C-AST Generation With Python

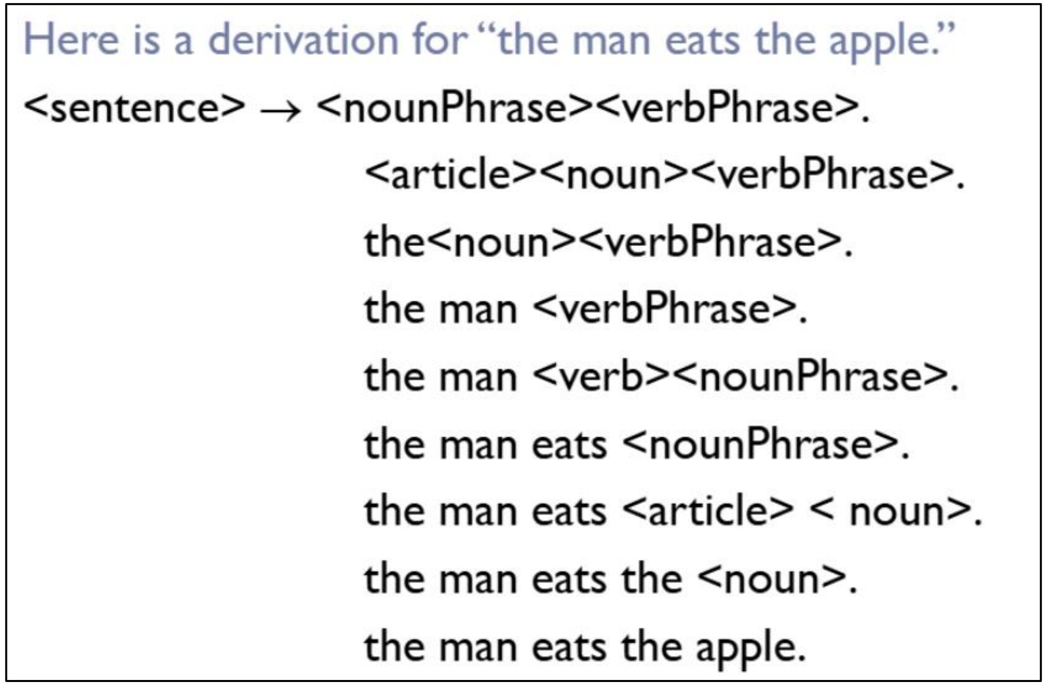
Programming Languages_SWE3006_41

Assignment 2

Due Date: April 11th, 11:59pm

Course Review

- Derivation: repeated application of rules, starting with the start symbol and ending with a sentence

