CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
gcc
gcc (GNU C and C++ compiler)

- First compiler for GNU and adapted by many operating systems
- MPLABX is calling a customized variant of GCC to generate the hex file
- Supports both C and C++

\[ \text{Gnu} \]
\[ \text{Gnu's not Unix} \]
**gcc**

Basic Usage

**Syntax**

```bash
gcc -o outfile source files
```

- `-o` Sets the executable output name
  - Without argument defaults to `a` (a.exe on cygwin)
- **Source files** the set of source files to compile.
  - Example
    - `gcc SimpleMain.c`
    - `gcc -o mml mml_tester.c MatrixMath.c`
gcc
Object Files

- Object files allow individual compilation of source files (skips the link step)
  - Generally not runnable but have machine code for the source file
- In large projects this is essential as full compiles can take hours to complete
- .o files can then be compiled together without the flag as normal.
HTTP://XKCD.COM/303/

The #1 programmer excuse for legitimately slacking off:
"My code's compiling."

Hey! Get back to work!

Compiling!

Oh. Carry on.

525252
gcc
Object File Creation

Syntax

gcc -c source files

- \texttt{\textasciitilde}c \hspace{1em} \text{ Tells the compiler to skip linking}
- \texttt{Source files} \hspace{1em} \text{ the set of source files to compile into an object}
  - Each file will generate a different \texttt{.o} file
  - Example
    - gcc -c mml tester.c
foo.c

make

foo.o
make

- Command line tool designed to make the process of compiling code easier
- Parses a makefile to determine which actions to take
- MPLABX generates a makefile with the project and that is called when the hammer button is clicked
- Incredibly powerful tool as complex as C itself, will only cover the very basic commands
Invoking make

• Simply type “make” on the command line
• Make will attempt use a file called makefile and process targets from it
• If called without arguments it attempts to run the target all
• With arguments it attempts to create that specific target
• Smart enough to only execute targets that need updating
make
makefile contents

target: dependencies
actions to make target

newtarget: dependency1 dependency2
more actions
Selective Compilation with make

all: main.o foo.o

gcc < main.o foo.o

main.o: main.c
gcc -c main.c
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Battle Boats
NMEA 0183

'GPS'

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'S'</td>
<td>The start-of-message identifier, always a dollar-sign</td>
</tr>
<tr>
<td>MESSAGE_ID</td>
<td>A 3-character string identifying the type of message.</td>
</tr>
<tr>
<td></td>
<td>A comma separates the MESSAGE_ID from the subsequent data</td>
</tr>
<tr>
<td>DATA1,DATA2,DATA3,…</td>
<td>A comma-separated list of data, all encoded as ASCII characters</td>
</tr>
<tr>
<td>*XX</td>
<td>A message ends with an asterisk and then a checksum byte encoded as two separate ASCII hexadecimal characters (like '0A'). This checksum is calculated from ALL bytes between the 'S' and the '*'</td>
</tr>
<tr>
<td>\n</td>
<td>A newline character actually ends the string.</td>
</tr>
</tbody>
</table>

$£00,123,$
Agent A generates a random 16-bit number that is its "guess" along with another 16-bit number that is used as the encryption key.

Agent A then transmits a checksum of both its guess and key (which is an 8-bit XOR of all of their bytes) along with an encrypted version of its guess (which is a 16-bit XOR of the guess with the encryptionKey).

During this time Agent B is doing the same thing.
• Once Agent A has received Agent B's encrypted
guess and checksum, it transmits the unencrypted
guess and the encryption key (and Agent B does the
same).

• 5. Agent B can now verify Agent A's information by
verifying both the checksum and the encryption key
(and Agent A does the same).

