CS50 Reference Sheet #2

Arrays

Arrays are a type of data structure that can store a collection of elements of the same type. They are declared and accessed by an index (which starts at 0) inside of square brackets (i.e. ages[0], ages[1], etc.)

Arrays are declared in a similar way to variables, but must include square brackets and the number of elements required. The square brackets tell the computer that this is an array, and not just a variable with one element.

Example: int ages[3]; //means declare an array of three int's

Arrays are always indexed starting at 0. The last element of the array of n-elements is always n-1.

Arrays can also be multidimensional, by using multiple square brackets (i.e. argv[1][0], board[i][j]);

Strings

Strings are arrays of characters, or chars. Strings are not a native C datatype, rather CS50 creates a special string datatype for us in the cs50.h library.

To iterate through a string (meaning accessing each individual character), you will often use a for loop, where strlen is a function in the string.h library that returns (in this case) the length of string s.

```
for (int i = 0; i < strlen(s); i++)
    {
        // DO SOMETHING
    }
```

Command Line Arguments

Programs, like functions, can take in arguments. Thus far, we haven't passed any arguments to main, and have declared it like this:

```
int main(void)
```

However, we can also declare main like this:

```
int main(int argc, string argv[])
```

The parameters argc and argv provide a representation of the program's command line. argc is the number of strings (arguments) that make up the command line (including the program name), and argv is an array that contains those strings.

If the command line for instance is:

```
./pennies 31 1
```

then argc = 3, argv[0] = "./pennies", argv[1] = "31", argv[2] = "1". Going one level deeper, each string in the argv array is itself an array of chars, so we can index into the characters in argv[1] so that argv[1][0] = '3' and argv[1][1] = '1'.

Characters and ASCII Values

Everything that goes on under the hood of a computer is done in binary – the language of 0s and 1s.

In order to use binary to express alphabetic, numeric, and other characters, we need some kind of mapping between characters and numbers.

ASCII (American Standard Code for Information Interchange) is an encoding system that does exactly that.

For example, uppercase 'A' is represented by the number 65, and lowercase 'a' is represented by the number 97.

Since characters, or chars, are stored using their corresponding ASCII value, when we add or subtract a number from a char, we are adding or subtracting their ASCII value.

For instance, 'a' + 2 evaluates as 'c', and 'B' + ('a' – 'A'), which is the same as 'B' + 32, evaluates to 'b'.

Binary	ASCII Code	Letter
0100 0001	65	A
0100 0010	66	B
0100 0011	67	C
0100 0100	68	D
0100 0101	69	E
0100 0110	70	F
0100 0111	71	G
0100 1000	72	H
0100 1001	73	I
0100 1010	74	J
0100 1011	75	K
0100 1100	76	L
0100 1101	77	M
0100 1110	78	N
0100 1111	79	O
0101 0000	80	P
0101 0001	81	Q
0101 0010	82	R
0101 0011	83	S
0101 0100	84	T
0101 0101	85	U
0101 0110	86	V
0101 0111	87	W
0101 1000	88	X
0101 1001	89	Y
0101 1010	90	Z

Binary	ASCII Code	Letter
0110 0001	97	a
0110 0010	98	b
0110 0011	99	c
0110 0100	100	d
0110 0101	101	e
0110 0110	102	f
0110 0111	103	g
0110 1000	104	h
0110 1001	105	i
0110 1010	106	j
0110 1011	107	k
0110 1100	108	l
0110 1101	109	m
0110 1110	110	n
0110 1111	111	o
0111 0000	112	p
0111 0001	113	q
0111 0010	114	r
0111 0011	115	s
0111 0100	116	t
0111 0101	117	u
0111 0110	118	v
0111 0111	119	w
0111 1000	120	x
0111 1001	121	y
0111 1010	122	z