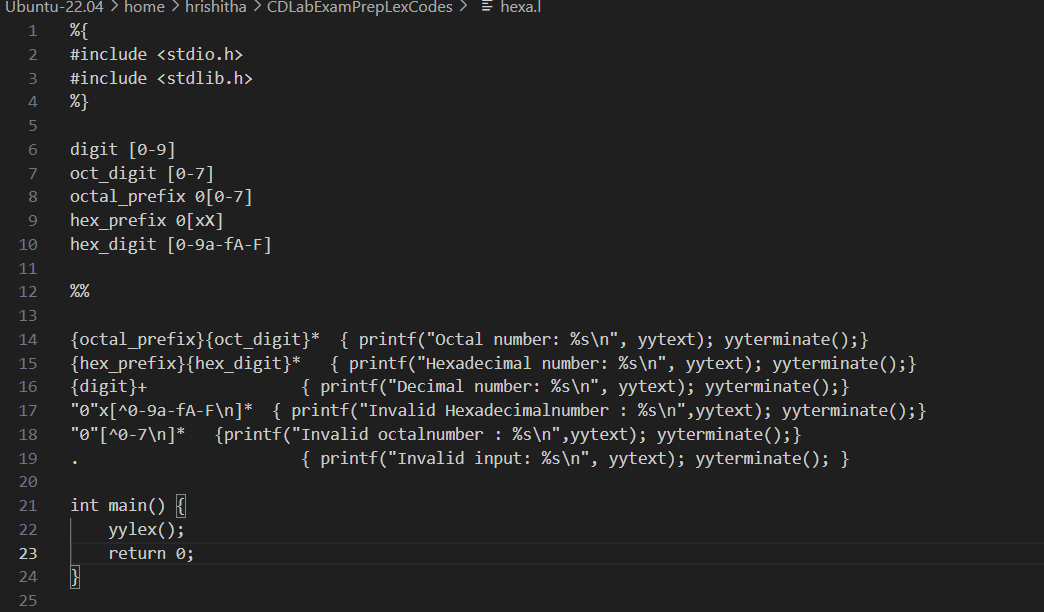
**Week 2 codes:**

1. Octal and hexadecimal

Code:



%{

#include <stdio.h>

#include <stdlib.h>

%}

digit [0-9]

oct\_digit [0-7]

octal\_prefix 0[0-7]

hex\_prefix 0[xX]

hex\_digit [0-9a-fA-F]

%%

{octal\_prefix}{oct\_digit}\* { printf("Octal number: %s\n", yytext); yyterminate();}

{hex\_prefix}{hex\_digit}\* { printf("Hexadecimal number: %s\n", yytext); yyterminate();}

{digit}+ { printf("Decimal number: %s\n", yytext); yyterminate();}

"0"x[^0-9a-fA-F\n]\* { printf("Invalid Hexadecimalnumber : %s\n",yytext); yyterminate();}

"0"[^0-7\n]\* {printf("Invalid octalnumber : %s\n",yytext); yyterminate();}

. { printf("Invalid input: %s\n", yytext); yyterminate(); }

int main() {

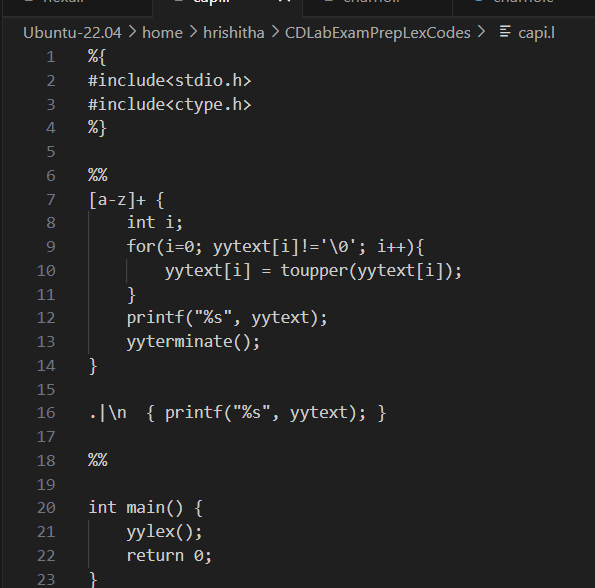
yylex();

return 0;