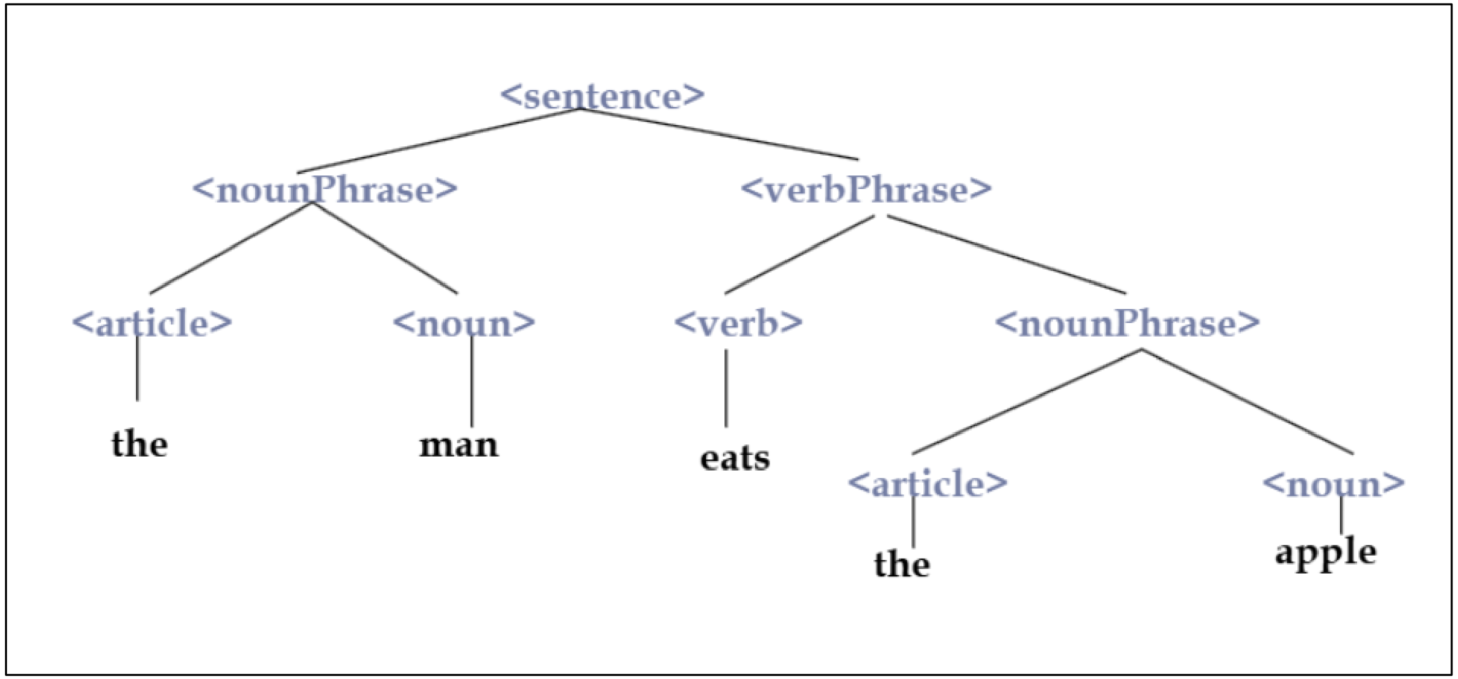


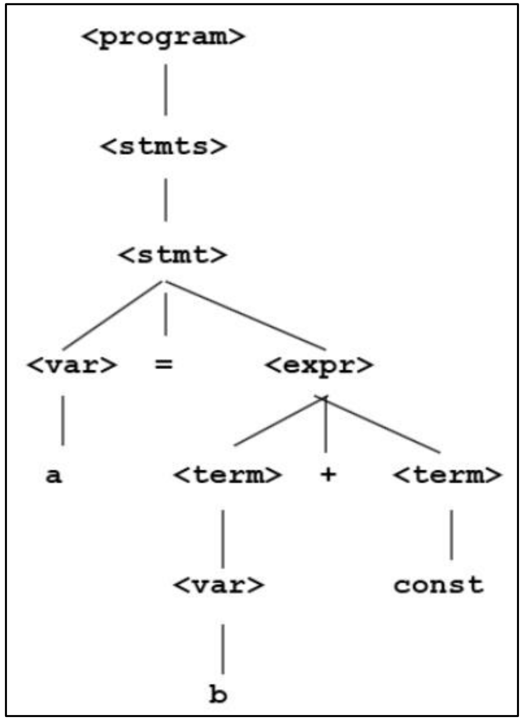
Here is a derivation for "the man eats the apple."

```
<sentence> → <nounPhrase><verbPhrase>.
             <article><noun><verbPhrase>.
             the<noun><verbPhrase>.
             the man <verbPhrase>.
             the man <verb><nounPhrase>.
             the man eats <nounPhrase>.
             the man eats <article> < noun>.
             the man eats the <noun>.
             the man eats the apple.
```

