Characters Lesson Outline

1. Characters Lesson Outline
2. Numeric Encoding of Non-numeric Data #1
3. Numeric Encoding of Non-numeric Data #2
4. Numeric Encoding of Non-numeric Data #2
5. Representing Characters
6. How Characters Are Represented #1
7. How Characters Are Represented #2
8. Representing Digits
9. Representing Punctuation
10. ASCII
11. ASCII Table #1
12. ASCII Table #2
13. ASCII Table #3
14. ASCII Table #4
15. ASCII Confirmation Program #1
16. ASCII Confirmation Program #2
17. ASCII Confirmation Program #3
18. ASCII Confirmation Program #4
19. A char is an int #1
20. A char is an int #2
21. Declaring char Scalar Variables #1
22. Declaring char Scalar Variables #2
23. char Like int Example
24. char Scalar Literal Constants
25. char Scalar Literal Constant Example
26. Using char Scalar Variables
27. Using char Scalar Variables Example
28. char Arrays #1
29. char Arrays #2
30. Character Array Example #1
31. Character Array Example #2
In Programming Project #4, we encoded (represented) entree options using integer values.

For example:
1. chicken parmigiani
2. beef rigatoni
3. vegetarian lasagna
4. linguini and clams

If we wanted, we could add other entree options.

For example:
5. eggplant pasta alla Norma
6. spaghetti and meatballs
Numeric Encoding of Non-numeric Data #2

1. chicken parmigiani
2. beef rigatoni
3. vegetarian lasagna
4. linguini and clams
5. eggplant pasta alla Norma
6. spaghetti and meatballs

The numbers in these cases have no standard meaning with respect to the items that they encode; they’ve been chosen essentially at random.
Numeric Encoding of Non-numeric Data #3

1. chicken parmigiani
2. beef rigatoni
3. vegetarian lasagna
4. linguini and clams
5. eggplant pasta alla Norma
6. spaghetti and meatballs

So, we see that we can encode **qualitative** (non-numeric) values with **quantitative** (numeric) values, using **arbitrary** but **fixed** and **distinct** numeric values to encode a set of qualities.

That is, the code values can be any `int` values, but:

- they can’t change at runtime;
- the same `int` value can’t be used to encode two different things.
Representing Characters

What’s the most important set of non-numeric values in computing? It’s the one that allows the computer to communicate with us in a way that makes sense to actual real live human beings: natural language.

The most efficient way for computers to communicate in a natural language is by writing. Writing is based on characters. Characters are non-numeric. So, we want a way to encode characters numerically.
How Characters Are Represented #1

Here’s a code you might have used to play secret code games when you were a kid:

'A' = 1, 'B' = 2, 'C' = 3, 'D' = 4, . . . , 'Z' = 26

Now that you’ve grown up and taken CS1313, you realize that the numbers that you choose can be arbitrary, as long as they’re fixed and distinct.

So you could just as easily choose:

'A' = 65, 'B' = 66, 'C' = 67, 'D' = 68, . . . , 'Z' = 90

This is a perfectly reasonable encoding, if the only characters that you care about are upper case letters.

What about lower case?
How Characters Are Represented #2

'A' = 65, 'B' = 66, 'C' = 67, 'D' = 68, . . . , 'Z' = 90

What about lower case?

Well, you could add, for example:

'a' = 97, 'b' = 98, 'c' = 99, 'd' = 100, . . . , 'z' = 122

Are these the only characters that you need?
Representing Digits

Another kind of very important character is a digit. Here’s a possible encoding of the decimal digits:

'0' = 48, '1' = 49, '2' = 50, '3' = 51, . . ., '9' = 57

Notice that there’s an important distinction between the character to be represented, which happens to be a digit, and the numeric encoding, whose value **DOESN’T** have to have anything to do with the value of the digit being encoded.
Representing Punctuation

In addition to the upper case letters, the lower case letters and the digits, we also need to encode special characters such as punctuation.

This is starting to get pretty complicated, so maybe it’d help to have a standardized system.
ASCII

The *American Standard Code for Information Interchange* (ASCII)* is a standardized system for encoding characters numerically.

