Homework 2:

DUE 03/14

Void pointer = a pointer to something but we don't know the size or what kind of type it is.

- Different from a null pointer: points to nothing.
- With an arraylist you have to reallocate it to grow it. Can be a hassle.
- Linkedlist avoids this, you can allocate it on the heap instead of the stack.
- Tail allows us to add to the end of the list in linear time
- Generics dont exist in C
    - Void pointers help fill this role
- For functions, pointers must be passed into the function
    - Java: list.add(item)
    - C: add(list,item)
- Validation: test file:
- Continuous strict: its gonna compile your code and its gonna test it against the all of the cases, and tell you whether if the tests pass or fail.
- Make your own tests, 20, you can use chat GPT to do this

Notes:

- GCC internals and compilation process:
    - Compilation is a translation process.
        - Each step of the compilation process translate code from one form to another
        - There are many such stages in the pipeline.
        - C source, consists of *.c and *.h files
    - In general the high level overview looks like this:
        1. Preprocessor: get everything ready for staging process
        2. Compilation frontend: representation of our code into its essentials
        3. Compilation backend: generate some type of machine code; object files that are able to be loaded into actual CPUs
        4. Linking:
           (compile time)
           —----------
           (runtime)
        5. Loading

C preprocessor:

- The first step in compilation process
- Names: cpp
    - Purose:
    - organise all included code into one file
    - It removes comments
    - expands macros: #include or #whatever those are targeted toward the compiler and macros that are expanded. It will go and search for that file and paste it into your code.
        - #ifndef - if the constant has not been defined
- File type: *.i

Front end Compilation:

- Input: preprocessed C
- Output: ast
- Program: cc
- Purpose:
    - Check for syntax errors
    - Can also check for some semantic errors
    - Generate a syntax-independent representation of your code called an: **ABSTRACT SYNTAX TREE (AST)**

Program optimization:

- After the compilation stage, code exists in tree format
- This tree will be transformed into a number of **intermediate representations (IR)**
- Trees are converted to “generic” (different from java)
- This generic is converted to “gimple”
    - Gimple is the same code as generic but spaced out rather than condensed.
- In a gimple program: you have variable names that can be changed throughout the program: while a SSA form you have for example: a_1 and a_4 which would the same variable in gimple program: a. So SSA you can have different …
- The purpose of the front/midend IR transformation is to optimize and get the program into the correct backend representation.
- After the backend we can do optimization:
    - Reduce code by removing redundant calls
    - Inline function calls to reduce indirection
    - Reorder statements to make them more efficient
    - Skip code that produces results that are not consumed.
    - Remove code from binary that is never called.
    - Unroll loops
    - Automatic tail call optimization: