Command Line Arguments

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

string get_name();

int main(int argc, string argv[])
{
  string name;
  if(argc > 1){
    name = argv[1];
  } else {
    name = get_name();
  }

  printf("Welcome, %s!\n", name);
}

string get_name(){
  return get_string("Hello. What is your name?\n");
}
```

Compilation commands

- `clang -o hello ./hello.c` will compile a binary of `hello.c`
- `clang -o hello -lcs50 ./hello.c` will compile a binary of `hello.c` if it includes `#include <cs50.h>` by telling the compiler to link with the CS50 library.

Preprocessing

If a line starts with a `#`, it makes it a preprocessor directive.

For example, in `#include <cs50.h>`, the preprocessor will look for `cs50.h` which contains all the prototypes of the functions in `cs50.c`, and will copy and paste this prototypes. This allows you to use `get_string` in your program,

```c
#include <stdio.h>

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

to

```c
int printf(string format, ...);

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

Compiling

Compiler compiles into assembly.

```
    .section    __TEXT,__text,regular,pure_instructions
    .build_version macos, 14, 0 sdk_version 14, 5
    .globl  _main                           ## -- Begin function main
    .p2align    4, 0x90
_main:                                  ## @main
    .cfi_startproc
## %bb.0:
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register %rbp
    leaq    L_.str(%rip), %rdi
    movb    $0, %al
```

```
    callq   _printf
    xorl    %eax, %eax
    popq    %rbp
    retq
    .cfi_endproc
                                        ## -- End function
    .section    __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz  "Hello, world\n"

.subsections_via_symbols
```

Assembling

Assembling takes assembly code into binary:

```
0000000 facf feed 0007 0100 0003 0000 0001 0000
0000010 0004 0000 0208 0000 2000 0000 0000 0000
0000020 0019 0000 0188 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0000 0000 0000 0000 0000
0000040 0088 0000 0000 0000 0228 0000 0000 0000
0000050 0088 0000 0000 0000 0007 0000 0007 0000
0000060 0004 0000 0000 0000 5f5f 6574 7478 0000
0000070 0000 0000 0000 0000 5f5f 4554 5458 0000
0000080 0000 0000 0000 0000 0000 0000 0000 0000
0000090 0016 0000 0000 0000 0228 0000 0004 0000
00000a0 02b0 0000 0002 0000 0400 8000 0000 0000
00000b0 0000 0000 0000 0000 5f5f 7363 7274 6e69
00000c0 0067 0000 0000 0000 5f5f 4554 5458 0000
00000d0 0000 0000 0000 0000 0016 0000 0000 0000
00000e0 000e 0000 0000 0000 023e 0000 0000 0000
00000f0 0000 0000 0000 0000 0002 0000 0000 0000
0000100 0000 0000 0000 0000 5f5f 6f63 706d 6361
0000110 5f74 6e75 6977 646e 5f5f 444c 0000 0000
0000120 0000 0000 0000 0000 0028 0000 0000 0000
0000130 0020 0000 0000 0000 0250 0000 0003 0000
0000140 02c0 0000 0001 0000 0000 0200 0000 0000
0000150 0000 0000 0000 0000 5f5f 6865 665f 6172
0000160 656d 0000 0000 0000 5f5f 4554 5458 0000
0000170 0000 0000 0000 0000 0048 0000 0000 0000
0000180 0040 0000 0000 0000 0270 0000 0003 0000
0000190 0000 0000 0000 0000 000b 6800 0000 0000
00001a0 0000 0000 0000 0000 0032 0000 0018 0000
00001b0 0001 0000 0000 000e 0500 000e 0000 0000
00001c0 0002 0000 0018 0000 02c8 0000 0002 0000
```