It has several categories of characters:

- **letters:**
  - upper case ('A' = 65 through 'Z' = 90);
  - lower case ('a' = 97 through 'z' = 122);
- **digits** ('0' = 48 through '9' = 57);
- **punctuation**
  - space = 32 through slash = 47;
  - colon = 58 through at sign = 64;
  - open square bracket = 91 through backquote = 96;
  - open curly brace = 123 through tilde = 126;
- **control characters**, encoded as 0 through 31; also **DEL** (encoded as 127).

* [http://www.asciitable.com/](http://www.asciitable.com/)
<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Kbd</th>
<th>Name</th>
<th>Code</th>
<th>Char</th>
<th>Kbd</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>Ctrl-A</td>
<td>Null</td>
<td>16</td>
<td>DLE</td>
<td>Ctrl-P</td>
<td>Data Line Escape</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>Ctrl-A</td>
<td>Start of Heading</td>
<td>17</td>
<td>DC1</td>
<td>Ctrl-Q</td>
<td>Device Control 1</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>Ctrl-B</td>
<td>Start of Text</td>
<td>18</td>
<td>DC2</td>
<td>Ctrl-R</td>
<td>Device Control 2</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>Ctrl-C</td>
<td>End of Text</td>
<td>19</td>
<td>DC3</td>
<td>Ctrl-S</td>
<td>Device Control 3</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>Ctrl-D</td>
<td>End of Transmission</td>
<td>20</td>
<td>DC4</td>
<td>Ctrl-T</td>
<td>Device Control 4</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>Ctrl-E</td>
<td>Enquiry</td>
<td>21</td>
<td>NAK</td>
<td>Ctrl-U</td>
<td>Negative Acknowledge</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>Ctrl-F</td>
<td>Acknowledge</td>
<td>22</td>
<td>SYN</td>
<td>Ctrl-V</td>
<td>Synchronous File</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>Ctrl-G</td>
<td>Ring Bell</td>
<td>23</td>
<td>ETB</td>
<td>Ctrl-W</td>
<td>End Transmission Block</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>Ctrl-H</td>
<td>Backspace</td>
<td>24</td>
<td>CAN</td>
<td>Ctrl-X</td>
<td>Cancel</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>Ctrl-I</td>
<td>Horizontal Tab</td>
<td>25</td>
<td>EM</td>
<td>Ctrl-Y</td>
<td>End of Medium</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>Ctrl-J</td>
<td>Line Feed</td>
<td>26</td>
<td>SUB</td>
<td>Ctrl-Z</td>
<td>Substitute</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>Ctrl-K</td>
<td>Vertical Tab</td>
<td>27</td>
<td>ESC</td>
<td>Ctrl-Shift-K</td>
<td>Escape</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>Ctrl-L</td>
<td>Form Feed</td>
<td>28</td>
<td>FS</td>
<td>Ctrl-Shift-L</td>
<td>File Separator</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>Ctrl-M</td>
<td>Carriage Return</td>
<td>29</td>
<td>GS</td>
<td>Ctrl-Shift-M</td>
<td>Group Separator</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>Ctrl-N</td>
<td>Shift Out</td>
<td>30</td>
<td>RS</td>
<td>Ctrl-Shift-N</td>
<td>Record Separator</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>Ctrl-O</td>
<td>Shift In</td>
<td>31</td>
<td>US</td>
<td>Ctrl-Shift-O</td>
<td>Unit Separator</td>
</tr>
</tbody>
</table>
## ASCII Table #2

<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Name</th>
<th>Code</th>
<th>Char</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td></td>
<td>Blank space</td>
<td>48</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>Exclamation point (or “bang”)</td>
<td>49</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>Double quote</td>
<td>50</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>Pound (or hash)</td>
<td>51</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td>Dollar sign (or “buck”)</td>
<td>52</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td>Percent</td>
<td>53</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td>Ampersand (or “and”)</td>
<td>54</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>’</td>
<td>Single quote</td>
<td>55</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>Open parenthesis</td>
<td>56</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>)</td>
<td>Close parenthesis</td>
<td>57</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>*</td>
<td>Asterisk (or “star”)</td>
<td>58</td>
<td>:</td>
<td>Colon</td>
</tr>
<tr>
<td>43</td>
<td>+</td>
<td>Plus</td>
<td>59</td>
<td>;</td>
<td>Semicolon</td>
</tr>
<tr>
<td>44</td>
<td>,</td>
<td>Comma</td>
<td>60</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td>Hyphen</td>
<td>61</td>
<td>=</td>
<td>Equals Sign</td>
</tr>
<tr>
<td>46</td>
<td>.</td>
<td>Period (or “dot”)</td>
<td>62</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>Slash</td>
<td>63</td>
<td>?</td>
<td>Question mark</td>
</tr>
</tbody>
</table>
## ASCII Table #3

