Name: Fletcher O'Brien
Course: CSCI 312 Principles of Programming Languages
Assignment Deadline: March 26, 2025

Question 1

List two ways a typedef differs from macro text replacement:

1. **Macro text replacement works like a global variable, while typedef can be defined multiple times. Their scopes are different. For example, if you declare #define X 1 at the top, but type #define X 2 in main, and print at the end, you will get 1. If you declare typedef int num; and set num x = 1; but inside brackets under that you put typedef char num; and set num x = 'z' and print you will get an output of z.**
2. **Stringizing is something that can only be done with macros and not typedef, by putting a # infront of anything to make it a string.**

Question 2

Make a new directory called `Assignment2` in your ppl repo. Reimplement *The Piece of Code that Understandeth All Parsing* (Expert C Programming p. 88) in a file called `Assignment2/cdecl.c`:

- You must create, merge, and delete a branch for each function in the program
- Each branch must be named `feature/<function name>`
- Each function must be implemented using no less than two commits
- Each commit message should adhere to the following guidelines:
  - Limit the subject line to 50 characters
  - Capitalize the subject line
  - Do not end the subject line with a period
  - Use the imperative mood in the subject line (*Fix bug* not *Fixed bug* nor *Fixes bug*)
- Write your reflog to a file called `Assignment2/reflog.txt` using the following command: `Assignment2$ git reflog > reflog.txt`

Question 3

Use the procedure in the Appendix[1] for deciphering C declarations to decipher the following C declarations. First, replace the TODO marker in each "Me" bullet for each C declaration with your translation (using the procedure in the Appendix). Note that not all C declarations in this list are

[1] Bournoutian

legal. Simply replace the TODO marker with ILLEGAL for illegal declarations. Second, use gcc to compile your `cdecl.c`, and replace the TODO marker in each "cdecl.c" bullet for each C declaration with the output of your program. Third, scan each and every C declaration, and for the ones where your translation differs from the cdecl output, explain the difference. *You will not have points taken off for an incorrect translation as long as you explain the difference.*

1. `int argc;`
    - Me: **argc is int**
    - `cdecl.c`: **argc is int**
    - Explanation (if necessary): **N/A**
2. `int *p;`
    - Me: **p is pointer to int**
    - `cdecl.c`: **p is pointer to int**
    - Explanation (if necessary): **N/A**
3. `int a[];`
    - Me: **a is array of int**
    - `cdecl.c`: **a is array of int**
    - Explanation (if necessary): **N/A**
4. `int f();`
    - Me: **f is function returning int**
    - `cdecl.c`: **f is function returning int**
    - Explanation (if necessary): **N/A**
5. `char **argv;`
    - Me: **argv is pointer to pointer to char**
    - `cdecl.c`: **argv is pointer to pointer to char**
    - Explanation (if necessary): **N/A**
6. `int (*pa)[];`
    - Me: **pa is array of pointer to int**
    - `cdecl.c`: **pa is array of pointer to int**
    - Explanation (if necessary): **N/A**
7. `int (*pf)();`
    - Me: **pf is function returning pointer to int**

- `cdecl.c`: **pf is function returning pointer to int**
- Explanation (if necessary): **N/A**

8. `char *argv[];`
   - Me: **argv is array of pointer to char**
   - `cdecl.c`: **argv is array of pointer to char**
   - Explanation (if necessary): **N/A**
9. `int aa[][];`
   - Me: **aa is array of array of int**
   - `cdecl.c`: **aa is array of array of int**
   - Explanation (if necessary): **N/A**
10. `int af[]();`
    - Me: **ILLEGAL**
    - `cdecl.c`: **af is function returning array of int**
    - Explanation (if necessary): **N/A**
11. `int *fp();`
    - Me: **fp is function returning pointer to int**
    - `cdecl.c`: **fp is function returning pointer to int**
    - Explanation (if necessary): **N/A**
12. `int fa()[];`
    - Me: **ILLEGAL**
    - `cdecl.c`: **fa is array of function returning int**
    - Explanation (if necessary): **N/A**
13. `int ff()();`
    - Me: **ILLEGAL**
    - `cdecl.c`: **ff is function returning function returning int**
    - Explanation (if necessary): **N/A**
14. `int ***ppp;`
    - Me: **ppp is pointer to pointer to pointer to int**
    - `cdecl.c`: **ppp is pointer to pointer to pointer to int**
    - Explanation (if necessary): **N/A**