Course Review

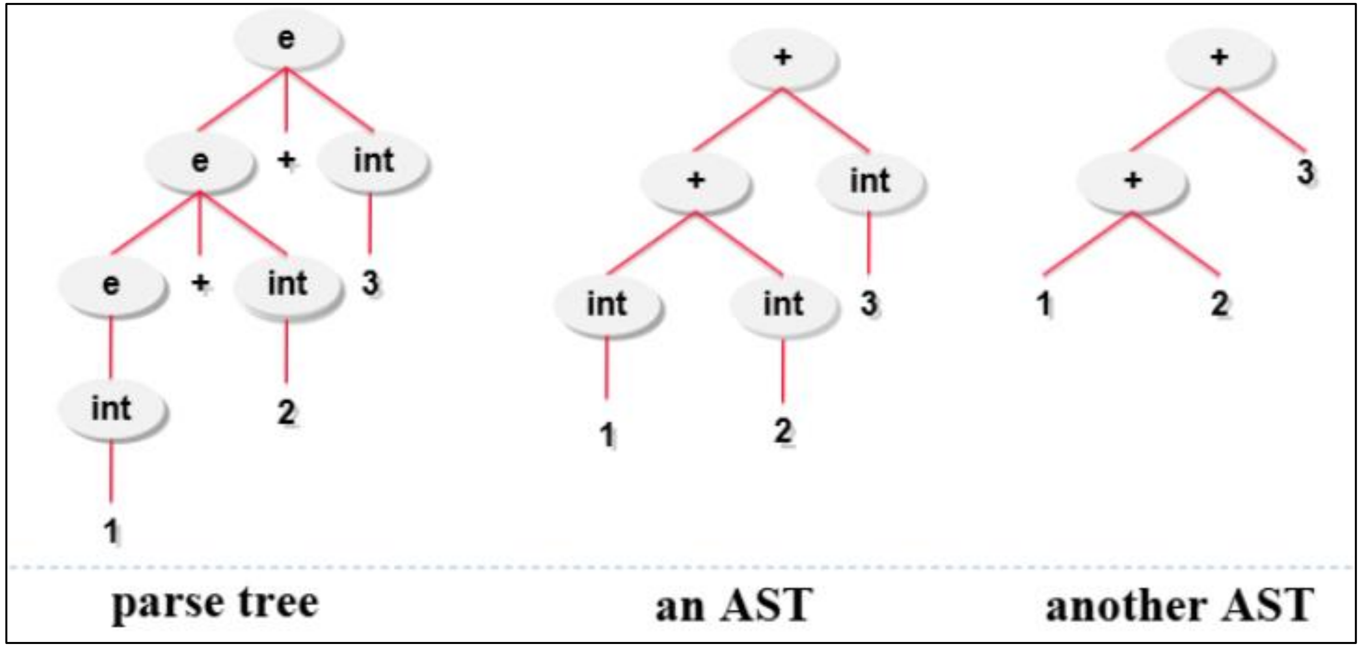
- Parse Tree: Hierarchical representation of a derivation