```
00001d0 02e8 0000 0010 0000 000b 0000 0050 0000
00001e0 0000 0000 0000 0000 0000 0000 0001 0000
00001f0 0001 0000 0001 0000 0000 0000 0000 0000
0000200 0000 0000 0000 0000 0000 0000 0000 0000
*
0000220 0000 0000 0000 0000 4855 e589 8d48 0b3d
0000230 0000 b000 e800 0000 0000 c031 c35d 6548
0000240 6c6c 2c6f 7720 726f 646c 000a 0000 0000
0000250 0000 0000 0000 0000 0016 0000 0000 0100
0000260 0000 0000 0000 0000 0000 0000 0000 0000
0000270 0014 0000 0000 0000 7a01 0052 7801 0110
0000280 0c10 0807 0190 0000 0024 0000 001c 0000
0000290 ff98 ffff ffff ffff 0016 0000 0000 0000
00002a0 4100 100e 0286 0d43 0006 0000 0000 0000
00002b0 000e 0000 0001 2d00 0007 0000 0002 1500
00002c0 0000 0000 0001 0600 0001 0000 010f 0000
00002d0 0000 0000 0000 0000 0007 0000 0001 0000
00002e0 0000 0000 0000 0000 5f00 616d 6e69 5f00
00002f0 7270 6e69 6674 0000
00002f8
```

Linking

Finally, the linker looks for the `#include`d files at the top (i.e. `#include <stdio.h>`) to import their c code.

For example, the linker will put the `printf` function underneath the main function to allow the main function to access it.

```
0000000 facf feed 0007 0100 0003 0000 0002 0000
0000010 0010 0000 0410 0000 0085 0020 0000 0000
0000020 0019 0000 0048 0000 5f5f 4150 4547 455a
0000030 4f52 0000 0000 0000 0000 0000 0000 0000
0000040 0000 0000 0001 0000 0000 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0019 0000 0188 0000
0000070 5f5f 4554 5458 0000 0000 0000 0000 0000
0000080 0000 0000 0001 0000 1000 0000 0000 0000
0000090 0000 0000 0000 0000 1000 0000 0000 0000
00000a0 0005 0000 0005 0000 0004 0000 0000 0000
00000b0 5f5f 6574 7478 0000 0000 0000 0000 0000
00000c0 5f5f 4554 5458 0000 0000 0000 0000 0000
00000d0 0f70 0000 0001 0000 0016 0000 0000 0000
00000e0 0f70 0000 0004 0000 0000 0000 0000 0000
00000f0 0400 8000 0000 0000 0000 0000 0000 0000
0000100 5f5f 7473 6275 0073 0000 0000 0000 0000
```

```
0000110 5f5f 4554 5458 0000 0000 0000 0000 0000
0000120 0f86 0000 0001 0000 0006 0000 0000 0000
0000130 0f86 0000 0000 0000 0000 0000 0000 0000
0000140 0408 8000 0000 0000 0006 0000 0000 0000
0000150 5f5f 7363 7274 6e69 0067 0000 0000 0000
0000160 5f5f 4554 5458 0000 0000 0000 0000 0000
0000170 0f8c 0000 0001 0000 000e 0000 0000 0000
0000180 0f8c 0000 0000 0000 0000 0000 0000 0000
0000190 0002 0000 0000 0000 0000 0000 0000 0000
00001a0 5f5f 6e75 6977 646e 695f 666e 006f 0000
00001b0 5f5f 4554 5458 0000 0000 0000 0000 0000
00001c0 0f9c 0000 0001 0000 0058 0000 0000 0000
00001d0 0f9c 0000 0002 0000 0000 0000 0000 0000
00001e0 0000 0000 0000 0000 0000 0000 0000 0000
00001f0 0019 0000 0098 0000 5f5f 4144 4154 435f
0000200 4e4f 5453 0000 0000 1000 0000 0001 0000
0000210 1000 0000 0000 0000 1000 0000 0000 0000
0000220 1000 0000 0000 0000 0003 0000 0003 0000
0000230 0001 0000 0010 0000 5f5f 6f67 0074 0000
0000240 0000 0000 0000 0000 5f5f 4144 4154 435f
0000250 4e4f 5453 0000 0000 1000 0000 0001 0000
0000260 0008 0000 0000 0000 1000 0000 0003 0000
0000270 0000 0000 0000 0000 0006 0000 0001 0000
0000280 0000 0000 0000 0000 0019 0000 0048 0000
0000290 5f5f 494c 4b4e 4445 5449 0000 0000 0000
00002a0 2000 0000 0001 0000 1000 0000 0000 0000
00002b0 2000 0000 0000 0000 0100 0000 0000 0000
00002c0 0001 0000 0001 0000 0000 0000 0000 0000
00002d0 0034 8000 0010 0000 2000 0000 0068 0000
00002e0 0033 8000 0010 0000 2068 0000 0030 0000
00002f0 0002 0000 0018 0000 20a0 0000 0003 0000
0000300 20d8 0000 0028 0000 000b 0000 0050 0000
0000310 0000 0000 0000 0000 0000 0000 0002 0000
0000320 0002 0000 0001 0000 0000 0000 0000 0000
0000330 0000 0000 0000 0000 0000 0000 0000 0000
0000340 20d0 0000 0002 0000 0000 0000 0000 0000
0000350 0000 0000 0000 0000 000e 0000 0020 0000
0000360 000c 0000 752f 7273 6c2f 6269 642f 6c79
0000370 0064 0000 0000 0000 001b 0000 0018 0000
0000380 7b2f a65c b333 0630 9eb5 90f3 44c5 51b6
0000390 0032 0000 0020 0000 0001 0000 0000 000e
00003a0 0500 000e 0001 0000 0003 0000 0c00 041d
00003b0 002a 0000 0010 0000 0000 0000 0000 0000
00003c0 0028 8000 0018 0000 0f70 0000 0000 0000
00003d0 0000 0000 0000 0000 000c 0000 0038 0000
00003e0 0018 0000 0002 0000 7802 0541 0000 0001
```