• 6. Now both can agree on who should go first by
having either guessed higher or lower than the other
agent depending on if the XOR of the LSB of their
guesses is 1 or 0.
Sample Guess

606

Key

Guess

Checksum

Eyen

Key

Checksum

Alf
guess
key
guess
checksum

\text{guess}^\wedge \text{key} = \text{checksum}

\text{checksum}(\text{guess}, \text{key}) = \text{checksum}
Read byte( )

if (in)
  decode

max(porter1, porter2)
<table>
<thead>
<tr>
<th>Negotiation Data Set 1</th>
<th>$CHA, 37348, 117*46</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$DET, 9578, 46222*66</td>
</tr>
<tr>
<td>Negotiation Data Set 2</td>
<td>$CHA, 54104, 139*45</td>
</tr>
<tr>
<td></td>
<td>$DET, 32990, 21382*5e</td>
</tr>
<tr>
<td>Negotiation Data Set 3</td>
<td>$CHA, 62132, 70*79</td>
</tr>
<tr>
<td></td>
<td>$DET, 52343, 16067*50</td>
</tr>
<tr>
<td>Negotiation Data Set 4</td>
<td>$CHA, 36027, 55*7a</td>
</tr>
<tr>
<td></td>
<td>$DET, 7321, 36898*6e</td>
</tr>
<tr>
<td>HIT messages</td>
<td>$HIT, 3, 8, 1*43</td>
</tr>
<tr>
<td></td>
<td>$HIT, 0, 2, 0*4b</td>
</tr>
<tr>
<td></td>
<td>$HIT, 2, 3, 1*49</td>
</tr>
<tr>
<td></td>
<td>$HIT, 5, 6, 4*4e</td>
</tr>
<tr>
<td></td>
<td>$HIT, 0, 3, 0*4a</td>
</tr>
<tr>
<td></td>
<td>$HIT, 1, 7, 1*4e</td>
</tr>
<tr>
<td></td>
<td>$HIT, 4, 8, 0*45</td>
</tr>
<tr>
<td></td>
<td>$HIT, 5, 3, 3*4c</td>
</tr>
<tr>
<td></td>
<td>$HIT, 0, 5, 0*4c</td>
</tr>
<tr>
<td></td>
<td>$HIT, 5, 6, 1*4b</td>
</tr>
<tr>
<td></td>
<td>$HIT, 1, 1, 1*48</td>
</tr>
<tr>
<td></td>
<td>$HIT, 1, 0, 0*48</td>
</tr>
<tr>
<td></td>
<td>$HIT, 5, 2, 5*4b</td>
</tr>
<tr>
<td></td>
<td>$HIT, 2, 8, 0*43</td>
</tr>
<tr>
<td></td>
<td>$HIT, 0, 6, 0*4f</td>
</tr>
<tr>
<td></td>
<td>$HIT, 5, 9, 0*45</td>
</tr>
<tr>
<td></td>
<td>$HIT, 2, 8, 2*41</td>
</tr>
</tbody>
</table>
\$ H \text{IST}, 2, 8, 2 \times 4 \$

\[ a = [H \ \ I \ \ T, \ 2, 8, 2, \ 2, \ \ \ 0] \]

\[ \Rightarrow \text{check sum} = 0; \]

\[ \text{while}(a[i] \neq 0) \]
\[ \text{check sum} \land a[i]; \]
guess, key

hash = (guess >> 8) & 0xff;
hash ^= guess & 0xff;
\[ \text{guess} = \text{rand}() \Rightarrow \]

\[ \text{key} = \text{rand}(()) \]

\[ \text{eguess} = \text{guess} \wedge \text{key} \]

\[ \text{hash} \]

\[
\begin{array}{c}
\text{guess} \quad a \\
\text{hash} = a; \\
\text{hash}^\wedge = 6; \\
\vdash c; \\
\vdash d;
\end{array}
\]

\[
\begin{array}{c}
\text{key} \quad c \\
\end{array}
\]


$\text{eguess} = \text{guess}^\text{key}$

$\text{hash}(\text{guess}, \text{key}) = \text{hash}$
Agent run()

Protocol

// Generate events

switch (state)
    case state1:
        switch (event)
            case state2:
rand until hit