The `char` Type

- The C type `char` stores small integers.
- It is 8 bits (almost always).
- `char` guaranteed able to represent integers 0 .. +127.
- `char` mostly used to store ASCII character codes.
- Only use `char` for characters.
- Even if a numeric variable is only use for the values 0..9, use the type `int` for the variable.

ASCII Encoding

- ASCII (American Standard Code for Information Interchange)
- Specifies mapping of 128 characters to integers 0..127.
- The characters encoded include:
  - upper and lower case English letters: A-Z and a-z
  - digits: 0-9
  - common punctuation symbols
  - special non-printing characters: e.g newline and space.
- You don't have to memorize ASCII codes
  Single quotes give you the ASCII code for a character:

```
printf("%d", 'a'); // prints 97
printf("%d", 'A'); // prints 65
printf("%d", '0'); // prints 48
printf("%d", ' ' + '\n'); // prints 42 (32 + 10)
```

- Don't put ASCII codes in your program - use single quotes instead.

Manipulating Characters

The ASCII codes for the digits, the upper case letters and lower case letters are contiguous.
This allows some simple programming patterns:

```
// check for lowercase
if (c >= 'a' && c <= 'z') {
  ...
```

```
// check is a digit
if (c >= '0' && c <= '9') {
  // convert ASCII code to corresponding integer
  numeric_value = c - '0';
}
```

Reading a Character - scanf

```
scanf("%d", &my_int);
scanf("%c", &my_char);
```

- scanning an int ignores whitespace
- scanning a char does not ignore whitespace
- We can ignore leading whitespace with chars:

```
scanf(" %c", &character);
```

Reading a Character - getchar

C provides library functions for reading and writing characters

- `getchar` reads a byte from standard input.
- getchar returns an int
- getchar returns a special value (EOF usually -1) if it can not read a byte.
- Otherwise getchar returns an integer (0..255) inclusive.
- If standard input is a terminal or text file this likely be an ASCII code.
- Beware input often bufferred until entire line can be read.

```
int c;
printf("Please enter a character: ");
c = getchar();
printf("The ASCII code of the character is %d\n", c);
```

Reading a Character - getchar

Consider the following code:

```
int c1, c2;

printf("Please enter first character:\n");
c1 = getchar();
printf("Please enter second character:\n");
c2 = getchar();
printf("First %d\nSecond: %d\n", c1, c2);
```

The newline character from pressing *Enter* will be the second character read.

Reading a Character - getchar

How can we fix the program?

```
int c1, c2;

printf("Please enter first character:\n");
c1 = getchar();
getchar(); // reads and discards a character
printf("Please enter second character:\n");
c2 = getchar();
printf("First: %c\nSecond: %c\n", c1, c2);
```

End of Input

- Input functions such as `scanf` or `getchar` can fail because no input is available, e.g., if input is coming from a file and the end of the file is reached.
- On UNIX-like systems (Linux/OSX) typing **Ctrl + D** signals to the operating system no more input from the terminal.
- Windows has no equivalent - some Windows programs interprert **Ctrl + Z** similarly.
- `getchar` returns a special value to indicate there is no input was available.
- This non-ASCII value is #defined as `EOF` in stdio.h.
- On most systems EOF == -1. Note getchar otherwise returns (0.255) or (0..127) if input is ASCII
- There is no end-of-file character on modern operating systems.

Reading Characters to End of Input

Programming pattern for reading characters to the end of input:

```
int ch;

ch = getchar();
while (ch != EOF) {
    printf("'%c' read, ASCII code is %d\n", ch, ch);
    ch = getchar();
}
```

For comparison the programming pattern for reading integers to end of input:

```
int num;
// scanf returns the number of items read
while (scanf("%d", &num) == 1) {
    printf("you entered the number: %d\n", num);
}
```