Semester Project - Part 3 - CSE244 - Fall 2000

Due: Thursday, December 7, 2000, at 2:00pm

The third and final part of the semester project focuses on the actual generation of formatted output from Latex input. There are 5 tasks for this project, with the divisions given to differentiate between required work/point totals and bonuses. Note that you MUST utilize the `union` command to redefine the parsing stack in yacc/bison. This project is worth 150 points. The tasks for the project are:

1. Basic Text Processing Capabilities (35 points total), including: Section/Subsection/Table of Contents (5pts), Line Spacing/Single-Double-Triple (5pts), Page Numbering/Styles (2.5pts), Horizontal/Vertical Spacing (2.5pts), Italics/Roman Fonts (2.5pts), Paragraphs/Noindent (2.5pts), Right Justification (10pts), and Begin/End Single Blocks (5pts).

2. Advanced Text Processing capabilities (55 points total), including: Itemize Blocks (5pts), Enumerate Blocks (5pts), Center Blocks (5pts), Verbatim Blocks (5pts), Tabular Blocks (10pts), Table Blocks with Refs/Captions (5pts), and Relevant Combinations of Blocks (20pts).

3. Nested Blocks within Single Environment (15 points total): In this task, you must rewrite the grammar to support the definition of single blocks with nested blocks that have intervening text, e.g.,

   \begin{single}
   Here is some text before
   a begin end itemize block.
   \end{single}

   \begin{itemize}
   \item One item
   \item Another item
   \end{itemize}

   Here is some more text - does this work?

   \begin{single}

   To support this capability, you must carefully alter the grammar production rules. We recommend doing this last, since it will likely cause problems and the other parts can be designed, implemented, and tested, without this extension.

4. Full-Blown Verbatim (15 points total): All possible Latex commands that are embedded in a verbatim block (e.g., backslash, blocks, etc.) are ignored. The single block previously generated was output by enclosing all of the commands within a verbatim block.
5. Type/Error Checking (20 points total): Add type checking and error reporting to your Latex compiler. Implement the following type checking:

- Basic Begin/End Blocks (5pts): the type of block for the begin and end must match, e.g., both single, both verbatim, etc.
- Adv. Begin/End Blocks (5pts): Only supports the following combinations:
  
  - Single around Itemize/Enumerate/Center
  - All Combos of Itemize/Enumerate – Up to 3 levels deep
  - Center around Tabular/Verbatim

  All other combinations should result in an error.

- Tabular Specification (10pts): The number of columns in the table specification should match each table entry. For example:

  \begin{tabular}{ccc}
  One & Two & Three \\
  One & Two & Three & Four \\
  One & Two \\
  \end{tabular}

has an error in the second entry since four columns are given. Note that the third entry is allowed since there are less columns than specified, so Latex assumes a blank third column for the entry.

6. Documentation, Log, Testing (10 points total). Note that your documentation MUST be run through your text processor to generate formatted output from a Latex input file!

Note that you can have multiple versions of your “completed” project. For example, you may have a version that supports Basic and Advanced Text Processing Capabilities and Type-Checking, with a second version containing Verbatim and Nested Blocks enhancements. Points for each task are posted to allow you to maximize the points you receive for the project.

The course web site contains a number of important files, including:

- latex.l : A common latex lex file.
- latexp3.y : A yacc specification where verbatim works.
- latexp3code.y : A yacc specification with sample code generation.
- conflicts.txt : S/R and R/R Conflicts - Are all OK?
- util.c : A number of C utility routines.
- generate.c : A number of routines used to generate code.
- latex.input.txt : Sample input
- latexout.txt : Generated output for sample (with errors!)
- latextoc.txt : Generated table of contents for sample
Portions of these and other files will be discussed in class. Note that the files latex.input.txt, latex-out.txt, and latextoc.txt all have "txt" extensions for viewing on the course website. However, the code (latex.l, latexp3code.y, util.c, and generate.c) all use these files without the "txt" extension.

The third part of the project is a TEAM PROJECT and is due on Tuesday, December 7, 2000 at 2pm. You are to work in teams of 2 individuals, and teams may have a member in each of the two sections of CSE244. Also, if due to your schedule (or an odd number of people in the class), you may with prior permission work alone. If you work alone, you are still responsible for all of the project requirements! You must notify Prof. Demurjian (steveengr.uconn.edu) of your team by November 16th. For the final project report, please hand in the following:

1. The yacc/bison specification for tasks 1, 2, and 3. DO NOT HAND IN "c" files! Also include copies of any relevant "c" files. Make sure that these are well-commented, clear, and organized.

2. Documentation of your solution that includes: (1) any assumptions that you make regarding the output format and style of your document (e.g., margins, page number locations, handling underlining, etc.); (2) a log file that keeps track of all of your major design steps, implementation strategies, problems encountered (with lex/yacc) and their solutions, aspects of the project that were easy/hard, changes that were made to the grammar, etc. The key word in (2) is MAJOR design steps. NOTE AGAIN THAT YOUR DOCUMENTATION MUST BE GENERATED USING YOUR LATEX COMPILER!!!! (Thus, hand in both the documentation and the original input file!)

3. Test cases and test results for all tasks, clearly marked and organized. Note that in late November/early December, a set of representative test cases may be made available for your use.

4. A disk with all of your source files, test files, test results, output latex files, documentation, etc.

5. If you are unable to attend class by 2pm on December 7, you must submit your project before the deadline. Please do so by sending Prof. Demurjian (steveengr.uconn.edu) a zip file that contains all of the files for your project. Name the zip file using the one or two last names of the team members, e.g., smithjones.zip.

Project demonstrations will be arranged to allow you to demonstrate that your software compiles and works for all of the required test cases of the instructor. A Project Demo Schedule will be posted for sign up starting during the last week of November/first week of December. Only one member of a team is required to be present at the demo. You will be expected to re-lex, re-yacc, and re-compile your project for demonstration purposes.