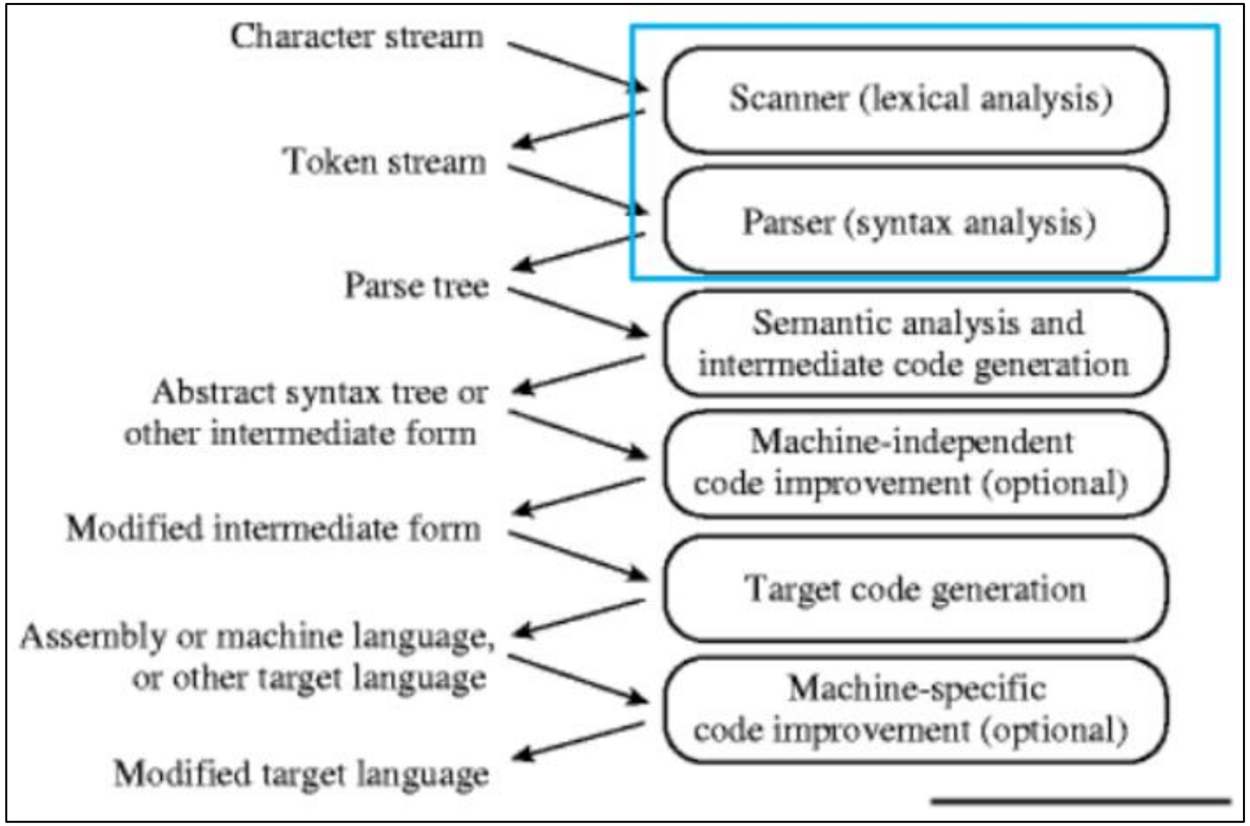
Course Review

- Parse Trees follow a grammar and have many nodes in which are artifacts of how the grammar was written
- Abstract Syntax Tree: A tree that eliminates useless structural nodes from a Parse Tree



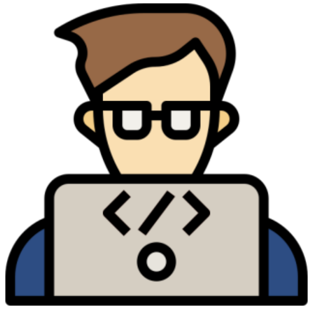
Course Review

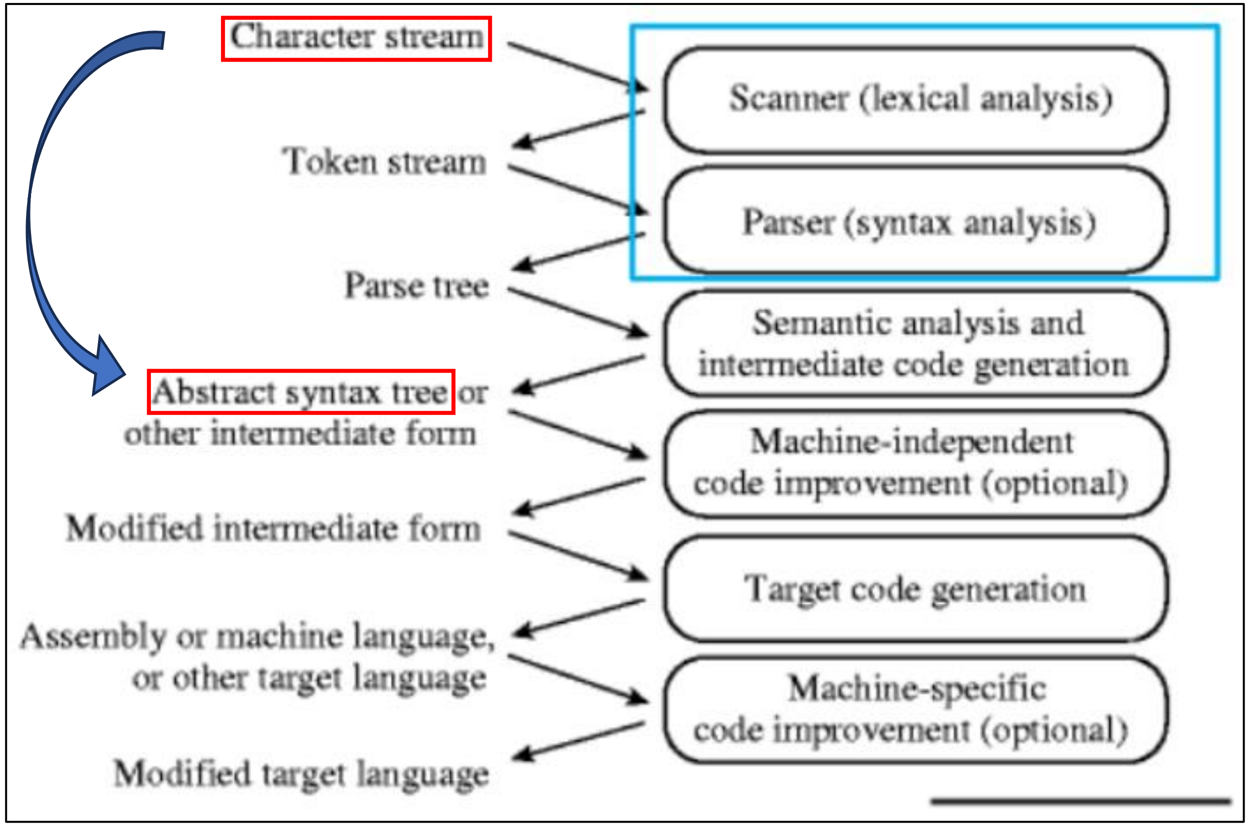
- Source Code => Executable Program converting process