}

1. Capitalize input

Code:



%{

#include<stdio.h>

#include<ctype.h>

%}

%%

[a-z]+ {

int i;

for(i=0; yytext[i]!='\0'; i++){

yytext[i] = toupper(yytext[i]);

}

printf("%s", yytext);

yyterminate();

}

.|\n { printf("%s", yytext); }

%%

int main() {

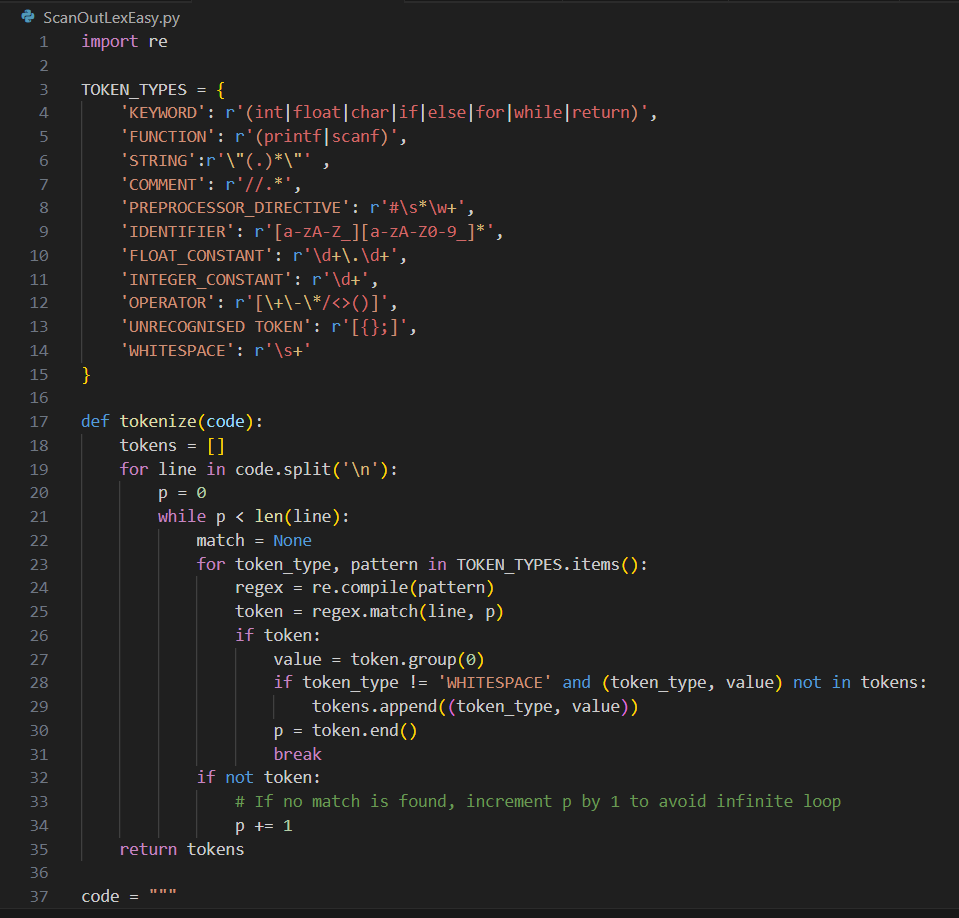
yylex();

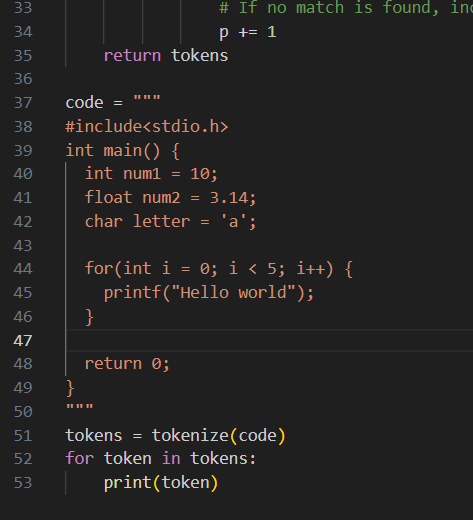
return 0;

