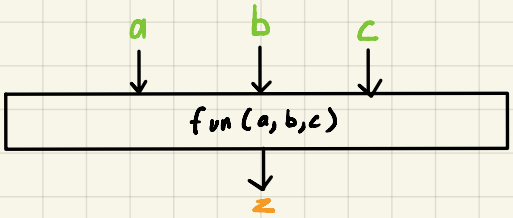
CS 50 - II

Functions

C and nearly all languages developed since allow us to write functions, sometimes also known as procedures, methods or subroutines.



A function is a blak box with a set of 0+ inputs and 1 output.

If we aren't writing the functions, ourselves, we don't need to know the underlying implementation

That's why most functions have, clear, obvious(ish) names, and are well-documented.

Functions can sometimes take no inputs or not have an output. In that case we declare the function has having a void argument list and a void return type.

Function Declaration

- return-type name(argument list); -

- The return-type is what kind of variable the function will output.
- The name is what you want to call the function.
- The argument-list is the comma-seperated set of inputs to the function. type and name.

Organisation

Functions help break up a complicated problem into more managable subparts.

Simplification

Smaller components tends to be easier to design, implement and debug.

Why use functions?

Reusability

Functions can be recycled; you only need to write them once, but can use them as often as you need.

Variable and Scope

Scope is a characteristic of a variable that defines from which functions that variable may be accessed.

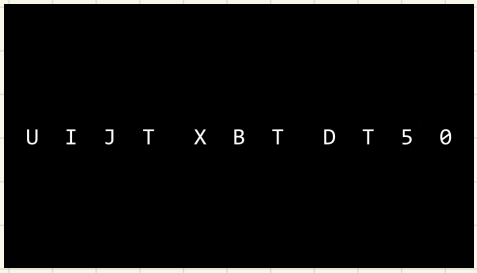
- Local variables can only be accessed from the functions in which they are created.
- Global variables can be accessed by any function in the program.

For the most part, local variable are passed by value in function calls.
It means; the callee receives a copy of the passed variable, not the variable itself. The variable in the caller remains unchanged.

Compiling

Encryption is the act of hiding plain text from prying eyes.
Decrypting, then, is the act of taking an encrypted piece of text and return it to a human-readable form.

A Compiler is a specialized computer program that converts source code into machine code that can be understood by a computer.



An encrypted piece of text may look like the above.

```c
#include <stdio.h>

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

Encrypting →

← Decrypting

```
01111111 01000101 01001100 01000110 00000010 00000001 00000001 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00111110 00000000 00000001 00000000 00000000 00000000
10110000 00000101 01000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11010000 00010011 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 01000000 00000000 00111000 00000000
00001001 00000000 01000000 00000000 00100100 00000000 00100001 00000000
00000110 00000000 00000000 00000000 00000101 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 01000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 01000000 00000000 00000000 00000000 00000000 00000000
11111000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
11111000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
00001000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000011 00000000 00000000 00000000 00000100 00000000 00000000 00000000
00111000 00000010 00000000 00000000 00000000 00000000 00000000 00000000
...
```

VSCode utilizes a compiler called clang or C Language.

If you were to type `make file_name`, it runs a command that executes clang to create an output file that you can run as a user.

```
clang -o hello hello.c

clang hello.c -lcs50
```

By default, the file is named a.out. But we can specify a name using the -o (output) argument.

When using an external library, we must use the -l argument. This will enable the compiler to access the library.

Compiling involves major steps, including the following:

Preprocessing

is where the header files, designated by a #, are effectively copied and pasted into the file. During this step, the code from each file .h is copied into the program.

Similarly, just as the code contains #include <stdio.h>, code contained within stdio.h somewhere in the computer is copied to the program.

Compiling

is where the program is converted into assembly code.

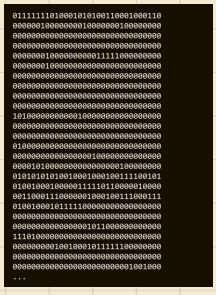
This step can be visualized as follows:

```
...
main:                                   # @main
    .cfi_startproc
# BB#0:
    pushq   %rbp
.Ltmp0:
    .cfi_def_cfa_offset 16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
.Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    xorl    %eax, %eax
    movl    %eax, %edi
    movabsq     $.L.str, %rsi
    movb    $0, %al
    callq   get_string
    movabsq     $.L.str.1, %rdi
    movq    %rax, -8(%rbp)
    movq    -8(%rbp), %rsi
    movb    $0, %al
    callq   printf
    ...
```

assembling

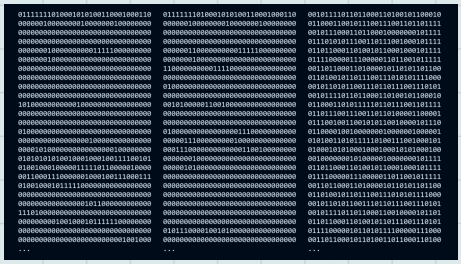
involves the compiler converting the assembly code into machine code.

This step can be visualized as follows:



Linking step

code from the included libraries are converted also into machine code and combined with the code. The final executable file is then outputted.



Debugging

Everyone will make mistakes while coding.

Printf

printf is a very useful way of debugging your code. You could add displays to have a feed back

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

Running this code, you will see numerous statements. And might realize that further code needs to be corrected.

Debugger; a software tool created by programmers to help break down bugs in code.

buggy0.c

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     for (int i = 0; i <= 3; i++)
6     {
7         printf("#\n");
8     }
9 }
```