Character stream
Scanner (lexical analysis)
Token stream
Parser (syntax analysis)
Parse tree
Semantic analysis and intermediate code generation
Abstract syntax tree or other intermediate form
Machine-independent code improvement (optional)
Modified intermediate form
Target code generation
Assembly or machine language, or other target language
Machine-specific code improvement (optional)
Modified target language

Assignment Goal 1

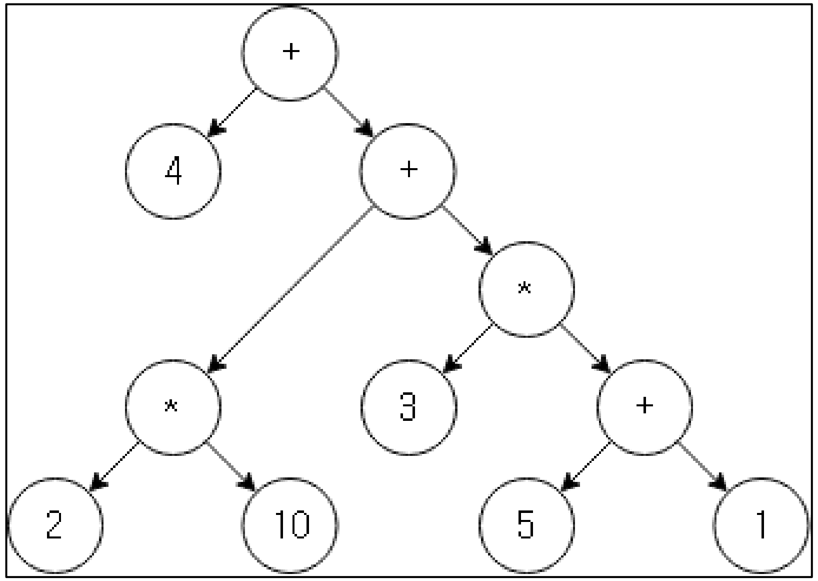
- The first goal of the assignment is to generate an abstract syntax tree(AST) from C source code through Python.




Character stream
Scanner (lexical analysis)
Token stream
Parser (syntax analysis)
Parse tree
Semantic analysis and intermediate code generation
Abstract syntax tree or other intermediate form
Machine-independent code improvement (optional)
Modified intermediate form
Target code generation
Assembly or machine language, or other target language
Machine-specific code improvement (optional)
Modified target language

Assignment Goal 2

- The second goal of the assignment is to traverse the generated AST and compute the output.



Generated AST

Traverse AST

Final Answer

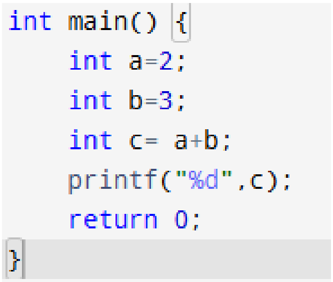
Assignment Explanation

- The assignment final form will be Python source code

- The program will accept a file path containing C source code as the first argument and generate the AST of the code.

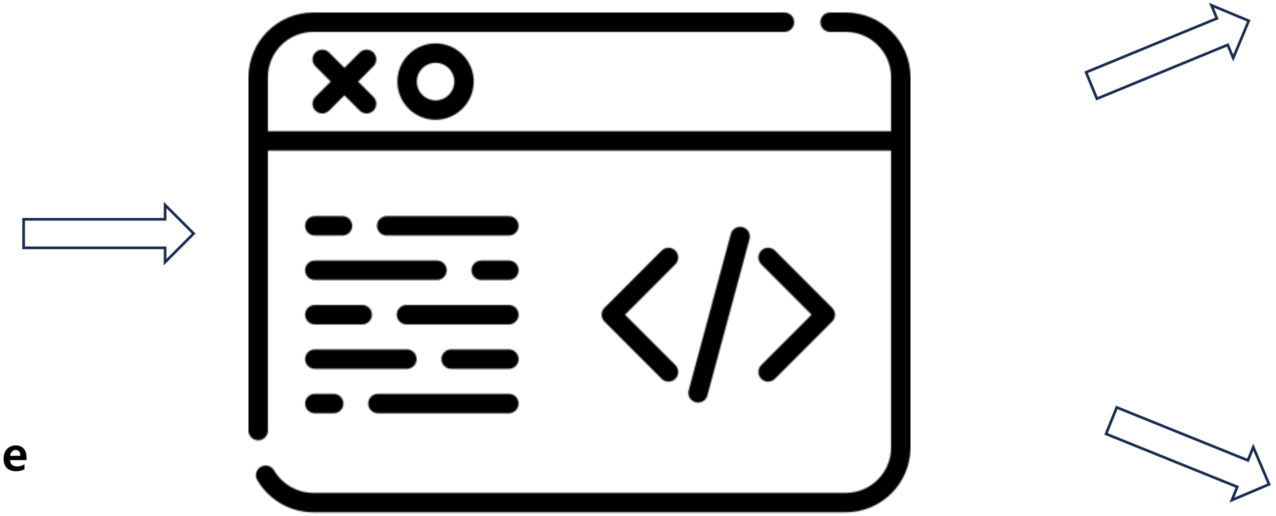
- Subsequently, the program will traverse the AST and compute the standard output of the source code.

Assignment Explanation



```
int main() {
    int a=2;
    int b=3;
    int c= a+b;
    printf("%d",c);
    return 0;
}
```

External C Source



Your Program

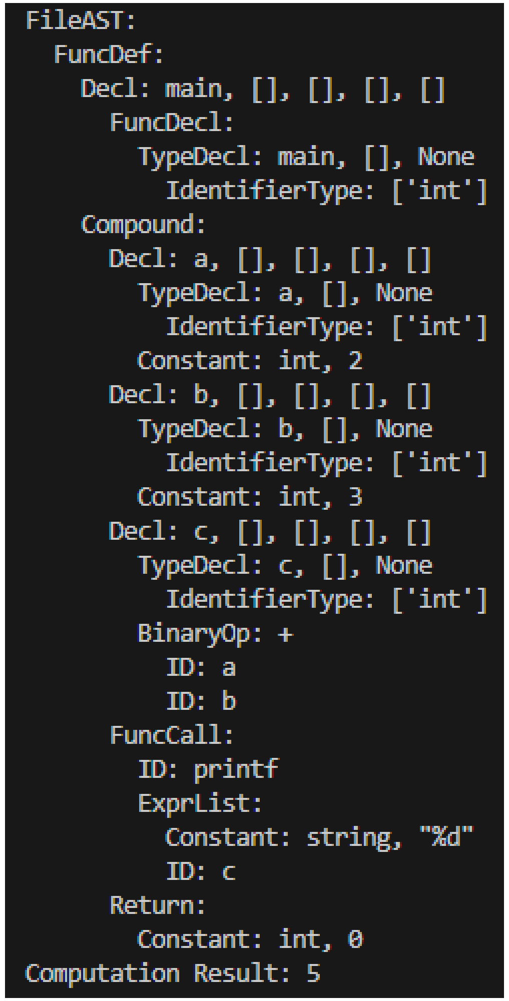
<AST>

Computation Result: 5

Output Example

- The program first prints the generated AST.
- The AST should be followed by the standard output of the C source code in the format of "Computation Result: [result]"

```
int main() {
    int a=2;
    int b=3;
    int c= a+b;
    printf("%d",c);
    return 0;
}
```

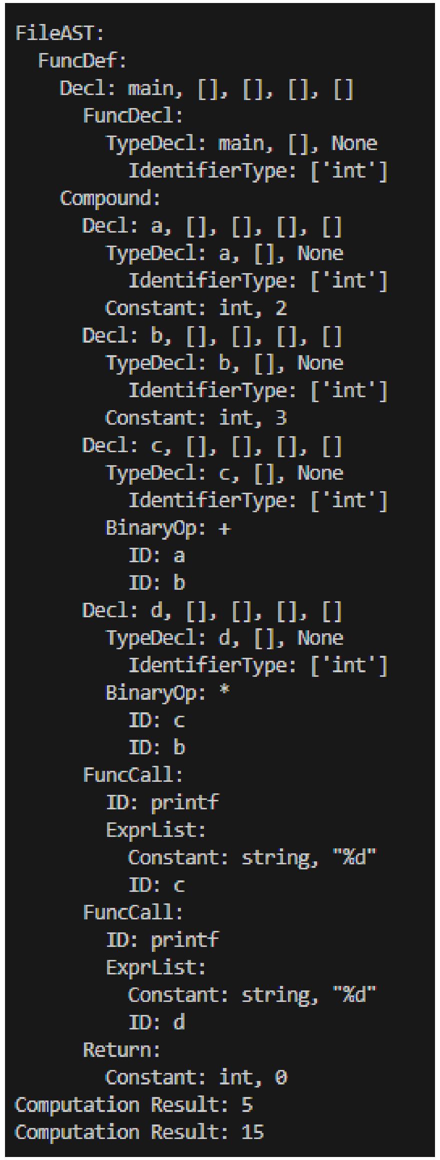


```
FileAST:
  FuncDef:
    Decl: main, [], [], [], []
      FuncDecl:
        TypeDecl: main, [], None
          IdentifierType: ['int']
    Compound:
      Decl: a, [], [], [], []
        TypeDecl: a, [], None
          IdentifierType: ['int']
        Constant: int, 2
      Decl: b, [], [], [], []
        TypeDecl: b, [], None
          IdentifierType: ['int']
        Constant: int, 3
      Decl: c, [], [], [], []
        TypeDecl: c, [], None
          IdentifierType: ['int']
        BinaryOp: +
          ID: a
          ID: b
      FuncCall:
        ID: printf
        ExprList:
          Constant: string, "%d"
          ID: c
      Return:
        Constant: int, 0
Computation Result: 5
```

Output Example

- There can be multiple calls to the printf() function. Each call should be handled in the format of “Computation Result: [result]”
- Each printf() call will only print a single variable without escape sequences (₩t, ₩n, etc)

```
int main() {
    int a=2;
    int b=3;
    int c= a+b;
    int d= c*b;
    printf("%d",c);
    printf("%d",d);
    return 0;
}
```



```
FileAST:
  FuncDef:
    Decl: main, [], [], [], []
      FuncDecl:
        TypeDecl: main, [], None
          IdentifierType: ['int']
    Compound:
      Decl: a, [], [], [], []
        TypeDecl: a, [], None
          IdentifierType: ['int']
        Constant: int, 2
      Decl: b, [], [], [], []
        TypeDecl: b, [], None
          IdentifierType: ['int']
        Constant: int, 3
      Decl: c, [], [], [], []
        TypeDecl: c, [], None
          IdentifierType: ['int']
        BinaryOp: +
          ID: a
          ID: b
      Decl: d, [], [], [], []
        TypeDecl: d, [], None
          IdentifierType: ['int']
        BinaryOp: *
          ID: c
          ID: b
      FuncCall:
        ID: printf
        ExprList:
          Constant: string, "%d"
          ID: c
      FuncCall:
        ID: printf
        ExprList:
          Constant: string, "%d"
          ID: d
      Return:
        Constant: int, 0
Computation Result: 5
Computation Result: 15
```

Output AST Format

- The format of the AST must be identical with the format of the show() function of the <pycparser.c_ast.FileAST> class of the pycparser library.