}

1. Scanner without lex

Code:





import re

TOKEN\_TYPES = {

'KEYWORD': r'(int|float|char|if|else|for|while|return)',

'FUNCTION': r'(printf|scanf)',

'STRING':r'\"(.)\*\"' ,

'COMMENT': r'//.\*',

'PREPROCESSOR\_DIRECTIVE': r'#\s\*\w+',

'IDENTIFIER': r'[a-zA-Z\_][a-zA-Z0-9\_]\*',

'FLOAT\_CONSTANT': r'\d+\.\d+',

'INTEGER\_CONSTANT': r'\d+',

'OPERATOR': r'[\+\-\\*/<>()]',

'UNRECOGNISED TOKEN': r'[{};]',

'WHITESPACE': r'\s+'

}

def tokenize(code):

tokens = []

for line in code.split('\n'):

p = 0

while p < len(line):

match = None

for token\_type, pattern in TOKEN\_TYPES.items():

regex = re.compile(pattern)

token = regex.match(line, p)

if token:

value = token.group(0)

if token\_type != 'WHITESPACE' and (token\_type, value) not in tokens:

tokens.append((token\_type, value))

p = token.end()

break

if not token:

# If no match is found, increment p by 1 to avoid infinite loop

p += 1

return tokens

code = """

#include<stdio.h>

int main() {

int num1 = 10;

float num2 = 3.14;

char letter = 'a';

for(int i = 0; i < 5; i++) {

printf("Hello world");

}

return 0;

}

"""

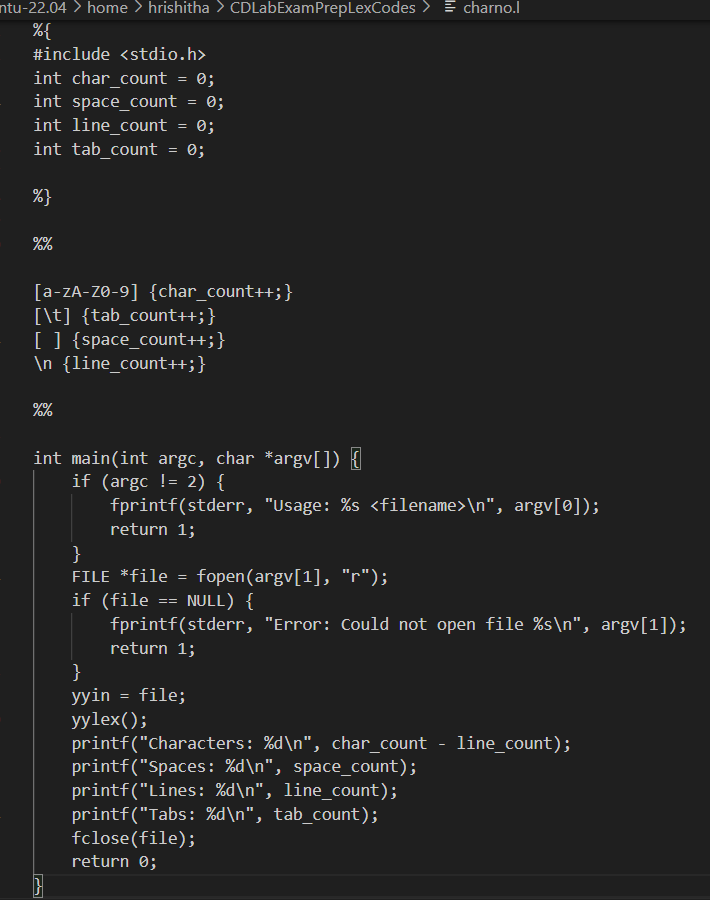
tokens = tokenize(code)

for token in tokens:

print(token)

