What is C?

- C is a general-purpose, procedural, imperative programming language.
- C provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language.
- “C combines the power and performance of assembly with the flexibility and ease-of-use of assembly.”
- C is one of the most widely used programming languages of all time.
History of C & Unix

- The origin of C is closely tied to the development of the Unix operating system.

- Unix was developed by Ken Thompson and Dennis Ritchie as the operating system for a system called the PDP-7.

- C was developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs, and used to re-implement the Unix operating system for a system called a PDP-11.

- Prior to the development of Unix, most operating system programming was done in system-dependent assembly language. No portability of operating systems as a result!
History of the world

- Pretty much everything on the web uses those two things: C and Unix.
- Windows was once written in C,
- Unix underpins both Mac OS X and iOS.
- Linux is a derivative of Unix and powers the majority of the world's servers.
- The list goes on and on and on…
- Denis Ritchie passed in 2011
  http://www.wired.com/2011/10/thedennisritchieeffect/

left: Ken Thompson, right: Denis Ritchie
Yet another contribution

- In 1978, Brian Kernighan and Dennis Ritchie published the first edition of *The C Programming Language*

- This book, known to C programmers as "K&R"

- The second edition of the book covers the later ANSI C standard.
Is C still in wide use?
Modern C use cases

- C is widely used for "system programming", including implementing operating systems and embedded system applications.

- Other programming languages are often implemented in C. The implementations of Python, Perl 5 and PHP are all written in C.

- C allows efficient implementations of algorithms and data structures, which is useful for programs that perform a lot of computation.

- Excellent for learning! As you are “close to the metal”.

- Development of end-user applications has shifted to newer, higher-level languages, such as Java.
C vs Java
## Comparison to Java

<table>
<thead>
<tr>
<th>Thing</th>
<th>C</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>type of language</td>
<td>procedural</td>
<td>object-oriented</td>
</tr>
<tr>
<td>basic program unit</td>
<td>function</td>
<td>class</td>
</tr>
<tr>
<td>portability of source code</td>
<td>maybe</td>
<td>yes</td>
</tr>
<tr>
<td>portability of compiled code</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>compilation</td>
<td>creates machine language code</td>
<td>creates JVM ‘bytecode'</td>
</tr>
<tr>
<td>execution</td>
<td>OS loads and executes program</td>
<td>JVM loads byte code and interprets the program</td>
</tr>
<tr>
<td>memory management</td>
<td>manual</td>
<td>automatic</td>
</tr>
</tbody>
</table>

Comparison to Java: data types

- C has many of the same data types as Java
  - integer types: short, int, long
  - floating point types: float, double
- There are significant differences here in the type systems.
- While both Java and C are “statically typed”, Java is said to be “strongly & statically typed” whereas C is often said to be “weakly & statically typed”
Comparison to Java: operators

- C has the same set of mathematical operators as Java.
- Precedence and associatively rules related to operators are also the same.
Comparison to Java: control flow

- C has some of the same syntax same as Java…
  - if ( ) { } else { }
  - while ( ) { }
  - do { } while ( );
  - for(i=1; i <= 100; i++) { }
  - etc…
- Syntactically there is a fair amount of knowledge you can transfer
Comparison to Java: in general

- There are a number of similarities, but many more differences.
- It may look familiar to you, since C influenced a great number of languages.
- Syntax we take for granted as just “programming” are often descended from C.
- Don’t that that lull you into a false sense of security.
`Hello World’ in Java & C

```java
public class HelloWorld {
    /* A simple Java program */
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

