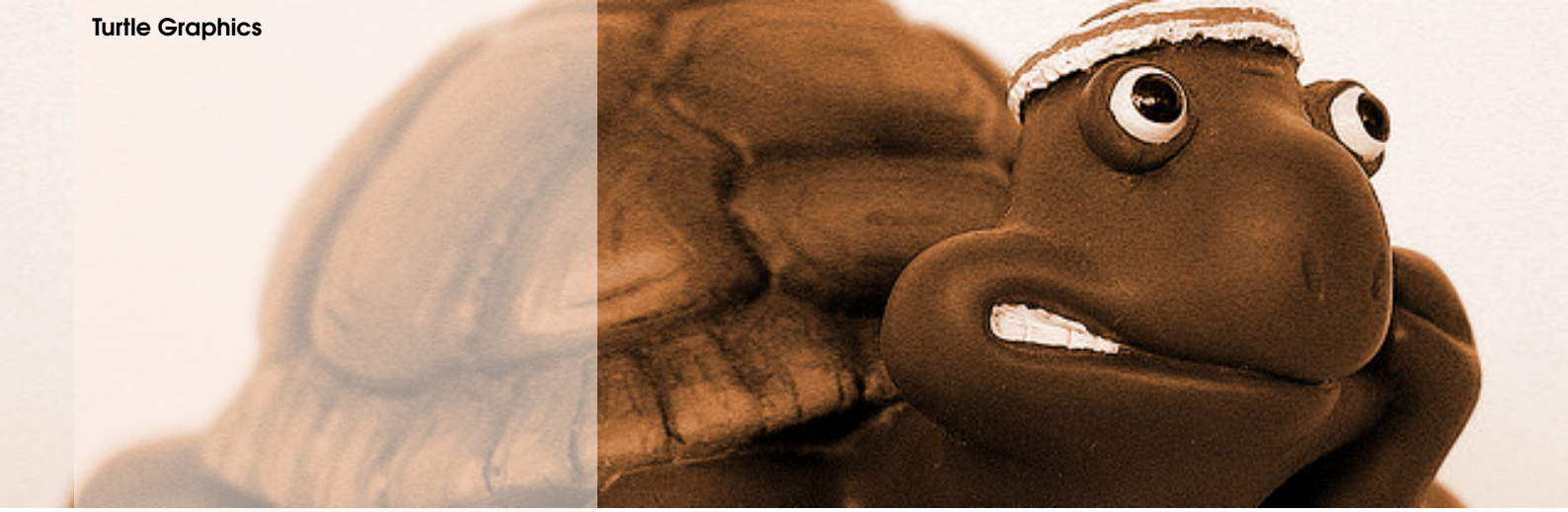


1. Week 12+ : Final Assignment

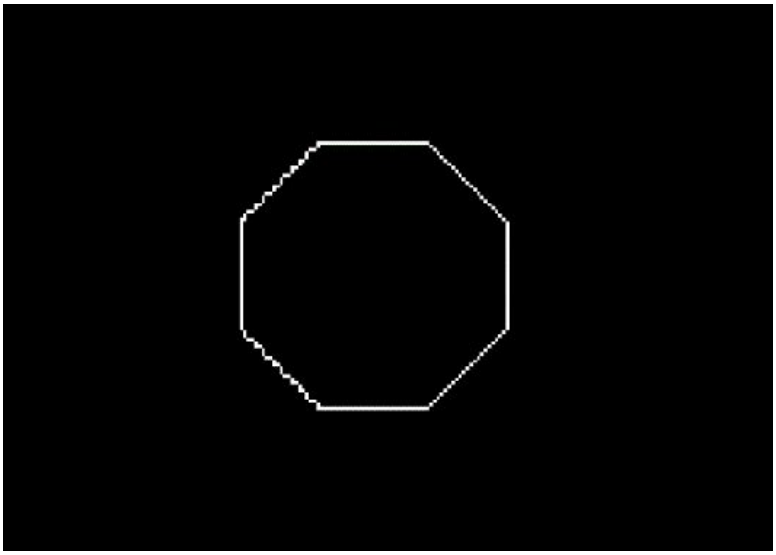
1.1 Turtle Graphics

History

- Many attempts have been made to create programming languages which are intuitive and easy to learn.
- One of the best of these was LOGO which allowed children as young as 3 to learn a computer language.
- A subset of this language involved a "turtle" which could be driven around the screen using simple instructions. The turtle, when viewed from above, was represented by a triangle.

