

Mikaella Reign L. Gangoso
BSCS 3 – 2
2021-12371
DBTK

1. Simulate the expressions using your code (hand written).

Program 1:

$a = 7, d = 4, f = 1$

$m = (a/2) + (1-d) * (2\%f)$

play(m)

Simulation:

$a = 7$

$d = 4$

$f = 1$

$m = (a/2) + (1-d) * (2\%f)$

$m = (7/2) + (1-4) * (2\%1)$

$m = 3.5 + (1-4) * (2\%1)$

$m = 3.5 + -3 * (2\%1)$

$m = 3.5 + -3 * 0$

$m = 3.5 + 0$

$m = 3.5$

Program 2:

$c = 3, f = 6, g = 10$

$r = 6/3 + c * 9\%8 - f + g * 7 - 1$

play(r)

Simulation:

$c = 3$

$f = 6$

$g = 10$

$r = 6/3 + c * 9\%8 - f + g * 7 - 1$

$r = 6/3 + 3 * 9\%8 - 6 + 10 * 7 - 1$

$r = 2 + 3 * 9\%8 - 6 + 10 * 7 - 1$

$r = 2 + 27\%8 - 6 + 10 * 7 - 1$

$r = 2 + 3 - 6 + 10 * 7 - 1$

$r = 2 + 3 - 6 + 70 - 1$

$r = 5 - 6 + 70 - 1$

$r = -1 + 70 - 1$

$r = 69 - 1$

$r = 68$

Program 3:

$b = 3, c = 1, e = 8$

$g = (7+b)/(c*10) + (e-8)$

play(g)

Simulation:

$b = 3$

$c = 1$

$e = 8$

$g = (7+b)/(c*10) + (e-8)$

$g = (7+3)/(1*10) + (8-8)$

$g = 10/(1*10) + (8-8)$

$g = 10/10 + (8-8)$

$g = 10/10 + 0$

$g = 1 + 0$

$g = 1$

2. Run your compiler and display the output to validate your simulation.

Program 1:

```
Terminal
3.5
```

Program 2:

```
Terminal
68.0
```

Program 3:

```
Terminal
1.0
```

3. Submit the screenshot of your code.

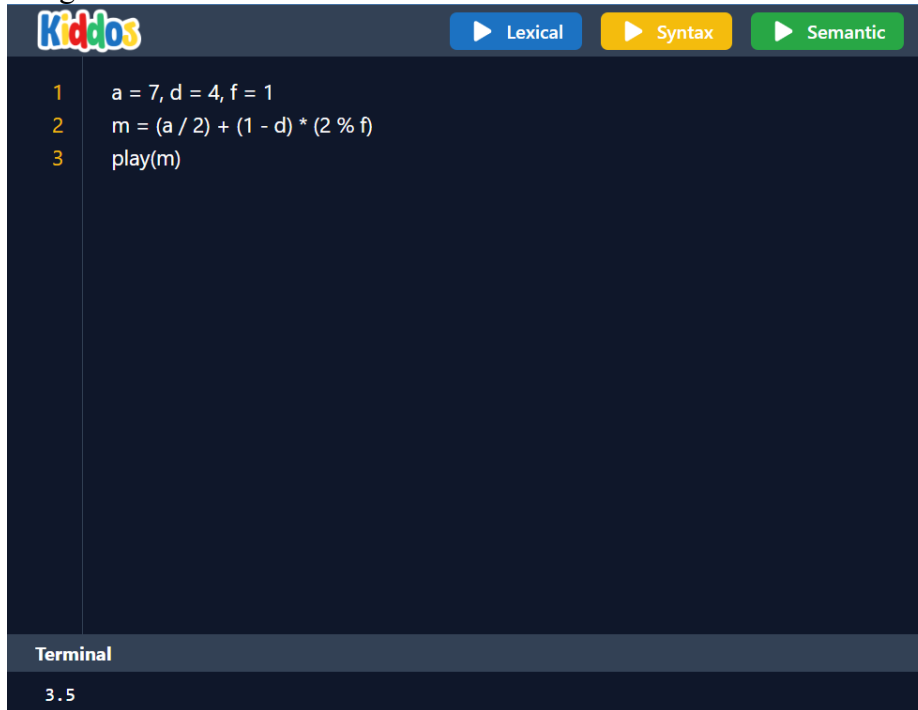
```

500
501 def validateStringUsingStackBuffer(parsing_table, grammar_ll,
502                                   table_term_list, input_string,
503                                   term_userdef, start_symbol, token_lines, lexemes):
504
505     print("Validate String <=> (input_string)\n")
506
507     # for more than one entries
508     # = in one cell of parsing table
509     if grammar_ll == False:
510         return ("Input String = (input_string) Grammar is not LL(1)")
511
512     # implementing stack buffer
513     stack = [start_symbol, '$']
514     buffer = []
515
516     # reverse input string store in buffer
517     input_string_list = [(token[0], index) for index, token in enumerate(token_lines)]
518     input_string_list.reverse()
519     buffer = [('$', None)] + input_string_list
520
521     # Initialize an empty dictionary to store token frequencies
522     token_pos = {}
523     processed_tokens = []
524     py_processed_tokens = []
525     py_lexemes = [x for x in lexemes if x != '\n']
526
527     sample = []
528
529     # global_def_pos = {}
530     # is_global = True
531     # local_def_pos = {}
532     # global_list_pos = {}
533     # global_func_pos = {}
534     # current_state = None
535
536     # Initialize a flag variable outside the loop
537     first_definition_or_statement = True
538
539     brace_stack = []
540     is_call = False
541     pos_paramtail = 0
542     pos_call = 0
543     a = 0
544
545     while True:
546         # keep track of processed tokens
547         if buffer[-1][0] != '$' and (not token_pos or token_pos[-1] != buffer[-1][1]):
548             token_pos.append(buffer[-1][1])
549             processed_tokens.append(buffer[-1][0])
550             py_processed_tokens.append(buffer[-1][0])
551
552         # sample.append(buffer[-1][0])
553
554         # end loop if all symbols matched
555         if stack == ['$'] and buffer == ['$']:
556             # Replace the placeholders with newline characters
557             return "Valid Syntax. (a) \n(processed_tokens) \n (lexemes) \n \n (py_processed_tokens) \n (py_lexemes) \n \n (sample)", py_lexemes
558         elif stack[0] not in term_userdef:
559             # take first of buffer (y) and top (x)
560             if stack[0] in dictio:
561                 x = list(dictio.keys()).index(stack[0])
562             else:
563                 expected_tokens = firsts["PROGRAM"]
564                 if 'd' in expected_tokens:
565                     expected_tokens.remove('d')
566                 if 's' in expected_tokens:
567                     expected_tokens.remove('s') # to remove 's'
568                 return "SyntaxError at Line (token_lines[len(processed_tokens)-1][1]) Invalid Token '(buffer[-1][0])' Expected: (expected_tokens)"
569             y = table_term_list.index(buffer[-1][0])
570             if parsing_table[x][y] != '':
571                 # final table entry received
572                 entry = parsing_table[x][y]
573                 lhs_rhs = entry.split("<=>")
574                 lhs_rhs[0] = lhs_rhs[0].replace('$', '').strip()
575                 entryrhs = lhs_rhs[1].split()
576                 stack = entryrhs + stack[1:]
577                 sample.append(lhs_rhs[0])
578                 sample.append(py_processed_tokens[-1])
579                 # sample.append(buffer[-1][0])
580                 # sample.append(token_tracker)
581
582             # Check for var_def_tail and const_def_tail
583             if lhs_rhs[0] in ["VAR_DEF_TAIL", "CONST_DEF_TAIL"] and buffer[-1][0] == ' ':
584                 if py_processed_tokens[-1] == ' ':
585                     # modify token
586                     py_processed_tokens[-1] = '\n'
587                     # modify lexeme
588                     py_lexemes[len(py_processed_tokens) - 1] = '\n'
589
590             # Check if a code block is formed
591             if lhs_rhs[0] in ["FUNC_DEF", "FUNC_FOR_LOOP", "FOR_LOOP", "WHILE_LOOP", "FUNC_WHILE_LOOP", "CONDIF_STMT", "FUNC_CONDIF_STMT", "LOOP_CONDIF_STMT", "FUNC_LOOP_CONDIF_STMT"]:
592                 brace_stack.append('-')
593
594             # ( -> : conversion
595             if lhs_rhs[0] in ["PARAMETER", "PARAM_TAIL", "RANGE_END", "RANGE_STEP", "INT_EXPR_TAIL", "EXPR_TAIL"] and buffer[-1][0] == ' ':
596                 pos_paramtail = len(py_processed_tokens)
597
598             if pos_paramtail != 0 and lhs_rhs[0] in ["FUNC_BODY", "LOOP_BODY", "FUNC_LOOP_BODY", "CODIF_BODY"]:
599                 py_processed_tokens[pos_paramtail] = ':'
600                 py_lexemes[pos_paramtail] = ':'
601                 pos_paramtail = 0
602
603             # Add indentation to code block contents
604             if lhs_rhs[0] in ["FUNC_BODY", "FUNC_MORE_BODY", "LOOP_BODY", "LOOP_MORE_BODY", "FUNC_LOOP_BODY", "FUNC_LOOP_MORE_BODY", "CODIF_BODY", "CODIF_MORE_BODY"]:
605                 index = len(py_processed_tokens) - 1
606                 indents = '\n' + '\t' * (len(brace_stack) if len(brace_stack) != 0 else 1)
607                 py_processed_tokens.insert(index, indents)
608                 py_lexemes.insert(index, indents)
609
610             # Remove closing brace
611             if buffer[-1][0] == ' ':
612                 py_processed_tokens[-2] = ' '
613                 py_lexemes[len(py_processed_tokens)-1] = ' '
614                 if brace_stack:
615                     brace_stack.pop()
616                 a+=1
617
618             # Remove call . from FUNC_CALL
619             if lhs_rhs[0] == "FUNC_CALL":
620                 py_processed_tokens.pop()
621                 py_lexemes.pop(len(py_processed_tokens))
622                 py_lexemes.pop(len(py_processed_tokens))
623                 pos_call = len(py_processed_tokens)
624                 is_call = True
625
626             if is_call and lhs_rhs[0] == "ARGUMENT":
627                 py_processed_tokens.pop(pos_call)
628                 pos_call = 0
629
630             # Add /n to def and statements outside code blocks
631             if len(brace_stack) == 0 and lhs_rhs[0] in ["DEFINITIONS", "STATEMENTS"]:
632                 if not first_definition_or_statement:
633                     py_processed_tokens.insert(-1, '\n')
634                     index = len(py_processed_tokens) - 2
635                     py_lexemes.insert(index, '\n')
636                 else:
637                     first_definition_or_statement = False
638
639

