

# Assignment 2

CSE 655

Due Wednesday, Jan 28, by 9:30 am

Consider the Core language defined in the lecture notes. Suppose we want to add to Core a `switch` statement with the following structure:

```
switch ( x with y )
  case lt  w1  gt  z1  : e1
  case lt  w2  gt  z2  : e2
  ...
  case lt  wn  gt  zn  : en
  default : en+1
endswitch ;
```

Here

- $x$  and  $y$  are some program variables
- $w_i$  and  $z_i$  are integer constants
- $e_i$  is an expression that evaluates to an integer value (non-terminal `<expr>` in the Core grammar)
- `switch`, `with`, `case`, `lt`, `gt`, `default`, and `endswitch` are new keywords

The number  $n \geq 1$  could be arbitrarily large (but  $n = 0$  is not allowed). This statement has the following semantics: if the current value of variable  $x$  is less than  $w_1$  and greater than  $z_1$ , then expression  $e_1$  is evaluated, the resulting integer value is assigned to  $y$ , the old value of  $y$  is assigned to  $x$ , and the execution of the `switch` statement completes. Otherwise, if the value of  $x$  is less than  $w_2$  and greater than  $z_2$ , expression  $e_2$  is evaluated, the resulting integer value is assigned to  $y$ , the old value of  $y$  is assigned to  $x$ , and the execution of the `switch` statement completes. Otherwise, if the value of  $x$  is  $< w_3$  and  $> z_3$ , etc. If the value of  $x$  does not trigger any of the first  $n$  cases, expression  $e_{n+1}$  is evaluated, the resulting integer value is assigned to  $y$ , the old value of  $y$  is assigned to  $x$ , and the execution of the `switch` statement completes. Note that the order in which the `cases` appear in the `switch` is important. Also, note that the `default` must be a part of the `switch`, and it must appear as the last alternative.

1. (5 pts) Extend the grammar of Core to allow such statements. Show *all* changes to the grammar. Reuse as much as possible from the existing non-terminals and productions in Core. Make sure that the new grammar is appropriate for recursive descend parsing. In addition, show a simple example of a `switch` statement and the corresponding parse tree based on your grammar.
2. (5 pts) Describe how the parse tree representation (table PT from the lecture notes) can be used to represent the *new* kinds of parse tree nodes. Keep in mind that for some terminals (e.g. `LPAREN`, etc.) we do not need to store info in the table. Describe in detail the structure of the table: the number of columns, the data stored in each column (and the data type — e.g. `int`, etc.), the meaning of this column data, etc. In addition, show PT for the `switch` example you defined in the previous question — but show only the rows for the `switch` statement and all of its sub-components (the entire parse subtree, all the way down to the leaves), not for other non-terminals such as `<prog>`.
3. (10 pts) Assuming the table representation from (2), how should we change the recursive descend parser for Core? Show the pseudo-code for all parser procedures, using the style of notation we have used in class. You need to show

- *all* changes that must be made to the procedures for the original grammar discussed in class (if nothing changes in a procedure, do not show it); this includes all calls to the scanner and all manipulations of PT
- the *complete* parsing procedures corresponding to all new non-terminals that you introduced in (1); this includes all calls to the scanner and all manipulations of PT
- do *not* show any of the code inside the scanner, but do show the calls from the parser to the scanner
- do *not* write any error-handling code (i.e., assume the input program is always valid)

The scanner API has two procedures: `getCurrent` which returns the current token but does *not* advance to the next token, and `nextToken` which moves to the next token without returning anything. At the very beginning of parsing, some other code (not yours) initializes the scanner in such a way that calling `getCurrent` for the very first time would return the first input token. There is a special `END_OF_FILE` token; if `getCurrent` returns it, it means that the end of the input program has been reached.

4. (5 pts) How should we change the recursive descend printer for Core? Show all changes to the original printer described in class: (a) the complete bodies of any changed existing procedures, and (b) the complete bodies of any new procedures. Show explicitly all accesses to PT elements.
5. (5 pts) How should we change the recursive descend executor for Core? Show all changes to the original executor described in class: (a) the complete bodies of any changed existing procedures, and (b) the complete bodies of any new procedures. The implementation should use your PT representation from (2), as created by your parser from (3). Assume that there are two helper procedures `setValue` and `getValue` that can be used to change and look up the values of program variables, respectively. You do not need to write the code for these procedures — assume that they are already defined by someone else. Procedure `getValue` takes as input a string value representing the name of a program variable, and returns an integer value representing the current value of that variable. Procedure `setValue` takes two parameters: a string value representing the name of a program variable, and an integer value representing the new value of that variable. Do not worry about uninitialized variables — assume that the input program never tries to read the value of an uninitialized variable. Make sure you explicitly show all necessary calls to these helper procedures from within your executor procedures.

- Assignments are to be done independently. General high-level discussion of assignments with others in the class is allowed, but the actual work should be your own. Assignments that show excessive similarities will be taken as evidence of cheating and dealt with accordingly.
- Assignments should be turned in **by the beginning of class** on the due day. Late assignments turned in by the beginning of the next class will be graded with 30% reduction. Assignments turned in later than that will not be accepted.
- Make the assignments readable and understandable. They are to be handed in on regular paper, legibly written or typed. If you have more than one sheet, **staple the sheets together**. If the grader has trouble reading or understanding what you have done, points will be deducted even if it can finally be determined that you have the correct answer.
- Your solutions have to be **precise and detailed**: you have to work out **all** details that are necessary to solve the problem using the approaches discussed in class. You also have to write your solutions in a way that convinces the grader that you understand all these details. Be careful, precise, and thorough.