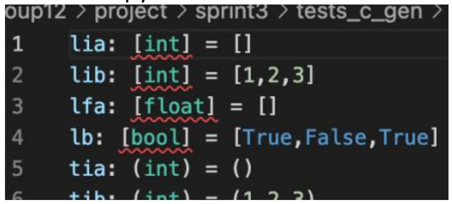
Evaluation Item	Unacceptable (0)	Acceptable (1)		Comments
Verifiability Note: if this item is "unacceptable", the rest of the assignment will not be marked.	The submission lacks a script for validating the parser, or the script provided could not be used to verify the parser.	The submission includes (a) a set of tests for verifying the parser and (b) a script to build your parser and to parse those tests. That script can be used on a lab machine to verify the parser outputs.		● A set of tests have been provided, and test_lex.py, test_yacc.py, test_lr_gen.py works as expected. ● However, the two new test files from this sprint: test_C_AST.py and test_c_gen.py do not seem to work properly. See additional notes for details. 0.5/**1**
Evaluation Item	**Unacceptable (0)**	**Marginal (1-2)**	**Proficient (3)**	**Comments**

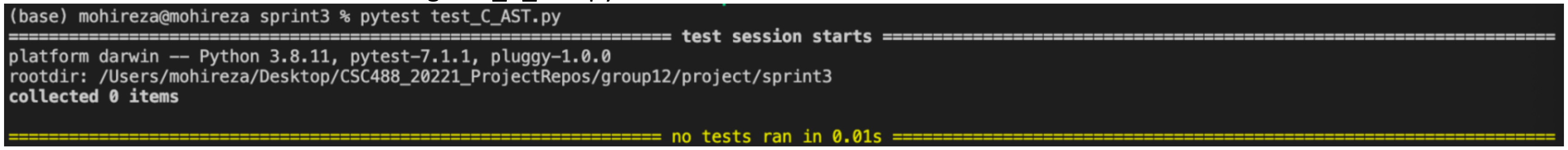
Correctness	The extension has broken the compiler's ability to generate code in many cases.	The extension has caused the compiler to fail in edge cases. Invalid code may be generated if the input is erroneous.	The compiler continues to generate legal code with the extension enabled. When it encounters an error in the input, an error is reported.	• One of the tests in test_gen.c fails and it is not clear from the documentation if this is intentional. See additional notes for details.
				• It is also not clear from the documentation which of the tests demonstrate the abilities of the type extension. However, looking at the files generated under tests_c_gen, I can see a file called "11_extended_types_received" and that seems to include type extension features.
				• The 11_extended_types_input.py doesn't seem to be valid python code:
				`oup12 > project > sprint3 > tests_c_gen >`
				`lia: [int] = []`
				`lib: [int] = [1,2,3]`
				`lfa: [float] = []`
				`lb: [bool] = [True,False,True]`
				`tia: (int) = ()`
				`tib: (int) = (1,2,3)`
				• It is not clear whether the test suite contains failing cases to check if the type checking works properly.
				
				1.5/**3**

Robustness	The extension only works on a very narrow case.	The extension fails to account for some edge cases or rules out some common cases.	The extension is capable of handling unexpected or incorrect input gracefully.	• It is not clear from the provided test suite whether or not the extension is capable of handling unexpected or incorrect input gracefully. **1.5/3**

Evaluation Item	Unacceptable (0)	Marginal (1-2)	Proficient (3)	Comments
Documentation	The extension is not clearly identified or the document does not explain its purpose.	The extension is identified clearly, but the documentation fails to clearly explain how it is implemented or why it is interesting.	The extension is identified clearly. The document explains what it does, how it is implemented, and why it is interesting.	● The documentation contains the required elements. 3/**3**

Additional Notes:

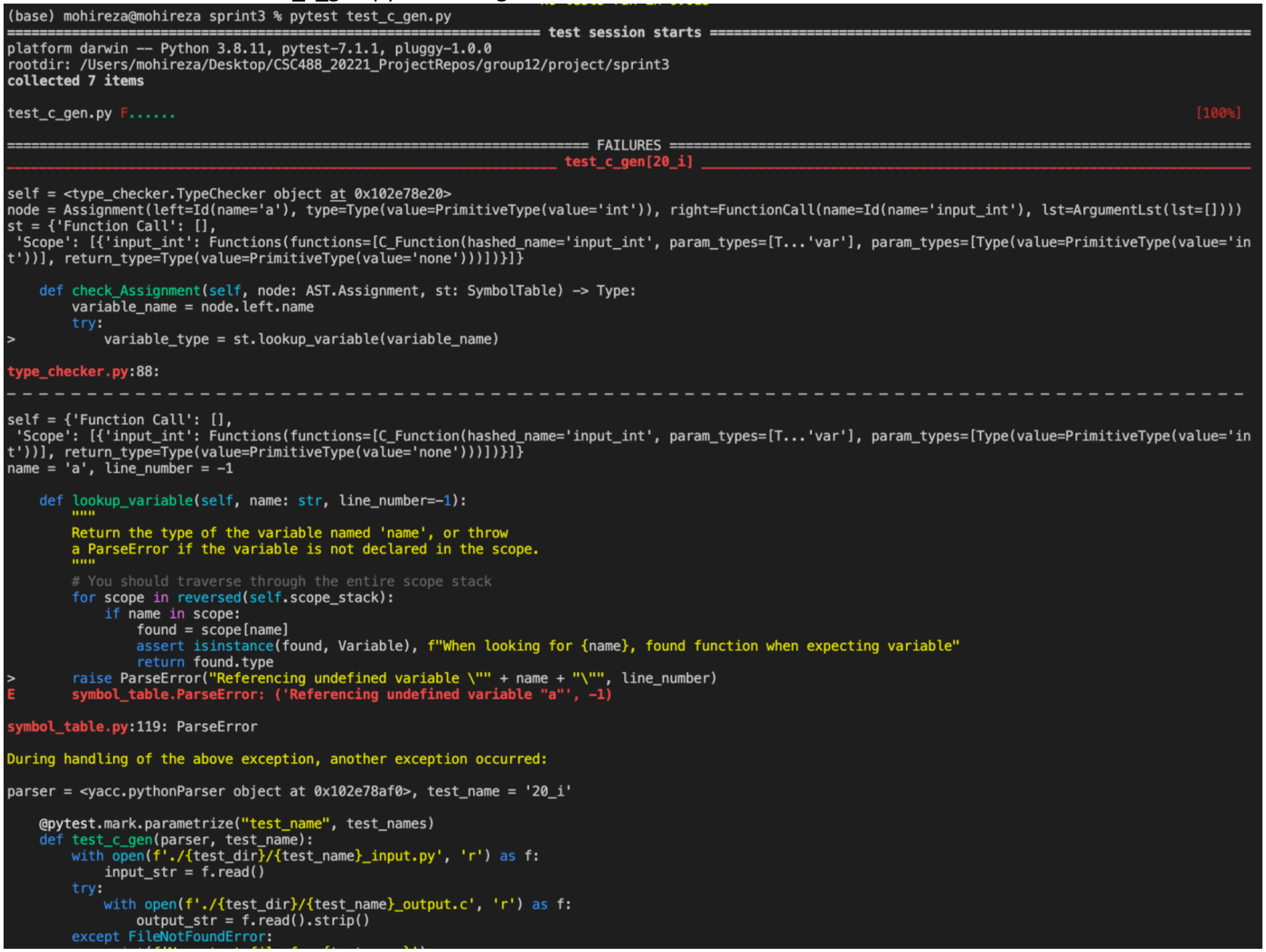
- IR for the type extension is generated in the 11_extended_types_ir_received.txt file.
- No tests seem to run when executing test_C_AST.py



```
(base) mohireza@mohireza sprint3 % pytest test_C_AST.py
============================================================ test session starts =============================================================
platform darwin -- Python 3.8.11, pytest-7.1.1, pluggy-1.0.0
rootdir: /Users/mohireza/Desktop/CSC488_20221_ProjectRepos/group12/project/sprint3
collected 0 items

============================================================ no tests ran in 0.01s ============================================================
```

- One of the test cases in test_c_gen.py fails and gives several errors:



```
(base) mohireza@mohireza sprint3 % pytest test_c_gen.py
================================================================= test session starts =================================================================
platform darwin -- Python 3.8.11, pytest-7.1.1, pluggy-1.0.0
rootdir: /Users/mohireza/Desktop/CSC488_20221_ProjectRepos/group12/project/sprint3
collected 7 items

test_c_gen.py F......                                                                                                                          [100%]

====================================================================== FAILURES =======================================================================
___________________________________________________________________ test_c_gen[20_i] ___________________________________________________________________

self = <type_checker.TypeChecker object at 0x102e78e20>
node = Assignment(left=Id(name='a'), type=Type(value=PrimitiveType(value='int')), right=FunctionCall(name=Id(name='input_int'), lst=ArgumentLst(lst=[])))
st = {'Function Call': [],
 'Scope': [{'input_int': Functions(functions=[C_Function(hashed_name='input_int', param_types=[T...'var'], param_types=[Type(value=PrimitiveType(value='in
t'))], return_type=Type(value=PrimitiveType(value='none')))])}]}

    def check_Assignment(self, node: AST.Assignment, st: SymbolTable) -> Type:
        variable_name = node.left.name
        try:
>           variable_type = st.lookup_variable(variable_name)

type_checker.py:88:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = {'Function Call': [],
 'Scope': [{'input_int': Functions(functions=[C_Function(hashed_name='input_int', param_types=[T...'var'], param_types=[Type(value=PrimitiveType(value='in
t'))], return_type=Type(value=PrimitiveType(value='none')))])}]}
name = 'a', line_number = -1

    def lookup_variable(self, name: str, line_number=-1):
        """
        Return the type of the variable named 'name', or throw
        a ParseError if the variable is not declared in the scope.
        """
        # You should traverse through the entire scope stack
        for scope in reversed(self.scope_stack):
            if name in scope:
                found = scope[name]
                assert isinstance(found, Variable), f"When looking for {name}, found function when expecting variable"
                return found.type
>       raise ParseError("Referencing undefined variable \"" + name + "\"", line_number)
E       symbol_table.ParseError: ('Referencing undefined variable "a"', -1)

symbol_table.py:119: ParseError

During handling of the above exception, another exception occurred:

parser = <yacc.pythonParser object at 0x102e78af0>, test_name = '20_i'

    @pytest.mark.parametrize("test_name", test_names)
    def test_c_gen(parser, test_name):
        with open(f'./{test_dir}/{test_name}_input.py', 'r') as f:
            input_str = f.read()
        try:
            with open(f'./{test_dir}/{test_name}_output.c', 'r') as f:
                output_str = f.read().strip()
        except FileNotFoundError:
```

- I am also seeing the presence of shift/reduce errors indicating grammar ambiguity. This needs to be fixed. **0.5 marks have been deducted for this.**
- Please check the following resources on how to deal with shift/reduce conflicts in your grammar:
    - https://www.ibm.com/docs/en/zos/2.2.0?topic=ambiguities-rules-help-remove
    - https://www.gnu.org/software/bison/manual/html_node/Shift_002fReduce.html
    - https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dh2/index.html

- *CHECK:*
    - *Is the IR created for the type extension?*
    - *Was the IR created in Sprint 2, if not, does it exist now?*
    - *Did they incorporate the changes from Sprint 2, if not, restate them here!*

Group Members:

1004545842	**CHEN**	**LITAO**	**litao.chen@mail.utoronto.ca**	**chenlita**
1003403872	**SELLATHURAI**	**JATHAVAN**	**jathavan.sellathurai@mail.utoronto.ca**	**sellat16**
1004910670	**YIN**	**YIFEI**	**yifei.yin@mail.utoronto.ca**	**yinyife2**

Score: 6**/10**