An Example

```
{
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
 FD 30
 LT 45
}
```



Adding Loops

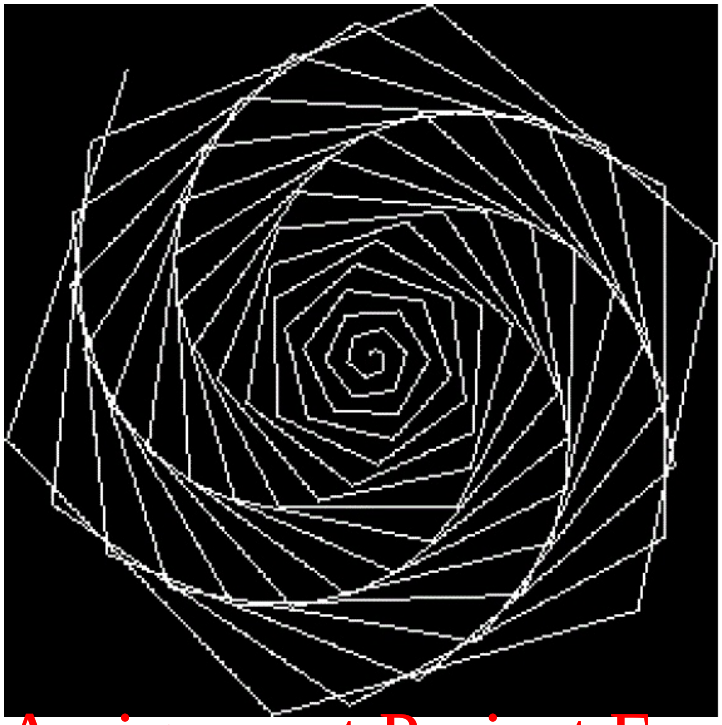
```
{
    DO A FROM 1 TO 8 {
        FD 80
        LT 45
    }
}
```


The Formal Grammar

```
<MAIN> ::= "{" <INSTRCTLST>
<INSTRCTLST> ::= <INSTRUCTION><INSTRCTLST> |
                 "}"
<INSTRUCTION> ::= <FD> |
                  <LT> |
                  <RT> |
                  <DO> |
                  <SET>
<FD> ::= "FD" <VARNUM>
<LT> ::= "LT" <VARNUM>
<RT> ::= "RT" <VARNUM>
<DO> ::= "DO" <VAR> "FROM" <VARNUM> "TO"
                  <VARNUM> "{" <INSTRCTLST>
<VAR> ::= [A−Z]
<VARNUM> ::= number | <VAR>
<SET> ::= "SET" <VAR> ":=" <POLISH>
<POLISH> ::= <OP> <POLISH> | <VARNUM> <POLISH> | ";"
<OP> ::= "+" | "−" | "∗" | "/"
```

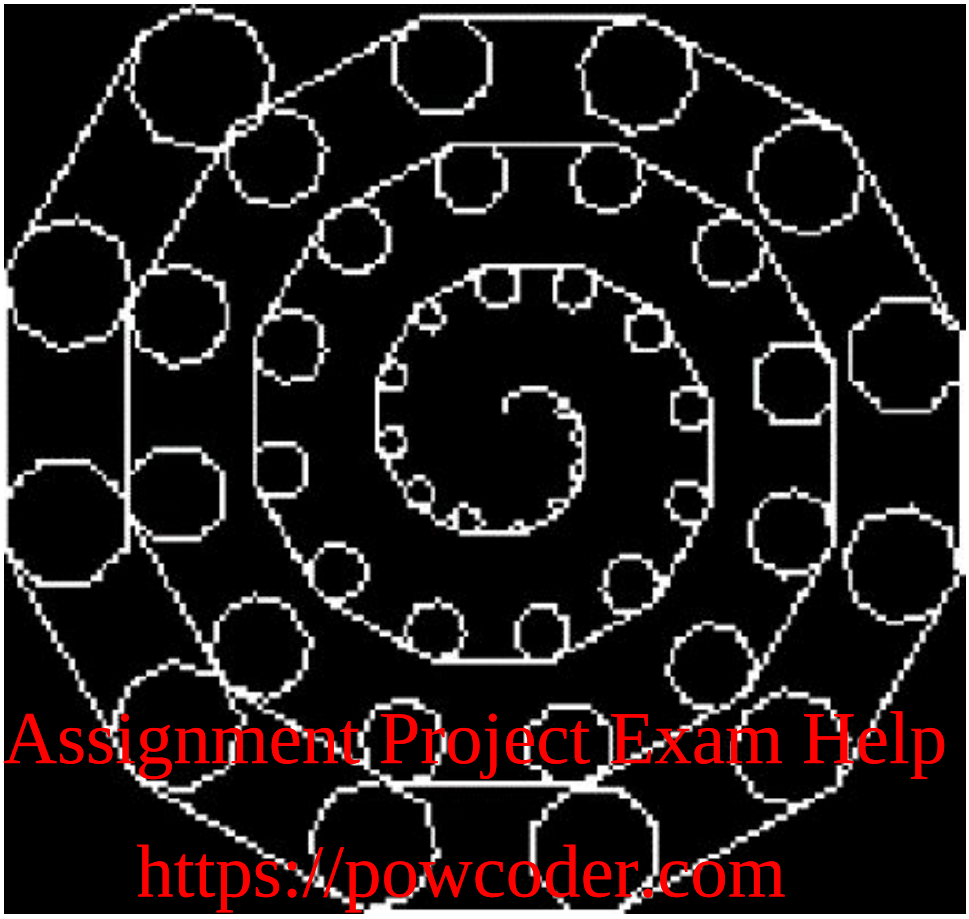
Using Variables

```
{
    DO A FROM 1 TO 100 {
        SET C := A 1.5 ∗ ;
        FD C
        RT 62
    }
}
```