**Week 3 codes:**

1. **Character count**

****

**%{**

**#include <stdio.h>**

**int char\_count = 0;**

**int space\_count = 0;**

**int line\_count = 0;**

**int tab\_count = 0;**

**%}**

**%%**

**[a-zA-Z0-9] {char\_count++;}**

**[\t] {tab\_count++;}**

**[ ] {space\_count++;}**

**\n {line\_count++;}**

**%%**

**int main(int argc, char \*argv[]) {**

**if (argc != 2) {**

**fprintf(stderr, "Usage: %s <filename>\n", argv[0]);**

**return 1;**

**}**

**FILE \*file = fopen(argv[1], "r");**

**if (file == NULL) {**

**fprintf(stderr, "Error: Could not open file %s\n", argv[1]);**

**return 1;**

**}**

**yyin = file;**

**yylex();**

**printf("Characters: %d\n", char\_count - line\_count);**

**printf("Spaces: %d\n", space\_count);**

**printf("Lines: %d\n", line\_count);**

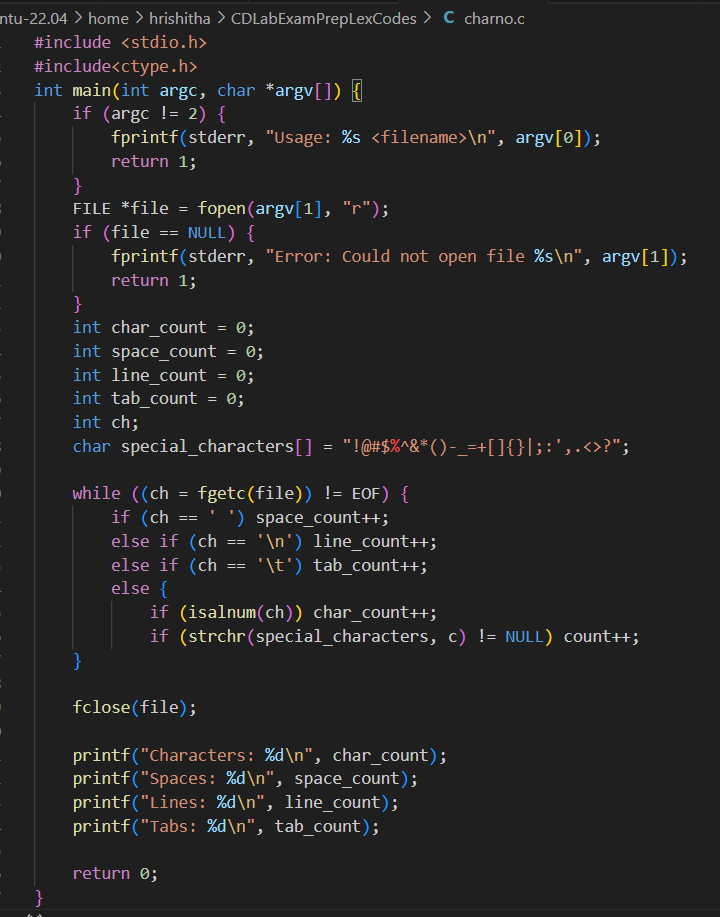
**printf("Tabs: %d\n", tab\_count);**

**fclose(file);**

**return 0;**

**}**

1. **Character count c program**

****

**#include <stdio.h>**

**#include<ctype.h>**

**int main(int argc, char \*argv[]) {**

**if (argc != 2) {**

**fprintf(stderr, "Usage: %s <filename>\n", argv[0]);**

**return 1;**

**}**

**FILE \*file = fopen(argv[1], "r");**

**if (file == NULL) {**

**fprintf(stderr, "Error: Could not open file %s\n", argv[1]);**

**return 1;**

**}**

**int char\_count = 0;**

**int space\_count = 0;**

**int line\_count = 0;**

**int tab\_count = 0;**

**int ch;**

**char special\_characters[] = "!@#$%^&\*()-\_=+[]{}|;:',.<>?";**

**while ((ch = fgetc(file)) != EOF) {**

**if (ch == ' ') space\_count++;**

**else if (ch == '\n') line\_count++;**

**else if (ch == '\t') tab\_count++;**

**else {**

**if (isalnum(ch)) char\_count++;**

**if (strchr(special\_characters, c) != NULL) count++;**

**}**

**fclose(file);**

**printf("Characters: %d\n", char\_count);**

**printf("Spaces: %d\n", space\_count);**

**printf("Lines: %d\n", line\_count);**

**printf("Tabs: %d\n", tab\_count);**

**return 0;**

**}**

**}**

1. **Tokenization by dfa**

**class DFA:**

**def \_\_init\_\_(self):**

**self.states = {'q0', 'q1'}**

**self.accept\_states = {'q1'}**

**self.transition = {**

**'q0': {'letter': 'q1'},**

**'q1': {'letter': 'q1', 'digit': 'q1'}**

**}**

**self.current\_state = 'q0'**

**def transition\_function(self, symbol):**

**if symbol.isalpha():**

**return 'letter'**

**elif symbol.isdigit():**

**return 'digit'**

**else:**

**return None**

**def is\_accept\_state(self):**

**return self.current\_state in self.accept\_states**

**def reset(self):**

**self.current\_state = 'q0'**

**def tokenize(self, input\_string):**

**tokens = []**

**current\_token = ''**

**for symbol in input\_string:**

**transition\_key = self.transition\_function(symbol)**

**if transition\_key is None:**

**if current\_token:**

**tokens.append(current\_token)**

**current\_token = ''**

**else:**

**next\_state = self.transition[self.current\_state][transition\_key]**

**self.current\_state = next\_state**

**current\_token += symbol**

**if current\_token:**

**tokens.append(current\_token)**

**return tokens**

**def main():**

**dfa = DFA()**

**input\_string = "var1 = 10 + var2"**

**tokens = dfa.tokenize(input\_string)**

**print("Input String:", input\_string)**

**print("Tokens:", tokens)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

1. **Scanner using lex**

