CS5304
Project Description

During this semester you will implement a compiler for Pascal Junior (PJ), a strict subset of Pascal. The project will emphasize:

- block structure and nested declarations,
- procedure invocation, stacks and displays, and
- formal parsing techniques and formal lexical techniques.

The compiler will use both bottom-up and top-down parse techniques to parse a PJ program. It will produce code for a hypothetical machine, HYPOMAC. HYPOMAC is a simple, stack-oriented machine that supports mechanisms for I/O and system function calls typical of a modern computer. Pertinent details of the machine language and operating system for HYPOMAC will be given later.

The project will be divided into six parts. These will be:

1. **Lexical Analyzer (1):** The lexical analyzer should correctly break any Pascal Junior program into a sequence of basic tokens.

2. **SLR Expression Grammar (1):** You will write an SLR grammar for Pascal Junior expressions. A tool will be provided that analyzes your grammar and tells you whether or not your grammar is SLR.

3. **Expression Analyzer (1):** The expression analyzer will implement an SLR parser for expressions.

4. **RD Parser (1):** You will write a recursive descent parser to parse Pascal Junior programs.

5. **Symbol Table Routines (1):** The symbol table must support nested declaration with the conventional interpretation of scope rules.

6. **Code Generation and Optimization (2):** The compiler will generate code that will run on HYPOMAC. To test the compiler, sample PJ programs and an interpreter for HYPOMAC will be provided.
Pascal Junior (PJ)

Pascal Junior is a subset of Pascal. The syntax of PJ is given in the attached Conway diagrams subject to these restrictions:

1. Comments, denoted (* ... *), behave exactly as in Pascal.
2. Blanks, end-of-line, and comments serve as separators. They are not the only separators though..... (* , - id ..... One or more separators must occur between word symbols and identifiers as in Pascal.
3. Type compatibility is enforced as in ANSI Pascal.
4. Real constants are of the form xxx.yyy where x and y each contain at least one digit (i.e. .5 is not a legal real constant). E notation is valid in PJ, e.g., 105.2E3. The mantissa for E notation must conform to rules of a real constant; the exponent can have a sign.
5. Boolean constants are true and false.
6. Type ALFA is PACKED ARRAY[1..10]OF CHAR.
7. Strings may not be broken across line boundaries.
8. Strings are delimited by the single quote character (') and cannot be empty.
9. The type TEXT is an abbreviation for FILE OF CHAR as in Pascal.
10. All files names must appear in the program header. All files names except "input" and "output" must be declared as type TEXT in the VAR section.
11. For a file F, F^ refers to the buffer variable as in Pascal.
12. Arrays are one dimensional and are declared using range designators that are non-negative integer and/or variable constants (integer), e.g. vec1 [0..50] of boolean, Arrays are referenced using valid mathematical expressions. Although a complete array aggregate can be passed an argument, no assignment can be made to it as an aggregate. Assignment must be made to individual array elements.
13. PJ includes the builtin functions CHR, EOLN, EOF, ORD, ROUND, SQR, and TRUNC. All builtin functions have exactly one argument.
14. PJ includes the builtin procedures RESET, REWRITE, PUT, GET, READ, WRITE, READLN, and WRITELN. READ, READLN, WRITE, and WRITELN must have a file variable specified, and may have at most one other expression as a parameter. For input, the expression must consist of a variable.
15. The semantics of PJ are exactly the semantics of the local Pascal compiler (when in doubt, try it).
16. PJ allows a null statement of the form BEGIN END. Be careful, PJ does not include a null block.
17. The string constant (note: this is not the char or alfa constant) is only valid as a literal in a write or writeln call.

18. Keywords are as follows:

    program, const, var, procedure, begin, end, integer, real, alfa, boolean, char, text, if,
    then, else, while, do, downto, for, to, not, or, div, mod, and, true, false, array, of.

    Each keyword will have a delimiter explicitly preceding and following it. In particular, "... 0.0else ..." is an error, not the end of an expression and the beginning of an "else".

19. WHEN IN DOUBT, TRY IT ON A PASCAL COMPILER!