```
00003f0 752f 7273 6c2f 6269 6c2f 6269 7953 7473
0000400 6d65 422e 642e 6c79 6269 0000 0000 0000
0000410 0026 0000 0010 0000 2098 0000 0008 0000
0000420 0029 0000 0010 0000 20a0 0000 0000 0000
0000430 0000 0000 0000 0000 0000 0000 0000 0000
*
0000f70 4855 e589 8d48 113d 0000 b000 e800 0004
0000f80 0000 c031 c35d 25ff 0074 0000 6548 6c6c
0000f90 2c6f 7720 726f 646c 000a 0000 0001 0000
0000fa0 001c 0000 0000 0000 001c 0000 0000 0000
0000fb0 001c 0000 0002 0000 0f70 0000 0040 0000
0000fc0 0040 0000 0f86 0000 0000 0000 0040 0000
0000fd0 0000 0000 0000 0000 0000 0000 0003 0000
0000fe0 000c 0001 0010 0001 0000 0000 0000 0100
0000ff0 0000 0000 0000 0000 0000 0000 0000 0000
0001000 0000 0000 0000 8000 0000 0000 0000 0000
0001010 0000 0000 0000 0000 0000 0000 0000 0000
*
0002000 0000 0000 0020 0000 0050 0000 0058 0000
0002010 0001 0000 0001 0000 0000 0000 0000 0000
0002020 0004 0000 0000 0000 0000 0000 0018 0000
0002030 0000 0000 0000 0000 0018 0000 1000 0006
0002040 1000 0000 0000 0000 0000 0000 0001 0000
0002050 0201 0000 0000 0000 5f00 7270 6e69 6674
0002060 0000 0000 0000 0000 0100 005f 0012 0000
0002070 0200 0000 0300 f000 001e 0200 6d5f 5f68
0002080 7865 6365 7475 5f65 6568 6461 7265 0900
0002090 616d 6e69 0d00 0000 1ef0 0000 0000 0000
00020a0 0002 0000 010f 0010 0000 0000 0001 0000
00020b0 0016 0000 010f 0000 0f70 0000 0001 0000
00020c0 001c 0000 0001 0100 0000 0000 0000 0000
00020d0 0002 0000 0002 0000 0020 5f5f 686d 655f
00020e0 6578 7563 6574 685f 6165 6564 0072 6d5f
00020f0 6961 006e 705f 6972 746e 0066 0000 0000
0002100
```

Decompiling

Decompiling is the opposite of compiling, going from a binary to source code. It is very hard to do so that means intellectual property is protected.

Cryptography

Encryption

The scrambling of plaintext data to ciphertext algorithmically generated using a key.

Decryption

The descrambling of ciphertext back to it's original plaintext using a key.

Caesar Cipher

It is done by adding a value to a letter, where the letter wraps around at z.

E.g. if the key is 5 and the message is "well", then: > Rember that the ASCII code for 'z' is 122. - w = 119, 119 + 5 = 124, wraparound = 98 = b - e = 101, 101 + 5 = 106, 106 = j - l = 108, 108 + 5 = 113, 113 = q - l = 108, 108 + 5 = 113, 113 = q

Therefore, the ciphertext is "bjqq"

Debugging

Using `printf`

See ./assets/example_one.c for source code.