**%{**

**#include <stdio.h>**

**%}**

**%option noyywrap**

**/\* Regular expressions for tokens \*/**

**DIGIT [0-9]**

**LETTER [a-zA-Z]**

**ID {LETTER}({LETTER}|{DIGIT})\***

**INT\_CONST {DIGIT}+**

**FLOAT\_CONST {DIGIT}+"."{DIGIT}+**

**WS [ \t\n\r]+**

**COMMENT ("//"(.)\*)**

**%%**

**\".\*\" { printf("%s is a string\n", yytext); }**

**\'.\*\' { printf("%s is a string\n", yytext); }**

**"for"|"if"|"else"|"while"|"int"|"char"|"float"|"return" { printf("%s is a keyword\n", yytext); }**

**"printf"|"scanf" {printf("%s is a function\n",yytext);}**

**{ID} { printf("Identifier: %s\n", yytext); }**

**{INT\_CONST} { printf("Integer Constant: %s\n", yytext); }**

**{FLOAT\_CONST} { printf("Float Constant: %s\n", yytext); }**

**"+"|"-"|"\*"|"/"|"="|"("|")"|"<"|">" { printf("Operator: %s\n",yytext); }**

**{WS} ; /\* Ignore whitespace \*/**

**{COMMENT} ; /\* Ignore comments \*/**

**"#"([^\n])\*\n { printf("Preprocessor Directive: %s\n", yytext); }**

**. { printf("Unrecognized token: %s\n", yytext); }**

**%%**

**int main(int argc, char \*argv[]) {**

**if (argc != 2) {**

**printf("Usage: %s input\_file\n", argv[0]);**

**return 1;**

**}**

**FILE \*file = fopen(argv[1], "r");**

**if (!file) {**

**perror("Error opening file");**

**return 1;**

**}**

**yyin = file;**

**yylex();**

**fclose(file);**

**return 0;**

**}**

**Week 4 codes:**

1. **Calculator**

**%{**

**#include <stdio.h>**

**#include <stdlib.h>**

**int op = 0;**

**float a, b;**

**void digi();**

**%}**

**dig [0-9]+|([0-9]\*)"."([0-9]+)**

**add "+"**

**sub "-"**

**mul "\*"**

**div "/"**

**pow "^"**

**ln \n**

**%%**

**{dig} { digi(); }**

**{add} { op = 1; }**

**{sub} { op = 2; }**

**{mul} { op = 3; }**

**{div} { op = 4; }**

**{pow} { op = 5; }**

**{ln} { printf("\n The Answer : %f\n\n", a); }**

**%%**

**void digi()**

**{**

**if (op == 0)**

**a = atof(yytext);**

**else**

**{**

**b = atof(yytext);**

**switch (op)**

**{**

**case 1:**

**a = a + b;**

**break;**

**case 2:**

**a = a - b;**

**break;**

**case 3:**

**a = a \* b;**

**break;**

**case 4:**

**a = a / b;**

**break;**

**case 5:**

**for (int i = 1; i < b; i++)**

**a \*= a;**

**break;**

**}**

**op = 0;**

**}**

**}**

**int main()**

**{**

**yylex();**

**return 0;**

**}**

**int yywrap()**

**{**

**return 1;**

**}**

1. **Count of printf scanf statements**

**%{**

**int printf\_count = 0;**

**int scanf\_count = 0;**

**%}**

**%%**

**"printf" { printf\_count++; }**

**"scanf" { scanf\_count++; }**

**. ;**

**%%**

**int yywrap() {**

**return 1;**

**}**

**int main(int argc, char \*argv[]) {**

**if (argc != 2) {**

**printf("Usage: %s <input\_file>\n", argv[0]);**

**return 1;**

**}**

**FILE \*fp = fopen(argv[1], "r");**

**if (fp == NULL) {**

**printf("Error opening file.\n");**

**return 1;**

**}**

**yyin = fp;**

**yylex();**

**printf("Number of printf statements: %d\n", printf\_count);**

**printf("Number of scanf statements: %d\n", scanf\_count);**

**fclose(fp);**

**return 0;**

**}**

1. **No of identifiers in file**