```

4. Submit the screenshot of your output after running your compiler.

Program 1:



The screenshot shows the Kiddos compiler interface. At the top, there is a header with the 'Kiddos' logo on the left and three buttons: 'Lexical' (blue), 'Syntax' (yellow), and 'Semantic' (green). Below the header is a dark blue editor area containing three lines of code, each preceded by a line number in yellow: '1 a = 7, d = 4, f = 1', '2 m = (a / 2) + (1 - d) * (2 % f)', and '3 play(m)'. At the bottom of the interface is a 'Terminal' section with a dark blue background, displaying the output '3.5'.

```
1 a = 7, d = 4, f = 1
2 m = (a / 2) + (1 - d) * (2 % f)
3 play(m)
```

Terminal

3.5

Program 2:



The screenshot shows the Kiddos compiler interface. At the top, there is a header with the 'Kiddos' logo on the left and three buttons: 'Lexical' (blue), 'Syntax' (yellow), and 'Semantic' (green). Below the header is a dark blue editor area containing three lines of code, each preceded by a line number in yellow: '1 c = 3, f = 6, g = 10', '2 r = 6 / 3 + c * 9 % 8 - f + g * 7 - 1', and '3 play(r)'. At the bottom of the interface is a 'Terminal' section with a dark blue background, displaying the output '68.0'.

```
1 c = 3, f = 6, g = 10
2 r = 6 / 3 + c * 9 % 8 - f + g * 7 - 1
3 play(r)
```

Terminal

68.0

Program 3:

