CISC 458
Tutorial 2

Phase I

Revised by Ahmed Harby from Previous Versions

Thursday, January 26, 2023

Modifying the Compiler

Source Files

- For the first phase, all of the changes will occur in the following three files:
  - `stdIdentifiers` (keywords)
  - `scan.ssl` (scanner S/SL source)
  - `parser.pt` (driver program)
- All files are in the `parser` subdirectory
- **Note**: All directories are relative to the working directory.
  - Working Directory: `~/cisc458/ptsrc`

stdIdentifiers

- Contains a list of the keywords and predeclared identifiers.
- Read from the library directory `lib/pt/` when you run the compiler.
- You will need to edit this file to:
  - Add support for new keywords.
  - Remove old keywords that are not needed.
  - Change the old predeclared identifiers to the new ones.
- Be careful not to mix keywords and predeclared identifiers!
  - The `scanner` logic in `parser.pt` is sensitive to the order of items in the `stdIdentifiers` file.

duby

keywords: if, while, case, type etc

predecl identifiers: integer, char, boolean, text, true, false etc

scan.ssl

character classes are defined set of input tokens
→ adding input tokens

- Contains the S/SL program that implements the scanner.
- You will need to insert and remove character classes and token definitions to match the requirements of the new language **Quby**.
- Compile with `ssl scan.ssl` to produce `scan.def` and `scan.sst`.
  - **Note**: "`make scanner`" does this for you.
  - Do not edit `scan.def` and `scan.sst`!

parser.pt

- After compiling `scan.ssl`, definitions generated in `scan.def` need to be consistent with those in `parser.pt`.
  - Replace contents between the following paraphrased comments with the lines in your new `scan.def`.

```
{ === Pasted contents of scan.def, ... }
... content ...
{ === End of contents of scan.def }
```

do with scripts

- Whenever you change the number of identifiers in `stdIdentifiers`, you have to update the `numberStdIdentifiers` constant in `parser.pt`. → do they mean maxStdIdentifies
- Needs to be compiled using the original `ptc`.
  - **Note**: "`make scanner`" does this for you.

Running the Scanner

Building the Scanner

- Build the scanner using the "`make scanner`" command.
- Running just "`make`" will enable the parser and everything that follows, which means the test programs must be valid **Quby** programs.
  - **Not recommended!** `make` **will fail!**
  - For `make` to succeed, you will need to modify all the other phases as well!
- Make sure to run the command in the working directory rather than the `parser` subdirectory.
  - Working Directory: `~/cisc458/ptsrc`

Running the Scanner

- We only need to run the first phase of the compiler:

```
ptc -o1 -t1 -L lib/pt test.pt
```

  - `-o1` tells the compiler to run the first pass/phase only.
  - `-t1` tells the compiler to trace the execution of the scanner pass.
  - `-L` specifies the custom library path.
  - `test.pt` is the source file to compile (eg. **Quby** syntax).

- `ptc --help` gives a detailed explanation of the parameters.

Examining the Output

- We use `ssltrace` to convert the output to a human-readable format:

implemented under scripts / scantrace

```
ssltrace "ptc -o1 -t1 -L lib/pt test.pt"
    lib/pt/scan.def -i
```

- **IMPORTANT**: Make sure the library path (`-L`) is the **DIRECT** path to the library, relative to the current directory.
  - "~" will not work as the home directory.
- `lib/pt/scan.def` is where `ssltrace` should look for the names associated with the token numbers.
  - This directory should also be relative to the current directory.
- `-i` gives the trace of accepted input.

Example: tracing accepted input tokens

- test.pt

```
'kj'
```

- ssltrace "ptc -o1 -t1 -L lib/pt test.pt"
  lib/pt/scan.def -i ⇒ scantrace test.pt -i

```
[ (lQuote)
 [ (lLetter)
 [ (lLetter)
 ? (lLetter)
 [ (lLetter)
 [ (lLetter)
 ? (lLetter)
 [ (lQuote)
 [ (lNewLine)
[ (lNewLine)
[ (lEndFile)
```

Example: tracing emitted output tokens

- `test.pt`

```
'kj'
```

- `ssltrace "ptc -o1 -t1 -L lib/pt test.pt" lib/pt/scan.def -e` ⇒ scantrace test.pt -e

```
 .pStringLiteral
 % Output token text 'kj'
.pNewLine
.pEndFile
```

- Type `ssltrace --help` for more options.

Testing Suggestion

- Put and run your test inside the test subdirectory.

```
### go to the test subdirectory
cd test
### make a test program test.pt there
vim test.pt
### run your own PT scanner/screener only on it
ptc -o1 -t1 -L ../lib/pt test.pt
### run it with ssltrace of scanner/screener
ssltrace "ptc -o1 -t1 -L ../lib/pt test.pt"
    ../lib/pt/scan.def
```

compas -o1 -t1 test.pt

scantrace test.pt

- **Bonus**: If working inside the test directory, there is a script called "ptc" that automatically points the compiler to the library path.

```
./ptc -o1 -t1 test.pt
```

Quick Review

After the following slides, you should be able to answer the following questions:

1. What should you edit and what should you **not** edit?
2. What is the working directory?
3. What kind of paths do `ssltrace` and `ptc` use?
4. Where should you run `make`? Should you ever just run `make` alone?
5. What options are available for debugging the `scanner`?

