In Programming Project #4, we encoded (represented) the entree using integer values:

0: no entree
1: burger
2: fish

If we wanted to add other entrees, we could have had, for example:

3: chicken
4: tofu
5: shrimp

The numbers in these cases have no standard meaning with respect to the entrees that they encode; they’ve been chosen essentially at random.

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

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, \ldots, "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 distinct, so you could just as easily choose:

```
"A" = 65, "B" = 66, "C" = 67, "D" = 68, \ldots, "Z" = 90
```

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

What about lower case?

Well, you could add, for example:

```
"a" = 97, "b" = 98, "c" = 99, "d" = 100, \ldots, "z" = 122
```

Are these the only characters that you need?
Digits & Punctuation

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.

But wait, there’s more!

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 chart.

The American Standard Code for Information Interchange (ASCII)^n

<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></td>
<td>Null</td>
<td>16</td>
<td>HEX</td>
<td></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>DCE</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></td>
<td>Enquiry</td>
<td>21</td>
<td>NAK</td>
<td>Ctrl-U</td>
<td>Negative Ack</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>Ctrl-P</td>
<td>Acknowledge</td>
<td>22</td>
<td>SYX</td>
<td>Ctrl-V</td>
<td>Synchronous File</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td></td>
<td>Ring Bell</td>
<td>23</td>
<td>EOT</td>
<td>Ctrl-W</td>
<td>End Transmission Block</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td></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></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-T-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-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-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-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-O</td>
<td>Unit Separator</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td>space</td>
<td>64</td>
<td>@</td>
<td></td>
<td>at</td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td></td>
<td>exclamation point</td>
<td>65</td>
<td>A</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td></td>
<td>double quote</td>
<td>66</td>
<td>B</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td></td>
<td>pound</td>
<td>67</td>
<td>C</td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>36</td>
<td>$</td>
<td></td>
<td>dollar sign</td>
<td>68</td>
<td>D</td>
<td></td>
<td>d</td>
</tr>
<tr>
<td>37</td>
<td>%</td>
<td></td>
<td>percent</td>
<td>69</td>
<td>E</td>
<td></td>
<td>e</td>
</tr>
<tr>
<td>38</td>
<td>&amp;</td>
<td></td>
<td>ampersand</td>
<td>70</td>
<td>F</td>
<td></td>
<td>f</td>
</tr>
<tr>
<td>39</td>
<td>(</td>
<td></td>
<td>left parentheses</td>
<td>71</td>
<td>G</td>
<td></td>
<td>g</td>
</tr>
<tr>
<td>40</td>
<td>)</td>
<td></td>
<td>right parentheses</td>
<td>72</td>
<td>H</td>
<td></td>
<td>h</td>
</tr>
<tr>
<td>41</td>
<td>,</td>
<td></td>
<td>comma</td>
<td>73</td>
<td>I</td>
<td></td>
<td>i</td>
</tr>
<tr>
<td>42</td>
<td>*</td>
<td></td>
<td>asterisk</td>
<td>74</td>
<td>J</td>
<td></td>
<td>j</td>
</tr>
<tr>
<td>43</td>
<td>+</td>
<td></td>
<td>plus</td>
<td>75</td>
<td>K</td>
<td></td>
<td>k</td>
</tr>
<tr>
<td>44</td>
<td>,</td>
<td></td>
<td>comma</td>
<td>76</td>
<td>L</td>
<td></td>
<td>l</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
<td></td>
<td>hyphen</td>
<td>77</td>
<td>M</td>
<td></td>
<td>m</td>
</tr>
<tr>
<td>46</td>
<td>.</td>
<td></td>
<td>period</td>
<td>78</td>
<td>N</td>
<td></td>