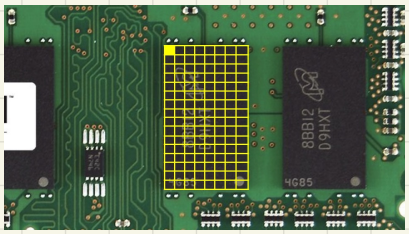
To utilize a debugger, first set a breakpoint ● by clicking to the left of a line of code, to the left of the line number.
A red dot will appear. This acts as a stop sign, asking the compiler to pause such that you can consider what's happening in this part of code.

RDD; Rubber Duck Debugging

When having challenges with a code, consider how speaking out loud to, quite literally, a rubber duck about the code problem. Speak about the code.

Arrays

Each data type requires a certain amount of **system ressources**.



bool : □ int : □□□□ float: □□□□
long : □□□□□□□□ double: □□□□□□□□
char: □ string : ?

Inside of a computer, there is a **finite amount** of memory available.
We use arrays to hold values of the same type at contiguous memory locations.

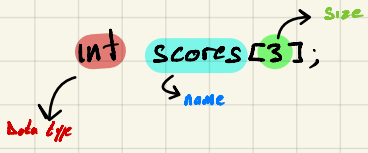
> An array is a block of contiguous space in memory ...
> which has been partitioned into small, identically-sized blocks of space called elements ...
> each of which can store a certain amount of data ...
> all of the same data type ...
> and which can be accessed directly by an index.

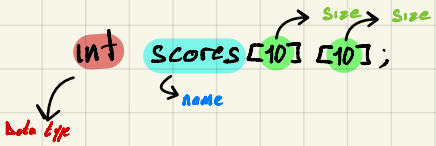
In C, the elements of an array are indexed starting from 0.

If an array consists of n elements, the first element is located at index 0. And the last element at index n-1.

→ C is very lenient. It will not prevent you from going "out of bounds" of your array.

Arrays are a way to store data back-to-back in mory such that this data is easily accessible.
`int scores[3]` is a way of telling the compiler to provide three back-to-back places in memory of size int to store three scores.





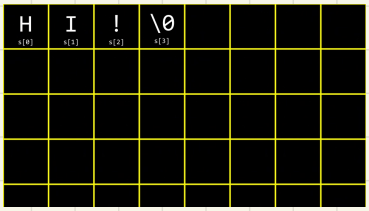
Arrays can consist of more than a single dimension. You can have as many size specifiers as you wish. You can think of it as a 10x10 grid of cells. In memory though, it's just a 100-element 1D array.

Arrays are **passed by reference**. The callee receives the actual array, not a copy of it.

String

A string is simply an array of variables of type char: an array of characters.

Strings end with a special character called a NUL character.



Command-Line Arguments

Command-Line arguments are arguments that are passed to your programs at the command line. Like -o or -l with clang.

If we want the user to provide data to our program before it starts running, we need a new form:

```
int main(void)
```

→

```
int main(int argc, string argv[])
```

number of arguments ↙ ↘ arguments.

The two special arguments enable you to know what data the user provided at the command line and how much data they provided.

argc (argument count)

This integer type variable will store the number of command-line arguments the user typed when the program was executed.

Command	argc
./greedy	1
./greedy 1024 cs50	3

argv (argument vector)

This array of strings stores, one string per element, the actual text he typed at the command line when the program was executed. The first element of argv is always found at argv[0]. The last element is always found at argv[argc-1].

Exit status

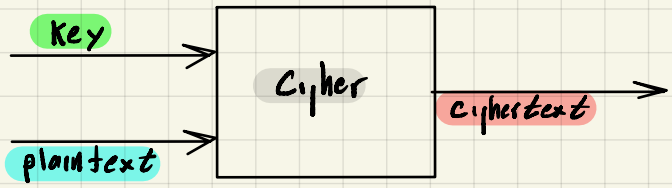
When a program ends, a special exit code is provided to the computer.

When a program exits without error, a stdout code of 0 is provided to the computer. Often, when an error occurs that results in the program ending, a stdout of 1 is provided by the computer.

Cryptography

Cryptography is the art of ciphering and deciphering a message

plaintext and a key are provided to a cipher, resulting in ciphered text.



The key is a special argument passed to the cipher along with the plaintext. The cipher uses the key to make decisions about how to implement its cipher algorithm.