<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Name</th>
<th>Code</th>
<th>Char</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>@</td>
<td>At</td>
<td>80</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>A</td>
<td></td>
<td>81</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>B</td>
<td></td>
<td>82</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>C</td>
<td></td>
<td>83</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>D</td>
<td></td>
<td>84</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>E</td>
<td></td>
<td>85</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td></td>
<td>86</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>G</td>
<td></td>
<td>87</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>H</td>
<td></td>
<td>88</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>I</td>
<td></td>
<td>89</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>J</td>
<td></td>
<td>90</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>K</td>
<td></td>
<td>91</td>
<td>[</td>
<td>Open square bracket</td>
</tr>
<tr>
<td>76</td>
<td>L</td>
<td></td>
<td>92</td>
<td>\</td>
<td>Backslash (or “bash”)</td>
</tr>
<tr>
<td>77</td>
<td>M</td>
<td></td>
<td>93</td>
<td>]</td>
<td>Close square bracket</td>
</tr>
<tr>
<td>78</td>
<td>N</td>
<td></td>
<td>94</td>
<td>^</td>
<td>Caret (or “fang”)</td>
</tr>
<tr>
<td>79</td>
<td>O</td>
<td></td>
<td>95</td>
<td>_</td>
<td>Underscore</td>
</tr>
</tbody>
</table>
## ASCII Table #4

<table>
<thead>
<tr>
<th>Code</th>
<th>Char</th>
<th>Name</th>
<th>Code</th>
<th>Char</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>`</td>
<td>Accent grave</td>
<td>112</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>a</td>
<td></td>
<td>113</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>b</td>
<td></td>
<td>114</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>c</td>
<td></td>
<td>115</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td></td>
<td>116</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>e</td>
<td></td>
<td>117</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>f</td>
<td></td>
<td>118</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>g</td>
<td></td>
<td>119</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>h</td>
<td></td>
<td>120</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>i</td>
<td></td>
<td>121</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>j</td>
<td></td>
<td>122</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>k</td>
<td></td>
<td>123</td>
<td>{</td>
<td>Open curly brace</td>
</tr>
<tr>
<td>108</td>
<td>l</td>
<td></td>
<td>124</td>
<td></td>
<td>Vertical bar (or “bar”)</td>
</tr>
<tr>
<td>109</td>
<td>m</td>
<td></td>
<td>125</td>
<td>}</td>
<td>Close curly brace</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td></td>
<td>126</td>
<td>~</td>
<td>Tilde</td>
</tr>
<tr>
<td>111</td>
<td>o</td>
<td></td>
<td>127</td>
<td>DEL</td>
<td>Delete</td>
</tr>
</tbody>
</table>
#include <stdio.h>

int main ()
{ /* main */
    const int first_printable_character_code = 32;
    const int last_printable_character_code = 126;
    const int program_success_code = 0;
    int index;

    for (index = first_printable_character_code;
       index <= last_printable_character_code;
       index++) {
        printf("ASCII Code #%3d is: %c\n", index, index);
    } /* for index */
    return program_success_code;
} /* main */
ASCII Confirmation Program #2

% gcc -o asciitest asciitest.c
% asciitest

ASCII Code # 32 is:    ASCII Code # 48 is: 0
ASCII Code # 33 is: !  ASCII Code # 49 is: 1
ASCII Code # 34 is: "  ASCII Code # 50 is: 2
ASCII Code # 35 is: #  ASCII Code # 51 is: 3
ASCII Code # 36 is: $  ASCII Code # 52 is: 4
ASCII Code # 37 is: %  ASCII Code # 53 is: 5
ASCII Code # 38 is: &  ASCII Code # 54 is: 6
ASCII Code # 39 is: '  ASCII Code # 55 is: 7
ASCII Code # 40 is: (  ASCII Code # 56 is: 8
ASCII Code # 41 is: )  ASCII Code # 57 is: 9
ASCII Code # 42 is: *  ASCII Code # 58 is: :
ASCII Code # 43 is: +  ASCII Code # 59 is: ;
ASCII Code # 44 is: ,  ASCII Code # 60 is: <
ASCII Code # 45 is: -  ASCII Code # 61 is: =
ASCII Code # 46 is: .  ASCII Code # 62 is: >
ASCII Code # 47 is: /  ASCII Code # 63 is: ?
ASCII Confirmation Program #3