Nested Loops


```
{
    DO A FROM 1 TO 50 {
        FD A
        RT 30
        DO B FROM 1 TO 8 {
            SET C := A 5 / ;
            FD C
            RT 45
        }
    }
}
```



Exercise 1.1 Implement a recursive descent parser - this will report whether or not a given turtle program follows the formal grammar or not. The input file is specified via `argv[1]` - there is **no** output if the input file is **valid**. ■

25%

Exercise 1.2 **Extend** the parser, so it becomes an interpreter. The instructions are now 'executed'. Do not write a new program for this, simply extend your existing parser. Output is via SDL. ■

25%

Exercise 1.3 Show a testing strategy on the above - you should give details of unit testing, white/black-box testing done on your code. Describe any test-harnesses used. In addition, give examples of the output of many different turtle programs. Convince me that every line of your C code has been tested. ■

25%

Exercise 1.4 Show an extension to the project in a direction of your choice. It should demonstrate your **understanding** of some aspect of programming or S/W engineering. If you extend the formal grammar make sure that you show the new, full grammar. ■

25%

Hints

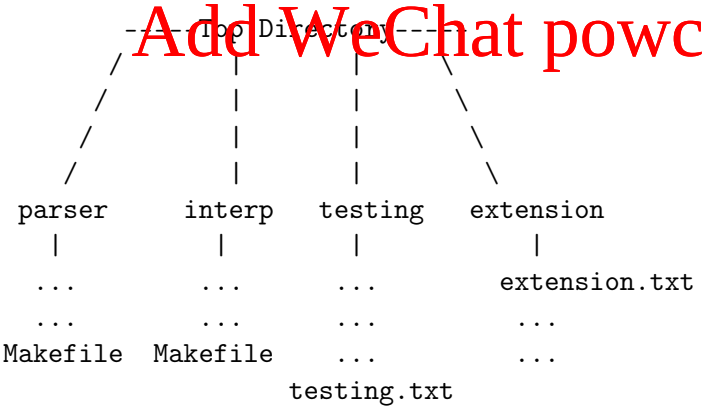
- Don't try to write the entire program in one go. Try a cut down version of the grammar first, e.g.:

```
<MAIN>        ::= "{" <INSTRCTLST>
<INSTRCTLST>  ::= <INSTRUCTION><INSTRCTLST> |
                  "}"
<INSTRUCTION> ::= <FD> | <LT> | <RT>
<FD>          ::= "FD" <VARNUM>
<LT>          ::= "LT" <VARNUM>
<RT>          ::= "RT" <VARNUM>
<VARNUM>      ::= number
```

- The language is simply a sequence of words (even the semi-colons), so use `fscanf()`.
- Some issues, such as what happens if you use an undefined variable, or if you use a variable before it is set, are not explained by the formal grammar. Use your own common-sense, and explain what you have done.
- I'm happy to sort extra help sessions in January - ask.
- Please also fill out a hard-copy of the self-assessment questionnaire on the web, so that you (and I!) have an idea of what mark you expect. Leave this in my room around deadline day. Don't forget to fill out your name (you'd be surprised at the number of people who do ....)

Submission

Please create a directory structure, so that I can easily find the different subsections. Your testing strategy will be explained in `testing.txt`, and your extension as `extension.txt`. For the parser, interpreter and extension sections, make sure there's a `Makefile`, so that I can easily build the code.



Bundle all of these up as one **single** `.zip` submission - not one for each subsection.