**%{**

**#include<stdio.h>**

**#include<string.h>**

**#define MAX\_IDENTIFIERS 100**

**int count = 0;**

**char identifiers[MAX\_IDENTIFIERS][100];**

**%}**

**letter [a-zA-Z]**

**digit [0-9]**

**id ({letter}({letter}|{digit})\*)**

**%%**

**\".\*\" { printf("%s is a string\n", yytext); }**

**\'.\*\' { printf("%s is a string\n", yytext); }**

**^#.\* { printf("%s is a preprocessor directive\n", yytext); }**

**"for"|"if"|"else"|"while"|"int"|"char"|"float"|"return" { printf("%s is a keyword\n", yytext); }**

**"printf"|"scanf" {printf("%s is a function\n",yytext);}**

**{id} {**

**int is\_repeat = 0;**

**for (int j = 0; j < count; j++) {**

**if (strcmp(identifiers[j], yytext) == 0) {**

**is\_repeat = 1;**

**break;**

**}**

**}**

**if (!is\_repeat) {**

**printf("%s is an identifier\n", yytext);**

**strcpy(identifiers[count], yytext);**

**count++;**

**}**

**}**

**. ;**

**%%**

**int yywrap(){**

**return 1;**

**}**

**int main()**

**{**

**FILE \*fp;**

**char file[100];**

**printf("\nEnter the filename: ");**

**scanf("%s", file);**

**fp = fopen(file, "r");**

**if (fp == NULL) {**

**printf("File not found\n");**

**return 1;**

**}**

**yyin = fp;**

**yylex();**

**printf("Total unique identifiers are: %d\n", count);**

**fclose(fp);**

**return 0;**

**}**

1. **First and follow**

**class First\_Follow():**

**def \_\_init\_\_(self, grammar):**

**self.grammar = grammar**

**self.non\_terminals = grammar.keys()**

**print(self.non\_terminals)**

**self.start = list(self.non\_terminals)[0]**

**self.rules = [(head, body) for head, bodies in grammar.items() for body in bodies]**

**def compute\_first(self, variable):**

**first = set()**

**productions = [rule[1] for rule in self.rules if rule[0] == variable]**

**for production in productions:**

**if not production[0].isupper():**

**first.add(production[0])**

**else:**

**for x in production:**

**first |= self.compute\_first(x)**

**if "@" not in first:**

**break**

**return first**

**def compute\_follow(self, variable):**

**follow = set()**

**if variable == self.start:**

**follow.add('$')**

**for rule in self.rules:**

**for j, char in enumerate(rule[1]):**

**if char == variable:**

**while j < len(rule[1]) - 1:**

**if not rule[1][j + 1].isupper():**

**follow.add(rule[1][j + 1])**

**break**

**else:**

**follow |= self.compute\_first(rule[1][j + 1])**

**if '@' not in self.compute\_first(rule[1][j + 1]):**

**break**

**j += 1**

**else:**

**if rule[0] != variable:**

**follow |= self.compute\_follow(rule[0])**

**follow.discard('@')**

**return follow**

**def print\_sets(self):**

**print("First Sets:")**

**for non\_terminal in self.non\_terminals:**

**print(f"{non\_terminal}: {self.compute\_first(non\_terminal)}")**

**print("\nFollow Sets:")**

**for non\_terminal in self.non\_terminals:**

**print(f"{non\_terminal}: {self.compute\_follow(non\_terminal)}")**

**# ε**

**def main():**

**# example\_grammar = {**

**# 'E': ['TZ'],**

**# 'Z': ['+TZ', '@'],**

**# 'T': ['FY'],**

**# 'Y': ['\*FY', '@'],**

**# 'F': ['(E)', 'i'],**

**# }**

**example\_grammar = {**

**'S' : ['CC'],**

**'C' : ['cC' , 'd']**

**}**

**ff = First\_Follow(example\_grammar)**

**print("epsilon is printed as @")**

**ff.print\_sets()**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

**Week 5 codes:**

1. **Ll parsing table**

