Week 1, continued

This is CS50. Harvard University. Fall 2014.

Cheng Gong

Table of Contents

Formula SAE at MIT

- Ansel Duff '15 speaks about engineering opportunity at MIT. Formula SAE builds electric cars and enters competitions. Engineers work directly on the car, while computer scientists write code controlling these cars. Contact fsae@mit.edu[1] for more information.

Introduction

- Experience the power of a bookbook™[2] is a humorous take on Apple's marketing, emphasizing an IKEA catalog's "features" like its ease of use and infinite battery life.
- Section by Friday at noon.

[1] mailto:fsae@mit.edu
[2] http://youtu.be/MOXQo7nURs0

- **Supersections**[3] next Sunday, with one less comfortable and one more comfortable.
- **Problem Set 0**[4] due Thursday at noon, unless a late day is used. **Problem Set 1**[5] will use C.
- Support available at **office hours** in four dining halls, with the schedule at http://cs50.harvard.edu/hours.
- **Questions** can also be asked and answered online at http://cs50.harvard.edu/discuss.
- Every Friday at 1:15pm CS50 hosts a **lunch** at Fire and Ice for students and staff to get to know each other. RSVP at http://cs50.harvard.edu/rsvp.

C

Functions and Syntax

- A volunteer from the audience, Alana, will represent the `[ say ]` block from Scratch. David will give her an **argument**, a white sheet of paper, on which he writes the words `hello, world`. By handing her the argument, David is telling her what to say.
- Now we have her put on a name tag that says `printf`. Alana writes the input given on the white paper on the touchscreen, simulating the effect of the `printf` function.
- Another volunteer, Javier, will represent the `GetString` function. He takes a white sheet of paper and gets a name from the audience, and then brings it back.
- Alana now gets a sheet of paper that has `hello, __` and the sheet of paper from Javier. She uses the name on the second sheet of paper, `Jonathan`, to fill in the blank on the first sheet.
- Recall that the **CS50 Appliance** is an operating system, called Ubuntu Linux, running in a window on your own computer, so everyone in the class has the same system to program in.
- `gedit` is the text editor in which we write the code, and terminal is the blinking prompt where we type in commands. It is a **CLI**, or command-line interface, as opposed to a **GUI**, a graphical user interface.

[3] http://cs50.harvard.edu/sections/1
[4] http://cs50.harvard.edu/psets/0
[5] http://cs50.harvard.edu/psets/1

Compilers, Commands, and Libraries

- Once we write code, recall that we need to enter commands like `make hello` that takes **source code** and converts it to **object code**, or zeroes and ones.
- The object code is run with a command like `./hello`, with the `.` representing the current directory, and the `/` separates the directory from the file name, so we are running the program named `hello` in the current folder.

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

- Let's take a closer look at the above code.
  - # Line 1 is including functions that people have written in the past, in this case functions in a file called `stdio.h`, in particular `printf`.
  - # Line 3, `main`, is like the `[ when (green flag) clicked ]` block, starting the program.
  - # Lines 4 and 6, curly braces, simply hold some code together.
  - # Line 5, `printf`, is a function that prints to a screen, with the parentheses surrounding the inputs to the function.
  - # Every time we call a function, we will use parentheses, even if they are empty.
  - # Strings are surrounded by double quotes.
  - # `\n` is telling `printf` to output a new line.
  - # Finally, a semicolon simply ends one instruction.
- You will learn to see the correct placement of these syntactical details as time goes on.
- Source code is passed into a compiler that generates patterns of zeroes and ones.
- `make` is not a compiler, but calls on `clang`, an actual compiler. Other compilers include Visual Studio and gcc. Let's compile and run:

```
jharvard@appliance (~): make hello
```

```
clang -ggdb3 -O0 -std=c99 -Wall -Werror    hello.c  -lcs50 -lm -o hello
```

- `rm` is the command to remove something, for example `rm hello`.
- So now we try to compile with `clang` ourselves:

```
jharvard@appliance (~): clang hello.c
jharvard@appliance (~): ./hello
bash: ./hello: No such file or directory
jharvard@appliance (~):
```

- `bash` is the name of the prompt we're using, and it can't find `./hello`.
- We run `ls` to see what is in the current directory, and see `a.out`. Back in the day, the default output for the compiler was decided to be `a.out`, so that is our compiled program.
- To rename a file, we use `mv`, short for move.

```
jharvard@appliance (~): mv a.out hello
jharvard@appliance (~): ./hello
hello, world
```

- Let's add code to our program:

```
#include <stdio.h>

int main(void)
{
    printf("state your name: ");
    string s = GetString();
    printf("hello, world\n");
}
```

- Now we get the same errors as last time:

```
jharvard@appliance (~): clang hello.c
hello.c:6:5: error: use of undeclared identifier 'string'; did you mean
 'stdin'?
    string s = GetString();
    ^~~~~
    stdin
...
```

- We need to include `cs50.h`, training wheels that include functions like `GetString`.

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    printf("state your name: ");
    string s = GetString();
    printf("hello, world\n");
}
```

- But the compiler doesn't know to combine the object code of both `hello` and the object code of `cs50.h`. So we have to say

```
jharvard@appliance (~): clang hello.c -lcs50
```

- Now the program compiles and runs as `a.out`, but still says only `hello, world`. We quickly fix this bug:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    printf("state your name: ");
    string s = GetString();
    printf("hello, %s\n", s);
}
```

- And now we run this, with `-o hello` to output a program named `hello` rather than the default `a.out`.

```
jharvard@appliance (~): clang -o hello hello.c -lcs50
```

- Since we don't want to remember these, and other arguments, to clang we simply use `make hello`.
- Recall that `printf` is in the library `stdio.h`. There's `\n`, `\r`, `\'`, `\"`, `\\`, `\0`, and other **escape sequences** that are special expressions that start with a backslash. For example `\"` is useful in printing the double quotes without causing an error by ending the string passed to `printf`.

- We also have placeholders like `%c` for printing a single character, `%d` for printing a decimal number, also written as `%i`, and `%s` for strings.
- In C, we also have different types of variables that can store different types of things. `char` stores a character, `float` is for a real number, `int` is for an integer, and a `long long` is for a really long number, more than an `int` can hold.
- In `CS50.h` we have two other data types, `string`, and `bool`, which can either be true or false. The library also contains `GetChar`, `GetInt`, `GetString`, etc, for getting a specific type of input from a user.

Conditions

- Conditions have the following structure:

```
if (condition)
{
    // do this
}
```

- The `//` in line 3 marks a comment, a note to yourself that has no impact on the program.
- There can also be two exclusive forks:

```
if (condition)
{
    // do this
}
else
{
    // do that
}
```

- Or three:

```
if (condition)
{
    // do this
}
else if (condition)
{
    // do that
}
else
{
    // do this other thing
}
```

- Incidentally, there are many other ways to write the same code, one being as follows:

```
if (condition) {
    // do this
} else if (condition) {
    // do that
} else {
    // do this other thing
}
```

- We prefer that braces are on their own lines, among other things, and will guide you along the way to a standard CS50 **style** that note these distinctions of new lines and tabs, for consistency and readability.

Boolean Expressions

- Boolean expressions can be combined with `&&` as "and", and `||` as "or":

```
if (condition && condition)
{
    // do this
}

if (condition || condition)
{
    // do this
}
```

Switches

- **Switches** are an alternate way of expressing `if` and `else if`:

```
switch (expression)
{
    case i:
        // do this
        break;

    case j:
        // do that
        break;

    default:
        // do this other thing
        break;
}
```

Loops

- Loops have various formats which we will return to:

```
for (initializations; condition; updates)
{
    // do this again and again
}
```

```
while (condition)
{
    // do this again and again
}
```

```
do
{
    // do this again and again
}
while (condition);
```

Variables

- Variables in C have a particular type. Here, the first line creates a new variable of the type `int`, and the second assigns a value of `0` to it.

```
int counter;
counter = 0;
```

- A more elegant way to write the above code:

```
int counter = 0;
```

Functions and Arguments

- Functions take arguments within parentheses:

```
string name = GetString();
printf("hello, %s\n", name);
```

- As an aside, jailbreaking an iPhone or generally any phone, is doing something the company didn't intend, such as installing applications from outside the App Store. This program, `iUnlock.c`[6], was written in C, and at the end of the day even advanced, familiar devices use the same code that we do in `hello.c`.
- Let's make another program, `adder.c`.

[6] http://cdn.cs50.net/2014/fall/lectures/1/w/src1w/iUnlock.c

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user for input
    printf("Give me an integer: ");
    int x = GetInt();
    printf("Give me another integer: ");
    int y = GetInt();

    // do the math
    printf("The sum of %i and %i is %i!\n", x, y, x + y);
}
```

- The last argument to `printf` in line 13, `x + y`, is simply what we want, the sum. Let’s compile and run it:

```
jharvard@appliance (~): make adder
clang -ggdb3 -O0 -std=c99 -Wall -Werror    adder.c  -lcs50 -lm -o adder
jharvard@appliance (~): ./adder
Give me an integer: 1
Giver me another integer: 2
The sum of 1 and 2 is 3!
jharvard@appliance (~):
```

- We can be even fancier with `conditions-0.c`:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    printf("I'd like an integer please: ");
    int n = GetInt();

    if (n > 0)
    {
        printf("You picked a positive number!\n");
    }
    else
    {
        printf("You picked a negative number!\n");
    }
}
```

- This will print whether the number is positive or negative. Let’s try to compile:

```
jharvard@appliance (~): make conditions-0
make: Nothing to be done for 'conditions-0'.
```

- So the mistake here was saving the file as `conditions-0` rather than `conditions-0.c`. Now we compile and run, but when we type in 0:

```
jharvard@appliance (~): ./conditions-0
I'd like an integer please: 0
You picked a negative number!
jharvard@appliance (~):
```

- So let’s add a third condition:

```
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    printf("I'd like an integer please: ");
    int n = GetInt();

    if (n > 0)
    {
        printf("You picked a positive number!\n");
    }
    else if (n < 0)
    {
        printf("You picked a negative number!\n");
    }
    else
    {
        printf("You picked zero!\n");
    }
}
```

```
jharvard@appliance (~): ./conditions-0
I'd like an integer please: 0
You picked zero!
jharvard@appliance (~):
```

- We conclude with Saroo Brierley: Homeward Bound[7] from Google, the story of a man who finds his home with the help Google Earth, testifying to the power of technology.

[7] http://www.youtube.com/watch?v=UXEvZ8B04bE