      dest[i] = src[i];
    }
  }
  ```

- What happens if `src` isn't null-terminated?

**OUT OF BOUNDS!!!**
Copying strings

• Using `strcpy` is now considered **unsafe**
  - You are **not permitted** to use it in CS 261
• Solution: specify a maximum length that is safe to copy
  - This is usually the allocated length of the destination
• Older alternative: `strncpy`
  - Requires a maximum length
  - However, it does not guarantee the result is null-terminated
• Newer alternative: `strcpy_s`
  - However, it is not in the C99 standard
• **Better alternative:** `snprintf`
  - Safe, C99-standard, and more powerful than the other two
• **printf** and **snprintf** are conceptually similar
  - The former prints to standard out
  - The latter "prints" to a string (character array)
  - The latter can also copy strings and convert to strings

- sniff(dest_str, max_size, "%s", src_str);       // copy string
- sniff(dest_str, max_size, "%d", int_var);      // int -> string
- sniff(dest_str, max_size, "%f", float_var);    // float -> string

int    printf ( char *format, ...)
int   sniff (char *buffer, int bufsize, char *format, ...)

destination string;   maximum size
must be at least   "bufsize" bytes
• How do we declare an array of strings?
Arrays of arrays

- Array of string (char*) pointers
  - Two (roughly) equivalent syntax choices
    - char *name[];
    - char **name;
  - Must allocate.Initialize each sub-array separately

- Command-line parameters
  - int main (int argc, char *argv[])
  - Example: "./program -a myfile.txt"
    - argc = 3
    - argv[0] = "./program"
    - argv[1] = "-a"
    - argv[2] = "myfile.txt"
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define STR_LEN 8

int main(int argc, char **argv)
{
    // check parameters
    if (argc != 3) {
        fprintf(stderr, "Usage: ./hello2 <fname> <lname>\n");
        exit(EXIT_FAILURE);
    }

    // convert name to "First L." format
    char fullname[STR_LEN];
    snprintf(fullname, STR_LEN, "%s %c.", argv[1], argv[2][0]);

    // output new full name
    printf("Hello, %s!\n", fullname);

    return EXIT_SUCCESS;
}
```