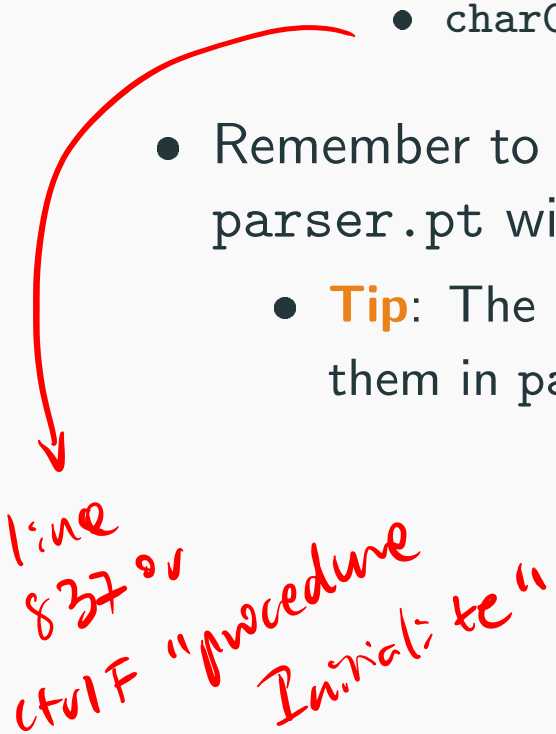
Quby- Phase I

Keywords

- From `stdIdentifiers`:
  - Add the new keywords (e.g. `fun`) and remove the old ones (e.g. `procedure`).
- From `scan.ssl`:
  - Add new keyword tokens (e.g. `pFun`) and remove unused ones (e.g. `pProcedure`).
- **Important**: The order of keywords in `stdIdentifiers` and keyword tokens in `scan.ssl` **MUST** must be the same.
- Make sure to update values of `numberStdIdentifiers`.

Character Classes

- Add new Quby input character classes such as **%** and **#**.
  - Should be added to `scan.ssl`.
  - Don't forget to update `charClassMap` in `parser.pt`.
    - `charClassMap` is in the `Initialize` procedure.
- Remember to recompile `scan.ssl` and update the values in `parser.pt` with the definitions in `scan.def`!
  - **Tip**: The "`make scanner`" command will remind you to update them in `parser.pt` if you forget.



String Literals

- **Quby** uses double instead of single quotations for string literals:
  - Old syntax: `‘content’`
  - New syntax: `"content"`
- The easiest way to do it is to change what is recognized as a quotation mark.
- Where in the source files are the quotation marks?
  - Examine the output from `ssltrace`:

```
lQuote, lLetter, ..., lLetter, lQuote
```

  - Modify the `charClassMap` to recognize `lQuote` when a double (instead of single) quotation is encountered.
  - **Hint**: A single line of code in `parser.pt`.

Syntax Tokens

- Add new Quby syntax tokens (e.g., pElsif) for the Quby keywords using,val, def, unless, elsif, break, when and module.
- Remove the old PT syntax tokens for the keywords not, until, program,const, procedure, begin and repeat, and the PT colon equals (":=").
- Update the scanning rules to handle the tokens appropriately.
  - E.g. you need to add a new rule to emit a `pPercent` token when an "%" character is encountered.
- Remember to recompile `scan.ssl` and update the values in `parser.pt` with the definitions in `scan.def`!
  - **Tip**: The "`make scanner`" command will remind you to update them in `parser.pt` if you forget.

Comments

- **Quby** uses % for comments.
  - Replaces PT { } and (* *).
- You need to modify the `Comment` rule in `scan.ssl` to reflect the new syntax
  - Make sure to emit the appropriate number of `pNewLine` tokens!

Tips & Suggestions

Suggestions

- Try to work together **as a team** to ensure that everyone understands all the changes.
  - Very important to start now, not when the smartest person in the group realizes they are overwhelmed!
- Keep track of your changes. Make frequent backups!
- Keep a copy of the original PT source beside your modified one. It can serve as a reference/guide by examining the `ssltrace` of the original compiler.

More Suggestions

- Create unit tests for each of your changes. Tests don't have to be valid programs (parser is disabled).
- Don't forget error cases!
  - Scanner should not succeed in parsing the `until` keyword.
- Create even more tests to convince yourself (and us) that your changes have not broken anything else.
- Your tests (that you submit) are required to cover all of your changes, which you should also document!
- For more specifics, tune in next tutorial, or refer to the **Project Marking Criteria** document posted by Dr. Karim on the course webpage.

Don't forget to...

- Recompile! Highly recommended to only use "`make scanner`"!
- `stdIdentifiers`, `scan.ssl`, `parser.pt` must be in sync! If you get an "Assertion 17" error then you've probably forgotten to update one of them.
- Beware not to override each other's changes when working from different terminals in the same directory.
- Backup, Backup, Backup!
- *You will be using your solution to each phase as a basis for the next one!*
- **START EARLY!**

Next Tutorial

- What to hand in
- How to hand it in
- Advising Sessions:
  - Starting Thursday 26
  - Every Monday, Wednesday, Thursday and Friday