**class First\_Follow():**

**def \_\_init\_\_(self, grammar):**

**self.grammar = grammar**

**self.non\_terminals = grammar.keys()**

**print(self.non\_terminals)**

**self.start = list(self.non\_terminals)[0]**

**self.rules = [(head, body) for head, bodies in grammar.items() for body in bodies]**

**def compute\_first(self, variable):**

**first = set()**

**productions = [rule[1] for rule in self.rules if rule[0] == variable]**

**for production in productions:**

**if not production[0].isupper():**

**first.add(production[0])**

**else:**

**first |= self.compute\_first(production[0])**

**return first**

**def compute\_follow(self, variable):**

**follow = set()**

**if variable == self.start:**

**follow.add('$')**

**for rule in self.rules:**

**for j, char in enumerate(rule[1]):**

**if char == variable:**

**while j < len(rule[1]) - 1:**

**if not rule[1][j + 1].isupper():**

**follow.add(rule[1][j + 1])**

**break**

**else:**

**follow |= self.compute\_first(rule[1][j + 1])**

**if '@' not in self.compute\_first(rule[1][j + 1]):**

**break**

**j += 1**

**else:**

**if rule[0] != variable:**

**follow |= self.compute\_follow(rule[0])**

**follow.discard('@')**

**return follow**

**def print\_sets(self):**

**print("First Sets:")**

**for non\_terminal in self.non\_terminals:**

**print(f"{non\_terminal}: {self.compute\_first(non\_terminal)}")**

**print("\nFollow Sets:")**

**for non\_terminal in self.non\_terminals:**

**print(f"{non\_terminal}: {self.compute\_follow(non\_terminal)}")**

**def computeFirstOneRHS(self, variable):**

**first = set()**

**if not variable[0].isupper():**

**first.add(variable[0])**

**else:**

**for x in variable:**

**first |= self.compute\_first(x)**

**if '@' not in first:**

**break**

**return first**

**def compute\_parsing\_table(self):**

**print('\nParsing Table')**

**table = {}**

**for rule in self.rules:**

**rule1, rule2 = rule**

**first = list(self.computeFirstOneRHS(rule2))**

**if '@' in first:**

**first.extend(self.compute\_follow(rule1))**

**while '@' in first:**

**first.remove('@')**

**for terminal in first:**

**key = (rule1, terminal)**

**if key in table:**

**table[key].append(rule2)**

**else:**

**table[key] = [rule2]**

**for key, value in table.items():**

**print(f'{key} : {value}')**

**return table**

**# @**

**def main():**

**example\_grammar = {**

**'E': ['TA'],**

**'A': ['+TA', '@'],**

**'T': ['FB'],**

**'B': ['\*FB', '@'],**

**'F': ['(E)', 'i']**

**}**

**ff = First\_Follow(example\_grammar)**

**ff.print\_sets()**

**ans = ff.compute\_parsing\_table()**

**print(ans)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**