```c
#include <stdio.h>
/* A simple C program */
int main()
{
    printf("Hello World\n");
    return 0;
}
```
#include <stdio.h>
/* A simple C program */
int main()
{
    printf("Hello World\n");
    return 0;
}
C’s Compile and Execute Cycle
Writing and running C

- Write text of program using an editor such as sublime, save as file e.g. my_program.c

- From command line, run the compiler to convert program from source to an “executable” or “binary”…
  
  
  gcc -Wall -g -o my_program my_program.c

  • (Compiler may give errors or warnings; edit source file, fix and re-compile)

  • From the command line, execute program like so…

  ./my_program

  • If runtime errors occur, debug, re-compile and execute.
Compilation command in detail

$ gcc -Wall -g -o my_program my_program.c

The compiler
Generate all warnings
Keep debugging information
Name the generated executable (default: a.out)
One or more C files
What really happens when we compile?

- When you type `gcc` you really initiate four different ‘stages’ that execute in sequence.
  - Preprocessing
  - Compilation
  - Assembly
  - Linking
Source code is “expanded” into a larger form that is simpler for the compiler to work with.

Any line that starts with ‘#’ is a line that is interpreted by the ‘preprocessor’.

Include files are “pasted in” (#include)

Comments are stripped out ( /* */ , // )

Continued lines (i.e. very long lines) are joined ( \ )

Other things like ‘macro expansion’
Compiling, Assembling & Linking

- The compiler converts the resulting text into binary code the CPU can run directly.

- The compilation process involves really several steps:
  - **Compiler**: C → Assembly
  - **Assembler**: Assembly → Object Code
  - **Linker**: Links object code and needed libraries into one executable file.
Viewing intermediate states of compilation

- We can run only pre-processing by giving the `-E` option to `gcc`.
  - ex. `gcc -E my_program.c > my_program.i`

- We can view the assembly generated by giving the `-S` option to `gcc`.
  - ex. `gcc -S my_program.c`
  - By default this will output to a file called `my_program.s`

- We can stop only generate the object code by giving the `-c` option to `gcc`.
  - ex. `gcc -c hello.c`
C Data Types
Data Types

- This table lists the data types in C together with their min size (in bytes) on a 32-bit and 64-bit systems.

<table>
<thead>
<tr>
<th>type</th>
<th>size (32bit)</th>
<th>size (64bit)</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
<td>char c = 'a';</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
<td>short s = 175;</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
<td>int i = 2147483647;</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>8</td>
<td>long int l = 2147483647;</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>8</td>
<td>long long int ll = 9223372036854775807;</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
<td>float f = 1.0;</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
<td>double d = 1.0;</td>
</tr>
<tr>
<td>pointer</td>
<td>4</td>
<td>8</td>
<td>int *x = NULL;</td>
</tr>
</tbody>
</table>

- There is also the concept of ‘signed’ and ‘unsigned’ integer types.
  - Using the keyword ‘unsigned’ changes the range of representation.
Data Types: Boolean

- Note that there was no ‘boolean’ type.
- C integer types represent true/false.
  - Zero is always interpreted as false
  - Any other value is interpreted as true.
- ex.

```c
int i = 0;
if (i) {
    printf("false");
} else
    printf("true");
```
Data Types: Char

- C’s ‘char’ is different than Java’s char type.
- It is more similar to Java's byte type, let's see if you can spot why.
- K&R defines a char as follows:
  - “a single byte, capable of holding one character in the local character set.”
- What's the problem here?
Data Types: Char

- C’s ‘char’ is different than Java’s char type.

- It is more similar to Java's byte type, let's see if you can spot why.

- K&R defines a char as follows:

  - “a single byte, capable of holding one character in the local character set.”

- What's the problem here?
  - Hint: How many bytes is Java’s char type?
Data Types: Char

- C’s ‘char’ is different than Java’s char type.
- It is more similar to Java's byte type, let's see if you can spot why.
- K&R defines a char as follows:
  - “a single byte, capable of holding one character in the local character set.”
- What's the problem here?
  - Hint: How many bytes is Java’s char type?
  - Modern character encodings require more bytes!!
Data Types: Pointers

- Java has no such data type as a pointer.
  - At least not explicitly.
- Java “Reference Types” in many ways behave similarly to pointers.
- When we learn about pointers in details I will argue that, if you understand Java “Reference Types” you in fact, already know a fair amount about pointers.
Printing Data Types

- The printf function provides a way of printing values of variables of different data types.

- We need to know the format specifier that printf expects for each given type.

- See types/types.c
Selection statements

- The syntax of the if, if ... else ... and switch statements are very similar to Java.

- The expression in the switch statement has to have an *integral* value (int, char, or an expression that evaluates to an integral value).

- See conditionals/conditionals.c
C also has three different loops: for loop, while loop, and do/while

Note that the control variable has to be declared before the loop!

You can use ‘break’ & ‘continue’ just as in Java.

See loops/loops.c

```c
int i;
for (i = 0; i < 10; i++) {
    print("i=%d\n",i);
}

int i = 10;
while (i > 0) {
    print("i=%d\n",i);
    i--;
}

int i = 10;
do {
    print("i=%d\n",i);
    i--;
} while (i > 0)
```
GOTO

- Don’t even think about it.
- I am not even going to explain it.
- They are generally considered to be bad programming style and result in code that is hard to understand and debug.
Reference
GCC argument summary

- GCC is the compiler, some of its options are...
  - –Wall : generate all warnings
  - –g : keep debugging information in another file
  - –o : followed by non-default name of executable
  - -E : only run preprocessor
  - -S : output only assembly
  - -c : output only object code
- Other options can be discovered by typing ‘man gcc’