<td>n</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td></td>
<td>slash</td>
<td>79</td>
<td>O</td>
<td></td>
<td>o</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td></td>
<td>0</td>
<td>80</td>
<td>P</td>
<td></td>
<td>p</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td></td>
<td>1</td>
<td>81</td>
<td>Q</td>
<td></td>
<td>q</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td></td>
<td>2</td>
<td>82</td>
<td>R</td>
<td></td>
<td>r</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
<td></td>
<td>3</td>
<td>83</td>
<td>S</td>
<td></td>
<td>s</td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td></td>
<td>4</td>
<td>84</td>
<td>T</td>
<td></td>
<td>t</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td></td>
<td>5</td>
<td>85</td>
<td>U</td>
<td></td>
<td>u</td>
</tr>
<tr>
<td>54</td>
<td>6</td>
<td></td>
<td>6</td>
<td>86</td>
<td>V</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>55</td>
<td>7</td>
<td></td>
<td>7</td>
<td>87</td>
<td>W</td>
<td></td>
<td>w</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td></td>
<td>8</td>
<td>88</td>
<td>X</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>57</td>
<td>9</td>
<td></td>
<td>9</td>
<td>89</td>
<td>Y</td>
<td></td>
<td>y</td>
</tr>
<tr>
<td>58</td>
<td>;</td>
<td></td>
<td>colon</td>
<td>90</td>
<td>Z</td>
<td></td>
<td>z</td>
</tr>
<tr>
<td>59</td>
<td>'</td>
<td></td>
<td>single quote</td>
<td>91</td>
<td>open square bracket</td>
<td>open square bracket</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>(</td>
<td></td>
<td>less than</td>
<td>92</td>
<td>open curly bracket</td>
<td>open curly bracket</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>)</td>
<td></td>
<td>greater than</td>
<td>93</td>
<td>close square bracket</td>
<td>close square bracket</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>?</td>
<td></td>
<td>question mark</td>
<td>94</td>
<td>equals</td>
<td>equals</td>
<td>equals</td>
</tr>
<tr>
<td>63</td>
<td>.</td>
<td></td>
<td>greater than</td>
<td>95</td>
<td>underscore</td>
<td>underscore</td>
<td>underscore</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td>accent grave</td>
<td>97</td>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>98</td>
<td></td>
<td></td>
<td>b</td>
<td>99</td>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>d</td>
<td>101</td>
<td></td>
<td></td>
<td>e</td>
</tr>
<tr>
<td>102</td>
<td></td>
<td></td>
<td>f</td>
<td>103</td>
<td></td>
<td></td>
<td>g</td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
<td>h</td>
<td>105</td>
<td></td>
<td></td>
<td>i</td>
</tr>
<tr>
<td>106</td>
<td></td>
<td></td>
<td>j</td>
<td>107</td>
<td></td>
<td></td>
<td>k</td>
</tr>
<tr>
<td>108</td>
<td></td>
<td></td>
<td>l</td>
<td>109</td>
<td></td>
<td></td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td>n</td>
<td>111</td>
<td></td>
<td></td>
<td>o</td>
</tr>
<tr>
<td>112</td>
<td></td>
<td></td>
<td>p</td>
<td>113</td>
<td></td>
<td></td>
<td>q</td>
</tr>
<tr>
<td>114</td>
<td></td>
<td></td>
<td>r</td>
<td>115</td>
<td></td>
<td></td>
<td>s</td>
</tr>
<tr>
<td>116</td>
<td></td>
<td></td>
<td>t</td>
<td>117</td>
<td></td>
<td></td>
<td>u</td>
</tr>
<tr>
<td>118</td>
<td></td>
<td></td>
<td>v</td>
<td>119</td>
<td></td>
<td></td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td>x</td>
<td>121</td>
<td></td>
<td></td>
<td>y</td>
</tr>
<tr>
<td>122</td>
<td></td>
<td></td>
<td>z</td>
<td>123</td>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>124</td>
<td></td>
<td></td>
<td>b</td>
<td>125</td>
<td></td>
<td></td>
<td>i</td>
</tr>
<tr>
<td>126</td>
<td></td>
<td></td>
<td>c</td>
<td>127</td>
<td></td>
<td></td>
<td>z</td>
</tr>
</tbody>
</table>
What Is 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, such as Ctrl-C (encoded as 3); also DEL (encoded as 127).