```c
#include <stdio.h>

int main(){
    for(int i = 0; i <= 3; i++){
        printf("#");
    }
    printf("\n");
}
```

The expected output is `###`, however the output is actually `####`.

Debugging process

Use `printf` to see what the program is doing

```c
#include <stdio.h>

int main(){
    for(int i = 0; i <= 3; i++){
        printf("i is %i | ", i);
        printf("#");
        printf("\n");
    }
    printf("\n")
}
```

This now gives the output of:

```
i is 0 | #
i is 1 | #
i is 2 | #
i is 3 | #
```

We can now see that the loop is going too many times, and that there is four loops. To fix this, we can change the condition of the `for loop` to be `<` instead of `<=`.

Fixed coe

```c
#include <stdio.h>

int main(){
    for(int i = 0; i < 3; i++){
```

```
            printf("#");
        }
        printf("\n");
}
```

And now the output is `###`

Using a Debugger

Instead of using printf's all over the place and having to remember to remove all of them when compiling, as well as constantly recompiling, we can use a debugger. This will allow us to be able to step over function's and keep a track of variable's values. This means that we don't have to keep compiling or remeber to remove `printf`'s. To do this at cs50.dev we can use the command `debug50`

Rubber Ducky Debugging

Traditionally, you can use a rubber ducky to explain the program that you are programming to be able to pick up where mistakes have been made. You can also use cs50.ai to use the Harvard made ChatGPT.

Memory

Memory Types

Memory Type	Size in Bytes
`bool`	1
`int`	4
`long`	8
`float`	4
`double`	8
`char`	1
`string`	?

String

A string is a dynamicly lengthed type, because it is a series of `char`'s

Memory Addresses

Each byte in memory has an address to store the bytes in physical memory. For example, you could argue a chip on a RAM stick has 4 bytes. Each byte will have an address so, as a developer, you can say "I want to store `0x4F` at address `address_1`". You can then go on to say "What is the value stored at address `address_1`?" and the computer will reply with `0x4F`.

scores.c

> See ./assets/scores.c for source code

version_1

```c
void version_1(){
  int score1 = 72;
  int score2 = 73;
  int score3 = 33;

  printf("Average: %f\n", (score1 + score2 + score3) / 3.0);
}
```

The problem with version 1, is that it has a lot of repeated variable names with numbers next to them. This can easily be solved with an array

version_2

```
void version_2(){
  int scores[] = {72, 73, 33};

  printf("Average: %f\n", (scores[0] + scores[1] + scores[2]) / 3.0);
}
```

The array has now been harcoded to have values `{72, 73, 33}`. The average gets made by adding all the values at their indices and divided by `3.0`. It would be nicer though to be able to choose how many scores to add and for it to be dynamic...

version_3

```
void version_3(){
  // Get number of scores
  const int SIZE = get_int("How many scores do you want to save? ");

  // Get Scores
  int scores[SIZE];
  for(int i = 0; i < SIZE; i++){
    scores[i] = get_int("Score %i: ", i + 1);
  }

  // Calculate Average
  float average = 0;
  for(int i = 0; i < SIZE; i++){
    average += scores[i];
  }
  average /= (float) SIZE;

  printf("Average: %f\n", average);
}
```

The user first gets asked how many values they want to add in `const int SIZE` (caps for convention of const). An array is then initialised and then looped over to add the values. After this, an averaging algorithm allows for the calculation of the average dynamically, and finally the average is `printf`d to the console.

If I was to improve on this, I would remove the `// Get Scores` and `// Calculate Average` blocks of code to their own functions to seperate concerns. However, nested functions aren't possible in C.

Strings

Strings in C are actually just arrays of characters. `char string[] = "Hello, world\n";`

This is the equivelant of: `{'H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '\n'}`

Which means, that you can do the following:

```c
#include <stdio.h>

int main(){
    char string[] = "HI!";

    for(int i = 0; i < 3; i++){
        printf("%c\n", string[i]);
    }
}
```

This will output:

```
H
I
!
```

However, strings also have a '\0' NUL byte at the end to denote the end of the string. This means we can do the following:

```c
#include <stdio.h>

int main(){
    char string[] = "Hi!";

    int i = 0;
    while(string[i] != '\0'){
        printf("%c\n", string[i]);
        i++;
    }
}
```

And this will dynamically output:

```
H
i
!
```

and this is because `string` is actually `{'H', 'i', '!', '\0'}`

String Lengths