1

```
/**
 * string-0.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Prints a string, one character per line.
 *
 * Demonstrates strings as arrays of chars and use of strlen.
 */

#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    // get line of text
    string s = GetString();

    // print string, one character per line
    for (int i = 0; i < strlen(s); i++)
    {
        printf("%c\n", s[i]);
    }
}
```

2

```
/**
 * string-1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Prints a string, one character per line.
 *
 * Demonstrates error checking.
 */

#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    // get line of text
    string s = GetString();

    // print string, one character per line
    if (s != NULL)
    {
        for (int i = 0; i < strlen(s); i++)
        {
            printf("%c\n", s[i]);
        }
    }
}
```

3

```
/**
 * string-2.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Prints a string, one character per line.
 *
 * Demonstrates optimization of a loop.
 */

#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    // get line of text
    string s = GetString();

    // print string, one character per line
    if (s != NULL)
    {
        for (int i = 0, n = strlen(s); i < n; i++)
        {
            printf("%c\n", s[i]);
        }
    }
}
```

4

```
void set_array(int array[4]);
void set_int(int x);

int main(void)
{
   int a = 10;
   int b[4] = { 0, 1, 2, 3 };
   set_int(a);
   set_array(b);
   printf("%d %d\n", a, b[0]);
}

void set_array(int array[4])
{
   array[0] = 22;
}

void set_int(int x)
{
   x = 22;
}
```

Overview

Recall that variables are used to store values. Quite frequently, we may want to use multiple variables to store a sequence of values: like a sequence of 10 test scores, or 50 addresses. For situations like these, C has a data structure called an **array**: which stores multiple values of the same type of data. For instance, an array of `int`s would store multiple `int` values back-to-back. The **string** type that you have been using is really just an `array` of `char`s.

Key Terms

- array
- string
- size
- index
- null-terminator

Arrays

```
1  int ages[5];
```

0	1	2	3	4

```
2  ages[0] = 28;
```

0	1	2	3	4
28				

```
3  ages[1] = 15;
4  ages[2] = ages[1];
5  ages[3] = ages[1] - 1;
6  ages[4] = 17;
```

0	1	2	3	4
28	15	15	14	17

```
7  for (int i = 0; i < 5; i++)
8  {
9      ages[i] += 1;
10 }
```

0	1	2	3	4
29	16	16	15	18

Like variables, arrays are declared by first stating the type of the data to be stored, followed by the name of the array. In brackets after the name of the array is the **size** of the array: which defines how many values the array will hold. For example, line 1 at left declares an array of 5 `int`s.

You can visualize an array as a sequence of boxes, each one holding a value, and each one with a numbered **index**, which is a number that can be used to access a specific value in an array. In C, arrays are zero-indexed, meaning that the first item in an array has index `0`, the second item has index `1`, etc.

To access a particular value in an array, use the name of the array, followed by the desired index in brackets. Line 2 at left sets the value of the first item in the `ages` array (the one at index `0`) to 28.

The value at each array index can be treated like a normal variable. For example, you can change its value, apply arithmetic or assignment operators to it.

Since each value in an array is referenced by its index number, it's easy to loop through an array. Lines 7 through 10 define up a `for` loop, which iterates through the entire array, and increases each age value by 1.

Strings

In C, a `string` is represented as an array of `char` values. Thus, when we write a line like `string s = "CS50";`, this information is stored as an array of `char`s, with one character at each index. The final index of a string in C is the **null-terminator**, represented by `'\0'`. The null-terminator is the character that tells a `string` that the `string` is over, and that there are no more characters in the `string`.

```
string s = "CS50";
```

0	1	2	3	4
'C'	'S'	'5'	'0'	'\0'

Since a `string` is just an array, you can index into the `string` just like you would index into any other array in order to access the value of a particular character. For instance, in the example above, indexing into `s[0]` would give you the character `'C'`, the first character in the string `"CS50"`.

This also makes it very easy to use a loop to interate through a string and perform computation on each individual character within a string, by first initializing the loop counter to 0, and repeating until the last index of the string. The function `strlen` takes in a string as input, and returns the length of the string as an integer, which may help in determining how many times the loop should repeat.

Tips and Tricks (1/3)

Don't forget that uninitialized memory contains indeterminate values.

Elements initialized to 0:

```
int my_array[20] = {0};
```

Elements contain garbage values:

```
int my_array[20];
```

Tips and Tricks (2/3)

Because C arrays always start at index 0, the last element of an array of n elements will be at index $n-1$. An extremely common bug is to attempt to reach that last element at index n rather than index $n-1$. Unlike other languages, C will not prevent programs from accessing elements past the end (or before the beginning) of arrays even though doing so may result in a segmentation fault or other undefined behavior.

Do

```
string my_array[5];
my_array[4] = "This is the last string.";
```

Don't

```
string my_array[5];
my_array[5] = "This is the last string.";
```

Tips and Tricks (3/3)

Array names aren't variables. Whole arrays cannot be assigned in the same way that regular values are; ie. the contents of array foo cannot be assigned to another array bar just by saying bar = foo. Instead, you must copy over each element, one at a time (i.e. usually with a loop).

Do

```
for (int i = 0; i < SIZE; i++)
    my_copy[i] = my_array[i];
```

Don't

```
my_copy = my_array;
```

Pokemon

Write a program that prompts the user to input the names of five Pokemon. Store those Pokemon in an array, and randomly select one to print out.
HINT: Don't reinvent the wheel — a function already exists that will return a random number!

```
jharvard@run.cs50.net (~): ./a.out
Give me a Pokemon: Butterfree
Give me a Pokemon: Clefairy
Give me a Pokemon: Diglett
Give me a Pokemon: Growlithe
Give me a Pokemon: Rapidash
Clefairy, I choose you!
```