- The source code for the show() method can be found in

https://github.com/eliben/pycparser/blob/main/pycparser/c_ast.py

Output AST Format

```
from pycparser import CParser

c_code = """
int main(int x, int y) {
    int a=2;
    int b=3;
    int c= a+b;
    printf("%d",c);
    return 0;
}
"""
parser = CParser()
ast = parser.parse(c_code)
ast.show()
```

You can utilize above code for simple example of output format

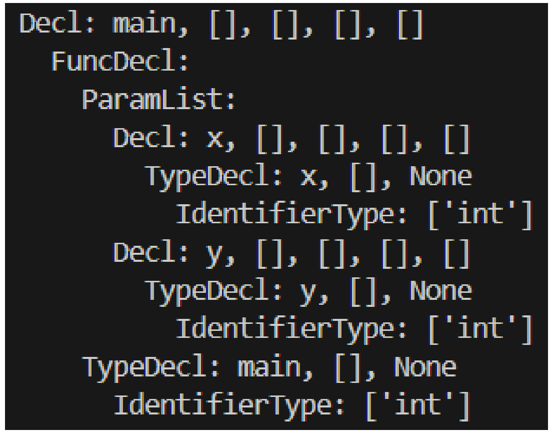
Output AST Format

- Each function has a FuncDecl (parameter list and type declaration) and a Compound (body)

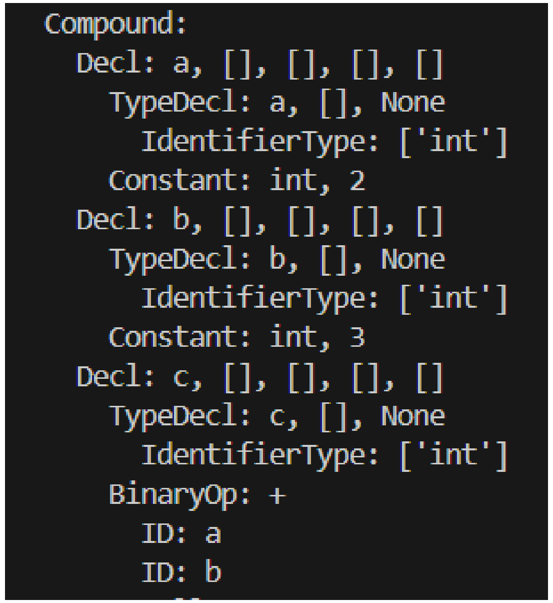
- Decl format follows
  ('name', 'quals', 'align', 'storage', 'funcspec’)
  ex) x, [], [], [], []

- TypeDecl format follows
  ('declname', 'quals', 'align’)
  ex) y, [], None

- You do not have to consider type qualifiers, alignment, storage classes and function specifications
  =>Decl will always have 4 [], TypeDecl 1 []+None



```
Decl: main, [], [], [], []
  FuncDecl:
    ParamList:
      Decl: x, [], [], [], []
        TypeDecl: x, [], None
          IdentifierType: ['int']
      Decl: y, [], [], [], []
        TypeDecl: y, [], None
          IdentifierType: ['int']
    TypeDecl: main, [], None
      IdentifierType: ['int']
```



```
  Compound:
    Decl: a, [], [], [], []
      TypeDecl: a, [], None
        IdentifierType: ['int']
      Constant: int, 2
    Decl: b, [], [], [], []
      TypeDecl: b, [], None
        IdentifierType: ['int']
      Constant: int, 3
    Decl: c, [], [], [], []
      TypeDecl: c, [], None
        IdentifierType: ['int']
      BinaryOp: +
        ID: a
        ID: b
```

Constraints of C Source(external file to be parsed into AST)

- The C source code will only contain arithmetic/bitwise expressions and function calls to user-defined functions.
  - No loops (for, while)
  - No conditional control (if, switch)

- The C source code will only contain int, float, double types.
  - You may neglect other types including structures and chars.

- The C source code will not utilize type qualifiers, alignment, storage classes or function specifications.

Constraints of Your Code

- Your program must be well documented with comments, explaining core logic.

- Usage of any external libraries are not allowed.

Any type of plagiarism, code sharing and usage of chat models like Chat-GPT will result in your final grade being F.

Evaluation

- Your program will be evaluated on test case C source code.

- If AST is generated correctly, you will get 70% for test case.
  - Your AST will be compared with the FileAST.show()

- If computation result is correct, you will get remaining 30%.

- There will be 10 test case C source code.

Submission

- You should submit the Python code and a report in Icampus.
  - Code: {student_id}_assignment2.py
  - Report: {student_id}_report.pdf
  - Compress the files into a zip file and submit the single zip file.
  - If you do not submit in this format, you will be given 0 points.

- Report format: Explain the logic behind your code in a concise manner within 3 pages.

- Final Score: 100pts
  - Code (AST generation and traversal): 80%
  - Report (explains the logic clearly): 20%