Review

- char word[50]
  - The length of the string to be stored must equal the size of the character array

Review

- char word[10]
  - What is the length of the longest string that can be stored in *word*?

Review

- char word[20] = “CS110”
  - what is the length of the above string?

Standard Input

Standard Input (stdin)

- Stream of data available to the program
  - continually get the next thing in the stream
  - not a set block of data that we can go backwards and look at things earlier in the stream
  - take the input as it comes
  - to retain, we need to store it in the program

A *scanf()* call

```c
#include <stdio.h>

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%i", &n);

    printf("You entered %i\n", n);

    return 0;
}
```

Program in terminal:

```
Enter a number: 10 is my input number
You entered 10
```

- *scanf()* looks for digits at the input stream
- when it read something that wasn't a digit, a space, it converted the digits it read into an integer and returned
- Entire piece of text would have been available to the program had the reading continued

Standard Input

- *scanf()* and *getchar()* both read from standard input
  - *scanf()* converts the next thing in the stream to an integer or floating-point number
  - *getchar()* just returns the next individual character

Blocking

- A function that reads from stdin (like *scanf()*) will block (pause execution) until new data is available from stdin
- *scanf()* makes the program stop and wait until there's new data in the input stream
- The keyboard input isn't sent to the input stream until the user hits enter

Input from files

We've worked with keyboard input so far

- the program stops and the input is not available until you hit enter
- New keyboard input is not available from stdin until the user presses enter

- Contents of a file can also be used as a program's stdin
- Command line syntax: *./program < filename*
- The contents of filename will be used as stdin for program
- **If < is reversed to > the output will be written to the file, overwriting the file's original contents**

Input from files

input.txt U

input.txt

```
1     35
```

```
(base) zwkbhowmiknb02:e14-letter_frequency kbhowmik$ ./stdin < input.txt
Enter a number: You entered 35
```

EOF

- Read everything from the standard input until there’s nothing else to read
  - We can use loop
- *getchar()* will return a special value called EOF if there’s nothing else to read from the stdin

ASCII Code

- The standard that defines each character means is known as ASCII
- *printf()* and *putchar()* interpret the value of *char* variables as an ASCII value
  - print out the corresponding character associated with its value
  - ASCII code/ integer value of A is 65. When we store A as a char, it’s storing 65
- char is an 8-bit value
  - 256 different values can be stored in one char variable

ASCII Code

```c
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);

    printf("The ASCII value of the input character: %i\n", c);

    return 0;
}
```

```
Enter a character: A
The ASCII value of the input character: 65
```

ASCII Code

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
0	000	00	00000000	NUL	&#000;		Null char
1	001	01	00000001	SOH	&#001;		Start of Heading
2	002	02	00000010	STX	&#002;		Start of Text
3	003	03	00000011	ETX	&#003;		End of Text
4	004	04	00000100	EOT	&#004;		End of Transmission
5	005	05	00000101	ENQ	&#005;		Enquiry
6	006	06	00000110	ACK	&#006;		Acknowledgment
7	007	07	00000111	BEL	&#007;		Bell
8	010	08	00001000	BS	&#008;		Back Space
9	011	09	00001001	HT			Horizontal Tab
10	012	0A	00001010	LF
	Line Feed
11	013	0B	00001011	VT	&#011;		Vertical Tab
12	014	0C	00001100	FF			Form Feed
13	015	0D	00001101	CR
	Carriage Return
14	016	0E	00001110	SO	&#014;		Shift Out / X-On
15	017	0F	00001111	SI	&#015;		Shift In / X-Off
16	020	10	00010000	DLE	&#016;		Data Line Escape
17	021	11	00010001	DC1	&#017;		Device Control 1 (oft. XON)
18	022	12	00010010	DC2	&#018;		Device Control 2
19	023	13	00010011	DC3	&#019;		Device Control 3 (oft. XOFF)
20	024	14	00010100	DC4	&#020;		Device Control 4
21	025	15	00010101	NAK	&#021;		Negative Acknowledgement
22	026	16	00010110	SYN	&#022;		Synchronous Idle
23	027	17	00010111	ETB	&#023;		End of Transmit Block
24	030	18	00011000	CAN	&#024;		Cancel
25	031	19	00011001	EM	&#025;		End of Medium
26	032	1A	00011010	SUB	&#026;		Substitute
27	033	1B	00011011	ESC	&#027;		Escape
28	034	1C	00011100	FS	&#028;		File Separator
29	035	1D	00011101	GS	&#029;		Group Separator
30	036	1E	00011110	RS	&#030;		Record Separator
31	037	1F	00011111	US	&#031;		Unit Separator

- The first 32 characters are not printable characters
  - called special control characters
- Character with the value 0
  - null character
  - that is placed at the end of a string in an array of characters
  - can use character 0 instead of '\0'
- None of the values are *EOF*
  - out of the range that can be stored by a char
  - *EOF* actually an integer
  - usually represented with -1
  - *getchar()* will return EOF when there's nothing else to read
  - *EOF* will overflow a char
  - It's a value that cannot be properly represented using char

Reading from *stdin* until *EOF*

- An *int* is used to hold the result of *getchar()*
- *getchar()* is okay with taking an *int* instead of a *char* as the character it needs to print
- Continually take input from the user and print using *putchar()* as long as *getchar()* doesn't return EOF
- Utilize the fact that *c = getchar()* is an assignment expression
- **After the assignment happens, the value of this expression becomes the value that was assigned**

```c
#include <stdio.h>

int main() {
    int c;

    while ((c = getchar()) != EOF) {
        putchar(c);
    }

    return 0;
}
```

Reading from input file

```c
#include <stdio.h>

int main() {
    int c;

    while ((c = getchar()) != EOF) {
        putchar(c);
    }

    return 0;
}
```

input.txt U

input.txt

```
1    This is a test.
2
3    This test is for reading some text.
```

```
(base) zwkbhowmiknb02:e14-letter_frequency kbhowmik$ ./stdin < input.txt
This is a test.

This test is for reading some text.
```

Reading from *stdin* until *EOF*

- Without the condition for terminating the loop, we'll just get a stream of nonsense because *getchar()* is returning EOF and assigning it to *c* but when we're trying to print it out, since this is not a valid character, gibberish is printed
- Terminal doesn't know how to visually represent EOF

```c
#include <stdio.h>

int main() {
    int c;

    while ((c = getchar())) {
        putchar(c);
    }

    return 0;
}
```

Reading from terminal

```c
#include <stdio.h>

int main() {
    int c;

    while ((c = getchar()) != EOF) {
        putchar(c);
    }

    return 0;
}
```

- *getchar()* will block as long as the user types
- Upon hitting *enter,* the input will be printed using *putchar()*
- *getchar()* will block again for input from user
- *ctrl+d* can be used to indicate *EOF* for keyboard input

Letter Frequency

count	0	0	0	……	0	0	0
index	0	1	2	……	23	24	25
	a	b	c	……	x	y	z