ASCII Confirmation Program

```c
#include <stdio.h>
int main ()
{
    const int first_printable_character_code = 32;
    const int last_printable_character_code = 126;
    int index;
    for (index = first_printable_character_code; index <= last_printable_character_code; index++)
    {
        printf("ASCII Code #%3d is: %c\n", index, index);
    }
    return 0;
}
```

Output appears on the next page.


A \textit{char} is an \textit{int}

#include <stdio.h>

int main ()
{ /* main */
  const int
    first_printable_character_code = 32;
  const int
    last_printable_character_code = 126;
  int index;
  for (index = first_printable_character_code;
       index <= last_printable_character_code;
       index++)
  { printf("ASCII Code #%3d is: %c\n", index, index);
    index = index + 1;
  }
  /* for index */
  return 0;
} /* main */

Notice that the variable named \textit{index} is declared as an \textit{int}, but in the \texttt{printf} statement, \textit{index} can be used not only as an \textit{int} but also as a \textit{char}. The reverse is also true:

#include <stdio.h>

int main ()
{ /* main */
  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);
    index = index + 1;
  }
  /* for index */
  return 0;
} /* main */

#include <stdio.h>

int main ()
{ /* main */
  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);
    index = index + 1;
  }
  /* for index */
  return 0;
} /* main */
Declaring char Scalar Variables

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: [###]
```

REMEMBER: a char variable is just like an int variable, except that it uses fewer bytes — typically, a char is 1 byte and an int is 4 bytes.

So, we can use chars in exactly the same ways that we use ints.

```c
% cat charadd.c
#include <stdio.h>

int main ()
{
 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 0;
}
```

```c
% 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 either in single quotes:

```c
'H'
```

Note that

```c
''
```

is illegal.

However, you can also represent an individual char literal using the octal 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:

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

int main ()
{
 int main ()
 {}
 printf(" Apostrophe: %c\n", \047);  
 return 0;
}
```

```c
% gcc -o apostrophe apostrophe.c
% apostrophe
Apostrophe: ’
```
Using char Scalar Variables

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);
```

Here's an example:

```c
#include <stdio.h>

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

We can fill this char array with characters and be able to print them out:

```c
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';
```

The problem here is how much of a pain it'd be to have to write words and sentences this way, one scalar char at a time.

char Arrays

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:

```c
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';
```
Character Array Example

```c
#include <stdio.h>

int 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]);
    }
    printf(".
    return 0;
}
```

Character Strings

A character string is a sequence of characters with the following properties:

- it is stored like a char array;
- it is used like a char scalar.

In C, we declare a character string like so:

```c
char my_name[my_name_length+1];
```

Notice that a character string is declared exactly like a char array; in fact, a character string is a char array.

The only difference in the declaration is that the length of the char string is one greater than the number of characters to be stored.

The last character in any C character string is the null character, called NUL, which corresponds to integer value 0:

```
\0
```

Thus, the null character (integer 0) is often referred to as the character string terminator.

In general, a numeric value that is used to indicate that a particular state has been reached — for example, the end of a list — is called a sentinel value.

So, the character string terminator NUL is a sentinel that indicates the end of the string in question.
Character String Assignment Example

```c
#include <stdio.h>

int main ()
{ /* main */
    const int my_name_length = 12;
    char my_name[my_name_length + 1];
    my_name = "Henry Neeman";
    printf("My name is %s\n", my_name);
    return 0;
} /* main */
```

When compiling with `gcc -o charstrassnbad charstrassnbad.c`
charstrassnbad.c: In function 'main':
charstrassnbad.c:8: incompatible types in assignment

The version above seems like it should work, but it doesn't!

```c
#include <stdio.h>
#include <string.h>

int main ()
{ /* main */
    const int my_name_length = 12;
    char my_name[my_name_length + 1];
    strcpy(my_name, "Henry Neeman");
    printf("My name is %s\n", my_name);
    return 0;
} /* main */
```

When compiling with `gcc -o charstrassn charstrassn.c`
charstrassn

This version works!

```
my_name: Henry Neeman
```

17