Week 1

This is CS50. Harvard University. Fall 2014.

Cheng Gong

Table of Contents

Introduction

- Colton Ogden's music can be downloaded at http://soundcloud.com/cs50.
- This week we move on to C, reusing all of the ideas we learned last week in Scratch.
- Recall that the `[ say [] ]` block is a statement, but also a function.
- `[ repeat [] ]` and `[ forever ]` are types of loops.
    # Mark Zuckerburg teaches REPEAT LOOPS[1] describes how a loop can do very powerful things, like wishing everyone on Facebook a happy birthday. Similarly, CS50 uses code to send people Dropbox coupon codes by iterating through email addresses.
- Hexagon blocks are boolean expressions inside conditions like `[ if < > ]`. Nested conditions can allow for three, four, or more branches.
    # Bill Gates explains IF & IF/ELSE statements[2] as how computers make decisions.

[1] http://www.youtube.com/watch?v=hYvcoRkAkOU
[2] http://www.youtube.com/watch?v=fVUL-vzrIcM

Announcements

- Sectioning Wednesday through Friday. **Supersections**[3] start Week 2, one more comfy one less comfortable. More details to come.
- **Sections**[4] start Week 3 (two weeks from now).
- **Questions** should go to heads@cs50.harvard.edu[5]
- **Problem Set 0** is now out.
- Support available at **office hours** in four dining halls, with the schedule at http://cs50.harvard.edu/hours.

How the PC Came to Be

- Two volunteers read a script with accompanying visuals about the history of the PC.
    # Paul Allen and Bill Gates started working with large computers called the PDP-10 family. After Allen discovered the MITS Altair 8800, they realized the potential of minicomputers and started writing a version of BASIC for the MITS. Despite setbacks and hurdles, the main program worked successfully and Allen and Gates sold their first commercial software as Microsoft.
- Today, three pages of the original source code hangs on campus in Maxwell Dworkin with Allen and Gates' signatures.
    # They set out to write a program that allows other people to write programs in code that looks more user-friendly:

```
10 PRINT "hello, world"
20 END
```

- Professor Harry Lewis gives a tour of the BASIC interpreter[6], showing us the code they wrote, talking about Allen and Gates, and pointing out a comment from Bill Gates about code that can be deleted to save memory space.

[3] http://cs50.harvard.edu/sections/1
[4] http://cs50.harvard.edu/sections
[5] mailto:heads@cs50.harvard.edu
[6] http://www.youtube.com/watch?v=FeUi67qU7fo

Source Code

- A theme of computer science is layering, or **abstraction**, building on the works of others before you.
- Today we'll start to look at source code that looks like this:

```
#include <stdio.h>

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

- A special program called a **compiler** converts the above **source code** to zeroes and ones, or **object code** that are understood by a CPU to ultimately print "hello world".

C

- Comparing Scratch and C, there is roughly a one-to-one correlation in the code below: `[ say ]` in Scratch is akin to `printf()` in C, and `[ when (green flag) clicked ]` is equivalent to `main`.

```
Scratch:                                C:

                                        #include <stdio.h>

[ when (green flag) clicked ]           int main(void)
|                                       {
|   [ say [hello, world] ]                  printf("hello, world\n");
L                                       }
```

- Below, `while (true)` is the same as `[ forever ]` in Scratch. `true` is the boolean expression that `while` checks, and since `true` will always be `true` the loop will run forever.

```
Scratch:                                C:

[ forever ]                             while (true)
|                                       {
|   [ say [hello, world] ]                  printf("hello, world\n");
```

```
L                                   }
```

- Below, `for (int i = 0; i < 10; i++)` is equivalent to `[ repeat [10] ]` in Scratch.

```
Scratch:                            C:

[ repeat [10] ]                     for (int i = 0; i < 10; i++)
|                                   {
|   [ say [hello, world] ]              printf("hello, world!\n");
L                                   }
```

- Variables store something, and in C we can say `int counter = 0` instead of `[ set [counter] to 0 ]`. `int` means we want to store an integer, or number.

```
Scratch:                            C:

[ set [counter] to 0 ]              int counter = 0;
[ forever ]                         while (true)
|                                   {
|   [ say (counter) ]                   printf("%d\n",  counter); ❶
|   [ change [counter] by [1] ]         counter++;
L                                   }
```

- Line 4, beginning with `printf` and containing a `%d` is simply syntax for printing some decimal number, which we say is whatever is in `counter`.
- Boolean expressions translate similarly, with more parentheses and symbols.