15. `int (**ppa)[];`
    - Me: **ppa is array of pointer to pointer to int**
    - `cdecl.c`: **ppa is array of pointer to pointer to int**
    - Explanation (if necessary): **N/A**
16. `int (**ppf)();`
    - Me: **ppf is function returning pointer to pointer to int**
    - `cdecl.c`: **ppf is function returning pointer to pointer to int**
    - Explanation (if necessary): **N/A**
17. `int *(*pap)[];`
    - Me: **pap is array of pointer to pointer to int**
    - `cdecl.c`: **pap is array of pointer to pointer to int**
    - Explanation (if necessary): **N/A (Would this not be pap is pointer to array of pointer to int? If not for rules in appendix.)**
18. `int (*paa)[][];`
    - Me: **paa is array of array of pointer to int**
    - `cdecl.c`: **paa is array of array of pointer to int**
    - Explanation (if necessary): **N/A**
19. `int (*paf)[]();`
    - Me: **ILLEGAL**
    - `cdecl.c`: **paf is function returning array of pointer to int**
    - Explanation (if necessary): **N/A (Illegal doesn't need explanation according to instructions)**
20. `int *(*pfp)();`
    - Me: **pfp is function returning pointer to pointer to int**
    - `cdecl.c`: **pfp is function returning pointer to pointer to int**
    - Explanation (if necessary): **N/A**
21. `int (*pfa)()[];`
    - Me: **ILLEGAL**
    - `cdecl.c`: **pfa is array of function returning pointer to int**
    - Explanation (if necessary): **N/A**
22. `int (*pff)()();`

- Me: **ILLEGAL**
- `cdecl.c`: **pff is function returning function returning pointer to int**
- Explanation (if necessary): **N/A**

23. `int **app[];`

- Me: **app is array of pointer to pointer to int**
- `cdecl.c`: **app is array of pointer to pointer to int**
- Explanation (if necessary): **N/A**

24. `int (*apa[])[];`

- Me: **apa is array of array of pointer to int**
- `cdecl.c`: **apa is array of array of pointer to int**
- Explanation (if necessary): **N/A**

25. `int (*apf[])();`

- Me: **ILLEGAL**
- `cdecl.c`: **apf is function returning array of pointer to int**
- Explanation (if necessary): **N/A**

26. `int *aap[][];`

- Me: **aap is array of array of pointer to int**
- `cdecl.c`: **aap is array of array of pointer to int**
- Explanation (if necessary): **N/A**

27. `int aaa[][][];`

- Me: **aaa is array of array of array of int**
- `cdecl.c`: **aaa is array of array of array of int**
- Explanation (if necessary): **N/A**

28. `int aaf[][]();`

- Me: **ILLEGAL**
- `cdecl.c`: **aaf is function returning array of array of int**
- Explanation (if necessary): **N/A**

29. `int *afp[]();`

- Me: **ILLEGAL**
- `cdecl.c`: **afp is function returning array of pointer to int**

- Explanation (if necessary): **N/A**

30. `int afa[]()[];`
    - Me: **ILLEGAL**
    - `cdecl.c`: **afa is array of function returning array of int**
    - Explanation (if necessary): **N/A**
31. `int aff[]()();`
    - Me: **ILLEGAL**
    - `cdecl.c`: **aff is function returning function returning array of int**
    - Explanation (if necessary): **N/A**
32. `int **fpp();`
    - Me: **fpp is function returning pointer to pointer to int**
    - `cdecl.c`: **fpp is function returning pointer to pointer to int**
    - Explanation (if necessary): **N/A**
33. `int (*fpa())[];`
    - Me: **ILLEGAL**
    - `cdecl.c`: **fpa is array of function returning pointer to int**
    - Explanation (if necessary): **N/A**
34. `int (*fpf())();`
    - Me: **ILLEGAL**
    - `cdecl.c`: **fpf is function returning function returning pointer to int**
    - Explanation (if necessary): **N/A**
35. `int *fap()[];`
    - Me: **ILLEGAL**
    - `cdecl.c`: **fap is array of function returning pointer to int**
    - Explanation (if necessary): **N/A**
36. `int faa()[][];`
    - Me: **ILLEGAL**
    - `cdecl.c`: **faa is array of array of function returning int**
    - Explanation (if necessary): **N/A**
37. `int faf()[]();`

- Me: **ILLEGAL**
- `cdecl.c`: **faf is function returning array of function returning int**
- Explanation (if necessary): **N/A**

38. `int *ffp()();`

    - Me: **ILLEGAL**
    - `cdecl.c`: **ffp is function returning function returning pointer to int**
    - Explanation (if necessary): **N/A**

Appendix

NOTE1. Read `*` as "pointer to" (always on the left side), `[ ]` as "array of" (always on the right side), and `( )` as "function returning" (always on the right side) as you encounter them in the declaration.

NOTE2. Illegal combinations include `[ ] ( )` (cannot have an array of functions), `( ) ( )` (cannot have a function that returns a function), and `( ) [ ]` (cannot have a function that returns an array).

Step 1. Find the identifier. This is your starting point. Then say to yourself, "<identifier> is." You've started your declaration.

Step 2. Look at the symbols on the right of the identifier. If, say, you find `( )` there, then you know that this is the declaration for a function. So you would then have "<identifier> is function returning". Or if you found a `[ ]` there, you would say "<identifier> is array of". Continue right until you run out of symbols or hit a *right* parenthesis. (If you hit a left parenthesis, that's the beginning of a `( )` symbol, even if there is stuff in between the parentheses. More on that below.)

Step 3. Look at the symbols to the left of the identifier. If it is not one of our symbols above (say, something like "int"), just say it. Otherwise, translate it into English using **NOTE1** above. Keep going left until you run out of symbols or hit a *left* parenthesis.

Now repeat steps 2 and 3 until you've formed your declaration.