ASCII Code # 64 is: @
ASCII Code # 65 is: A
ASCII Code # 66 is: B
ASCII Code # 67 is: C
ASCII Code # 68 is: D
ASCII Code # 69 is: E
ASCII Code # 70 is: F
ASCII Code # 71 is: G
ASCII Code # 72 is: H
ASCII Code # 73 is: I
ASCII Code # 74 is: J
ASCII Code # 75 is: K
ASCII Code # 76 is: L
ASCII Code # 77 is: M
ASCII Code # 78 is: N
ASCII Code # 79 is: O

ASCII Code # 80 is: P
ASCII Code # 81 is: Q
ASCII Code # 82 is: R
ASCII Code # 83 is: S
ASCII Code # 84 is: T
ASCII Code # 85 is: U
ASCII Code # 86 is: V
ASCII Code # 87 is: W
ASCII Code # 88 is: X
ASCII Code # 89 is: Y
ASCII Code # 90 is: Z
ASCII Code # 91 is: [ 
ASCII Code # 92 is: \ 
ASCII Code # 93 is: ] 
ASCII Code # 94 is: ^
ASCII Code # 95 is: _
ASCII Confirmation Program #4

ASCII Code # 96 is: `  ASCII Code #112 is: p
ASCII Code # 97 is: a  ASCII Code #113 is: q
ASCII Code # 98 is: b  ASCII Code #114 is: r
ASCII Code # 99 is: c  ASCII Code #115 is: s
ASCII Code #100 is: d  ASCII Code #116 is: t
ASCII Code #101 is: e  ASCII Code #117 is: u
ASCII Code #102 is: f  ASCII Code #118 is: v
ASCII Code #103 is: g  ASCII Code #119 is: w
ASCII Code #104 is: h  ASCII Code #120 is: x
ASCII Code #105 is: i  ASCII Code #121 is: y
ASCII Code #106 is: j  ASCII Code #122 is: z
ASCII Code #107 is: k  ASCII Code #123 is: {  
ASCII Code #108 is: l  ASCII Code #124 is: |
ASCII Code #109 is: m  ASCII Code #125 is: }
ASCII Code #110 is: n
ASCII Code #111 is: o
ASCII Code #123 is: {  
ASCII Code #124 is: |
ASCII Code #125 is: }  
ASCII Code #126 is: ~
A char is an int #1

```c
#include <stdio.h>

int main ()
{ /* main */
    const int first_printable_character_code =  32;
    const int last_printable_character_code = 126;
    const int program_success_code =   0;
    int index;

    for (index = first_printable_character_code;
        index <= last_printable_character_code;
        index++) {
        printf("ASCII Code #%3d is: %c\n",
            index, index);
    } /* for index */
    return program_success_code;
} /* main */
```

Notice that the variable named `index` is declared as an `int`, but in the `printf` statement, `index` can be used not only as an `int` but also as a `char`. The reverse is also true.
#include <stdio.h>

int main ()
{ /* main */
    const int program_success_code = 0;
    const char first_printable_character_code = 32;
    const char last_printable_character_code = 126;
    char index;

    for (index = first_printable_character_code;
        index <= last_printable_character_code;
        index++) {
        printf("ASCII Code #%3d is: %c\n",
            index, index);
    } /* for index */
    return program_success_code;
} /* main */

Notice that the variable named index is declared as a char, but in the printf statement, index can be used not only as a char but also as an int. The reverse is also true.
Declaring `char` Scalar Variables #1

Here’s a declaration of a `char` scalar variable:

```c
char first_initial;
```

This declaration tells the compiler to grab a group of bytes, name them `first_initial`, and think of them as storing a `char`.