```
Scratch:                            C:

< (x) < (y) >                       (x < y)
< < (x) < (y) > and < (y) < (z) > > ((x < y) && (y < z))
```

- The following chunks of code both say the relation between `x` and `y`:

```
Scratch:                                C:

[ if < (x) < (y) > ]                    if (x < y)
|                                       {
|   [ say [x is less than y] ]             printf("x is less than y
\n");                                   \n");
|                                       }
```

```
[ else             ]                    else if (x > y)
|   [ if < (x) > (y) > ]                {
|   |   [ say [x is greater than y] ]        printf("x is greater than y
\n");
|   |                                   }
|   [ else             ]                else
|   |   [ say [x is equal to y] ]       {
|   L                                       printf("x is equal to y\n");
L                                       }
```

Writing Code

- Most computers don't have a compiler preinstalled, though they can be downloaded like any other software. But instead of supporting hundreds of various computer configurations, we give you a standard Linux environment called the **CS50 Appliance**[7].
- A **hypervisor** will run the CS50 Appliance on your computer regardless of whether you have a Mac or PC, so all of us can have the illusion of running the same operating system on our computer.
- Within the Appliance, we will use a program called `gedit` that is a simple text editor. Note that the Appliance has a menu with applications, making it a full-fledged computer within your computer, so to speak.
- Let's open gedit and save a blank file as `hello.c`. We'll type in the code for it, but that only gives us the source code. To get the object code by compiling it, we run a program called `make` by typing in `make hello`, that tells `make` to look for a file called `hello.c`. `make` is smart enough to find a compiler in the Appliance and output the zeroes and ones in a file called `hello`.
- We can then run our program like so:

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

- In the first few weeks, we'll use the **terminal**, the black-and-white window in the bottom half of gedit, to focus on the underlying ideas without graphics or windows to distract us.
- If we remove the `\n` as follows,

[7] http://manual.cs50.net/appliance/2014/

```
#include <stdio.h>

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

- the terminal will look like:

```
jharvard@appliance (~): ./hello
hello, worldjharvard@appliance (~):
```

- because `\n` is telling `printf` to print a new line after `hello, world`.
- In the documentation, we encourage you to use Dropbox or an equivalent service to back up your programs automatically.
- With `hello-1.c`, we introduce a variable to store a name:

```
#include <stdio.h>

int main(void)
{
    string s = "David";
    printf("hello, %s\n", s);
}
```

- `%s` is a placeholder for a `string`, which will be replaced by `printf` when it runs.
- In a command-line interface like the terminal, we need to move into the folder to where our program is saved in order to compile and run it.
  - `cd` is a command we can run, which stands for `change directory`. We type `cd Dropbox` to change our folder, and by typing `ls`, which stands for list, we can see files in our folder.

```
jharvard@appliance (~): cd Dropbox
jharvard@appliance (~/Dropbox): ls
hello-1.c  src1m
jharvard@appliance (~/Dropbox):
```

- But when we tried to `make hello-1`, we get many errors. Let's look at the first:

```
jharvard@appliance (~/Dropbox): make hello-1
```

```
clang -ggdb3 -O0 -std=c99 -Wall -Werror    hello-1.c  -lcs50 -lm -o
 hello-1
hello-1.c:5:5: error: use of undeclared identifier 'string'; did you mean
 'stdin'?
    string s = "David";
    ^~~~~
    stdin
...
```

- The compiler tells us the error is in line 5, column 5. This is because `string` doesn't exist in C. CS50 created it as training wheels for the first few weeks in `cs50.h`, which we add as follows in line 1:

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

int main(void)
{
    string s = "David";
    printf("hello, %s\n");
}
```

- Functions previously written, like `printf`, are in files called `stdio.h` and `stdio.c`, that we can include in our programs so we can reuse those functions. `stdio.h` is the header file and `stdio.c` is the actual C source code, but if we link just `stdio.h` the compiler will know to include `stdio.c`.
- Now we get another error, in line 7:

```
jharvard@appliance (~/Dropbox): make hello-1
clang -ggdb3 -O0 -std=c99 -Wall -Werror    hello-1.c  -lcs50 -lm -o
 hello-1
hello-1.c:7:21: error: more '%' conversions than data arguments [-Werror,
 -Wformat]
    printf("hello, %s\n");
                    ~^
...
```

- Well, to fix this, we learn that the stuff inside the parentheses are **arguments**, or what we pass in to functions. Earlier, with the Scratch example, we specified that `%d` referred to the variable `counter`. In this case, the string we want to print is `s`, so we add `s` to the arguments in line 7.

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

int main(void)
{
    string s = "David";
    printf("hello, %s\n", s); ❶
}
```

- Now `hello-1.c` compiles and runs successfully, but let's make things more interesting. The CS50 Library, in `cs50.h`, contains functions like `GetInt` and `GetString`, which gets input from the user.
- Let's now write `hello-2.c` that gets a string as the name:

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

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

- Note that we have parentheses after `GetString`, signifying that it is a function. We also end every statement with a semicolon.
- `string s` means, create a new variable to hold a string, and call it `s`.
- Now we compile and run `hello-2`, and the blinking cursor waits for input and prints the name as we wanted. But we can extend the program further with another puzzle piece:

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

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

- And remember to run `make hello-2` every time the program changes, because while the source code has been updated we need a compiler to update the object code. Now we get:

```
jharvard@appliance (~/Dropbox): ./hello-2
Your name please: Daven
hello, Daven
jharvard@appliance (~/Dropbox):
```

- Eventually, we'll use loops and conditions to do even more interesting things.
- We end on `thadgavin.c`[8], code written to look pretty but not be readable by humans.

[8] http://cdn.cs50.net/2014/fall/lectures/1/m/src1m/thadgavin.c