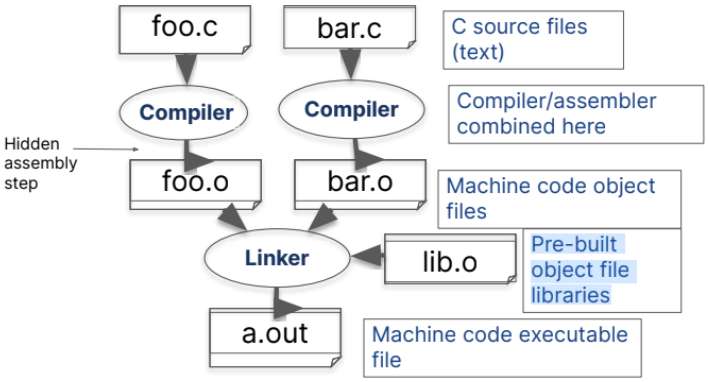
Chapter 2: C

- C (low-level, programming language) is not specialized to any particular area of application. Enabled the first OS not written in assembly. Great for writing memory management programs.
- 2.1 C Features
  - Compiled language → executables are rebuilt for each new system
  - Each variable holds garbage until initialization
  - Function parameters are pass by value
- How does the system understand our code? (Compiled v Interpreted)
  - Translation happens in 2 ways: compilation or interpretation (some lang. does both)
  - Compilation: C compilers map C program directly into architecture-specific machine code (string of 1s and 0s)
    - We will get into more detail soon.
  - Overview:



- C Syntax:
  - Main function: to accept arguments use: int main(int argc, char *argv[])
    - argc is number of strings on the command line: unix% sort myFile
    - argv is a pointer to an array containing the arguments as strings
  - Booleans: 0, NULL, and false statements evaluate to is FALSE in C
    - True and False can only be used by #include <stdbool.h> on header
  - Typed variables: must be declared (int, unsigned int, float, double[^], char, long,long long)
    - Consts and Enums: const is assigned to typed value once in declaration, value can't change; enum is a group of related integer constants
      - const float golden_ratio = 1.618;
      - enum color {RED, GREEN, BLUE};
  - Typed functions: must declare the type of data returning from function
    - Eg. int number_of_people () { return 3; }
  - Control flow: similar to other programming languages
    - tip: for (initialize; check; update)

- Key concepts are pointers, arrays, memory management
    - All the above is *unsafe*, meaning that when a program contains an error in these areas, it not only may not cause the program to crash, but will leave the program vulnerable. In this case use Rust or Go
- 2.2 Bitwise Operators
    - Bitwise operators are operators which change the bits of integer-like objects (ints, chars, etc)
        - & Bitwise AND. Useful for creating masks
        - | Bitwise OR. Useful for flipping bits on
        - ^ Bitwise XOR. Useful for flipping bits off
        - << Left shifts the bits of 1st operand < by # of bits thru 2nd operand
        - >> Right shifts the bits of 1st operand > by # of bits thru 2nd operand
        - ~ Inverts the bits
- 2.3 Pointers

  & Get the address of a variable

  * Get the value pointed to by a pointer
    - C is pass by value, which means pointers allow us to pass around objects withs having them copied
        - Declaring pointers create space for an object but does not put anything in that space (ie. it will be garbage
    - Pointers can point to anything, even other pointers, eg. *int * *a* is a pointer to an integer pointer which is called a handle.
- 2.4 Arrays
    - Represented as adjacent blocks of memory. We interact with them thru pointers:
        - int a[]; // represents arrays; points to first mem. loc of array
        - int *b; // represents arrays; points to first mem. loc of array
            - C arrays can be indexed into using [] subscripts like others. They also index into using pointer arithmetics.
                - Eg. $*(a + 2) \equiv a[2]$ // By adding 2 to a, C knows to look two memory locations into the array (i.e the third element).
        - C does this because it automatically computes the size of the objects in the arry to know how much to advance the pointer by:

```
int a[] = {1, 2, 3, 4, 5}; //ints are 4 bytes;
printf("%u", &a); // 0x2000
printf("%u", &(a+2)); // 0x2008
char *c = "abcdef"; //chars are 1 byte
printf("%u", &c); // 0x3000
printf("%u", &(a+c)); // 0x3002
```

  <u>*Note*</u>: C arrays don't know their own length or check bounds so always be passing an array to a function and its size
- 2.5 Structs

- Basic data structures in C. Like classes, they are composed of simpler data structure, but no inheritance
- 2.5.1 Struct Operators
  - → dereference a struct and get subfield
    - *Typedef* can be useful command with structs bc it names them cleanly