C is a high level language (HLL).
Its syntax is much the same as Java and C++, but no objects.

Actually, C++ and Java are much the same as C. C was designed in the 1970s.
Java
Why C?

1. Millions of lines of code have been written, so you are going to run into it.
   One place you are likely to run into it are the UW-Madison 500-level CS systems classes.

2. It gives a common language to reference, so the remainder of 354 can base examples on C code (not Java, C++, Ruby, Python, etc.).

3. Any C++ programmer also needs to know C.
Comments in C code

/* my comment */

opening delimiter  closing delimiter

comments may span lines
/* hello world program */

#include <stdio.h>

main() {
    printf("hello, world\n");
}
C’s `printf()` is like:

Java’s `System.out.print()`

or C++’s `cout`
Java source code

compiler

Java "byte codes"

Java interpreter (JVM)
high level language source code

compiler

assembly language source code

assembler

machine code

linking and loading

execution
high level language source code

C preprocessor

C compiler

assembly language source code
the C preprocessor does

1. `#include`

2. macro substitution
   for example:
   
   `#define MAX 100`

3. conditional compilation
#include <stdio.h>

#include "myheaders.h"
```c
#include <stdio.h>

#define MAX 100

main() {
    int x;

    x = 1;
    while (x <= MAX) {
        printf("%d\n", x);
        x++;
    }
}
```
C's basic types

- **char** – An ASCII character. Typically 1 byte.
- **int** – A 2's complement integer. Typically the size of a word on the architecture.
- **float** – An IEEE single precision floating point value. 32 bits.
- **double** – An IEEE double precision floating point value. 64 bits

Not listed here: pointer types.
Type qualifiers

If and when you believe you need to use these, look them up:

- **signed**
- **unsigned** — causes integer arithmetic operations to be ones for unsigned integers
- **long** — relates to size
- **short** — relates to size
- **const** — do not use this qualifier to define a constant. What happens when code tries to change a variable declared as `const` is implementation dependent.
C string

An array of characters, which uses the null character to delimit the end of the string.
'\0' is the null character. All zeros bit pattern in ASCII.

"Hi."

"12"
I/O Concept and Implementation

Logical places where input comes from and output goes to:

stdin
stdout
stderr
int printf(char *format [, arg1] ... );

function name

string, called the format string

optional, as referenced by string

return value:
# characters printed
printf("howdy");

A better way to do this (for security reasons) is puts("howdy");
Within the format string, to reference variable (values):

- %d for an integer
- %u for unsigned
- %c for a character
- %s for a string

Each `%character` references an argument
```c
int x, y;
x = 3;
y = 5;
printf("\%d + \%d = \%d\n", x, y, x + y);
```

```
3 + 5 = 8
```
```c
#include <stdio.h>

main()
{
    int x;
    x = 354;
    printf("[%d]\n", x);            // OUTPUT [354]
    printf("[%1d]\n", x);          // [354]
    printf("[%2d]\n", x);          // [354]
    printf("[%3d]\n", x);          // [354]
    printf("[%4d]\n", x);          // [ 354]
    printf("[%-4d]\n", x);         // [354 ]
}
```

Optional field width
#include <stdio.h>

main()
{
    float x;
    x = 1234.5678;
    printf("[%f]\n", x);          // OUTPUT
    printf("[ %f]\n", x);        // [1234.567749]
    printf("[1f]\n", x);         // [1234.567749]
    printf("[2f]\n", x);         // [1234.567749]
    printf("[20f]\n", x);        // [1234.567749]
    printf("[-20f]\n", x);       // [1234.567749]
    printf("[1.2f]\n", x);       // [1234.57]
    printf("[-2.3f]\n", x);      // [1234.568]
    printf("[-20.3f]\n", x);     // [1234.568]