**How many bytes in a `char` scalar?**

Each `char` scalar takes one byte:

```c
first_initial:
```

```c
   ______
  |       |
  |       |
  |       |
  |       |
  |_______|
```
Declaring char Scalar Variables #2

char first_initial;

first_initial: [ ]

REMEMBER: A char is just like an int, except that it uses fewer bytes:
typically, a char is 1 byte and an int is 4 bytes.
So, we can use char variables and constants in exactly the same ways that we use int variables and constants.
% cat charadd.c
#include <stdio.h>

int main ()
{ /* main */
    const int program_success_code = 0;
    int addend, augend;
    char sum;

    printf("What are the addend and augend?\n");
    scanf("%d %d", &addend, &augend);
    sum = addend + augend;
    printf("The sum is %d.\n", sum);
    return program_success_code;
} /* main */

% gcc -o charadd charadd.c
% charadd
What are the addend and augend?
1 4
The sum is 5.
char Scalar Literal Constants

A *character scalar literal constant* is a single char enclosed in single quotes:

```
'H'
```

Note that
```
''
```
is illegal.

However, you can also represent an individual char literal using the *octal* (base 8) code that represents it.

For example, the apostrophe character corresponds to ASCII code 39 decimal, which converts to 47 octal. So we can represent the apostrophe character like so:

```
'\047'
```
% cat apostrophe.c
#include <stdio.h>

int main ()
{
    const int program_success_code = 0;
    printf("Apostrophe: %c\n", '\047');
    return program_success_code;
}

% gcc -o apostrophe apostrophe.c
% apostrophe
Apostrophe: '
In C, we can use `char` scalar variables in many of the same ways that we use `int` scalar variables. As we saw, for example, we can declare them:

```c
char first_initial;
```

We can also assign `char` scalar values to `char` scalar variables, by enclosing them in single quotes:

```c
first_initial = 'H';
```

We can output `char` scalar values from `char` scalar variables, like so:

```c
printf("My first initial is %c.\n", first_initial);
```
Using char Scalar Variables Example

```c
#include <stdio.h>

int main ()
{
    const char computers_favorite_character = 'q';
    const int program_success_code = 0;
    char users_favorite_character;
    printf("What is your favorite character?\n\n");
    scanf("%c", &users_favorite_character);
    printf("Your favorite character is '%c'.\n", users_favorite_character);
    printf("My favorite character is '%c'.\n", computers_favorite_character);
    return program_success_code;
}
```

```
% gcc -o charscalar charscalar.c
% charscalar
What is your favorite character?
Z
Your favorite character is 'Z'.
My favorite character is 'q'.
```
In C, you can have an array of type `char`, just as you can have arrays of numeric types:

```c
char my_name[12];
```

We can fill this `char` array with characters and be able to print them out.
my_name[ 0] = 'H';
my_name[ 1] = 'e';
my_name[ 2] = 'n';
my_name[ 3] = 'r';
my_name[ 4] = 'y';
my_name[ 5] = ' '; 
my_name[ 6] = 'N';
my_name[ 7] = 'e';
my_name[ 8] = 'e';
my_name[ 9] = 'm';
my_name[10] = 'a';
my_name[11] = 'n';

Is this a good solution?
#include <stdio.h>

int main ()
{ /* main */
    const int my_name_length = 12;
    char my_name[my_name_length];
    int index;

    my_name[ 0] = 'H';
    my_name[ 1] = 'e';
    my_name[ 2] = 'n';
    my_name[ 3] = 'r';
    my_name[ 4] = 'y';
    my_name[ 5] = ' ';
    my_name[ 6] = 'N';
    my_name[ 7] = 'e';
    my_name[ 8] = 'e';
    my_name[ 9] = 'm';
    my_name[10] = 'a';
    my_name[11] = 'n';

    printf("My name is ");
    for (index = 0; index < my_name_length; index++) {
        printf("%c", my_name[index]);
    } /* for index */

    printf(".
");
    return 0;
} /* main */
Character Array Example #2

% gcc -o chararray chararray.c
% chararray

My name is Henry Neeman.

This is an improvement, but it’s still not an efficient way to assign a sequence of characters to a variable.

What we want is a kind of `char` variable whose use will be convenient for inputting, outputting and using sequences of characters.