cse378 - Lecture 3

- Announcements
  - HW #0 posted
  - HW #1 posted today, due Mon Oct 6th.

- Today:
  - Finish up memory
  - Control-flow (branches) in MIPS
    - if/then
    - loops
    - case/switch
  - Start: Array Indexing vs. Pointers
    - In particular pointer arithmetic
    - String representation

---

Representing strings

- A C-style string is represented by an array of bytes.
  - Elements are one-byte ASCII codes for each character.
  - A 0 value marks the end of the array.

<table>
<thead>
<tr>
<th>ASCII code</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>space</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>ASCII code</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>Character</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>ASCII code</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>Character</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>del</td>
</tr>
<tr>
<td>ASCII code</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
<td>108</td>
<td>109</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Character</td>
<td>.</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>ASCII code</td>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>
Null-terminated Strings

- For example, “Harry Potter” can be stored as a 13-byte array.

```
72 97 114 114 121 32 80 111 116 116 101 114 0
```

Harry  Potter \0

- Since strings can vary in length, we put a 0, or null, at the end of the string.
  - This is called a null-terminated string

- Computing string length
  - We'll look at two ways.

What does this C code do?

```c
int foo(char *s) {
    int L = 0;
    while (*s++) {
        ++L;
    }
    return L;
}
```
Array Indexing Implementation of strlen

```c
int strlen(char *string) {
    int len = 0;
    while (string[len] != 0) {
        len ++;
    }
    return len;
}
```

Pointers & Pointer Arithmetic

- Many programmers have a vague understanding of pointers
  — Looking at assembly code is useful for their comprehension.

```c
int strlen(char *string) {
    int len = 0;
    while (string[len] != 0) {
        len ++;
    }
    return len;
}
```
What is a Pointer?

- A pointer is an address.
- Two pointers that point to the same thing hold the same address
- Dereferencing a pointer means loading from the pointer’s address
- A pointer has a type; the type tells us what kind of load to do
  - Use load byte (lb) for char *
  - Use load half (lh) for short *
  - Use load word (lw) for int *
  - Use load single precision floating point (ls) for float *
- Pointer arithmetic is often used with pointers to arrays
  - Incrementing a pointer (i.e., ++) makes it point to the next element
  - The amount added to the point depends on the type of pointer
    - pointer = pointer + sizeof(pointer’s type)
    - 1 for char *, 4 for int *, 4 for float *, 8 for double *

What is really going on here...

```c
int strlen(char *string) {
    int len = 0;
    
    while (*string != 0) {  
        string ++;
        len ++;
    }
    
    return len;
}
```
Pointers Summary

- Pointers are just addresses!!
  - “Pointees” are locations in memory
- Pointer arithmetic updates the address held by the pointer
  - “string ++” points to the next element in an array
  - Pointers are typed so address is incremented by sizeof(pointee)