Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

Assignment 6

Course No.: CSE4130

Course Title: Formal Languages and Compilers Lab

Date of Submission-

23/08/2023

Submitted By-

MD Fardin Jaman Aranyak

190204093

B2

Year- 4th

Semester-1st

Department-CSE

```
import re
kws = ["int", "double", "float", "char", "for", "while", "do", "if", "else", "switch", "case"]
ops = "+-*/%=<>!|&"
pars = "(){}[]"
seps = ",;'\""
op = [None] * 5
opflag = 0  # Define opflag here
ids = []
output = ""

def assingment1():
    def remove_comments(line):
        in_string = False
        new_line = []
        i = 0

        while i < len(line):
            if line[i] == '"' or line[i] == "'":
                in_string = not in_string
                new_line.append(line[i])
                i += 1
                continue

            if not in_string:
                if line[i:i+2] == '//':
                    break
                elif line[i:i+2] == '/*':
                    while i < len(line) - 1 and line[i:i+2] != '*/':
                        i += 1
                    i += 2  # Skip '*/'
                    continue

            new_line.append(line[i])
            i += 1

        return ''.join(new_line)

    def main():
        input_filename = "input.c"
        output_filename = "output.c"
        filtered_code = []

        with open(input_filename, "r") as fp:
            in_multi_line_comment = False

            for line in fp:
                line = line.strip()  # Remove leading/trailing whitespace
                if in_multi_line_comment:
                    if '*/' in line:
                        in_multi_line_comment = False
                        _, line = line.split('*/', 1)
                    else:
                        continue

                line = remove_comments(line)

                if not line:
                    continue  # Skip empty lines

                filtered_code.append(line)

                if '/*' in line:
                    if '*/' not in line:
                        in_multi_line_comment = True

        single_line_code = ' '.join(filtered_code)

        with open(output_filename, "w") as fw:
            fw.write(single_line_code)

        print(single_line_code)

    if __name__ == "__main__":
        main()
def assignment2():
    import re

    kws = ["int", "double", "float", "char", "for", "while", "do", "if", "else", "switch", "case"]
    ops = "+-*/%=<>!|&"
    pars = "(){}[]"
    seps = ",;'\""
```

```
ids = {}
op = []
c = ''
s = ''
err = 0
rf = None

def read_file():
    global rf
    filename = input("\nEnter the filename: ")
    try:
        rf = open(filename, "r")
    except:
        print("Error opening file.")

def remove_comments(line):
    in_string = False
    new_line = []
    i = 0

    while i < len(line):
        if line[i] == '"' or line[i] == "'":
            in_string = not in_string
            new_line.append(line[i])
            i += 1
            continue

        if not in_string:
            if line[i:i+2] == '//':
                break
            elif line[i:i+2] == '/*':
                while i < len(line) - 1 and line[i:i+2] != '*/':
                    i += 1
                i += 2  # Skip '*/'
                continue

        new_line.append(line[i])
        i += 1

    return ''.join(new_line)
```

```
def write_to_file(filename, content):
    with open(filename, "w") as file:
        file.write(content)

def read_and_print_file(filename):
    # Open the file in read mode
    with open(filename, "r") as file:
        contents = file.read()

    # Print the contents of the file
    print(contents)

def lexemes():
    global s, op, c, err, rf,output
    code = remove_comments(rf.read())
    i = 0
    line_number = 1
    err=0
    while i < len(code):
        c = code[i]


        if c == '\n':
            line_number += 1
            i += 1
        elif c in seps:
            print(f"[sep {c}] ", end='')
            output+=f"[sep {c}] "
            i += 1
        elif c in pars:
            print(f"[par {c}] ", end='')
            output+=f"[par {c}] "
            i += 1
        elif c in ops:
            op = []
            while i < len(code) and code[i] in ops:
                op.append(code[i])
                i += 1
            print(f"[op {''.join(op)}] ", end='')
            output+=f"[op {''.join(op)}] "
        elif c.isalpha() or c == '_':
            s = ''
            while i < len(code) and (code[i].isalnum() or code[i] == '_'):
                s += code[i]
                i += 1
            if s in kws:
                print(f"[kw {s}] ", end='')
                output+=f"[kw {s}] "
            elif s == "return":
                print(f"[ret {s}] ", end='')
                output+=f"[ret {s}] "
            else:
                if s not in ids:
                    ids[s] = len(ids) + 1
                print(f"[id {ids[s]}] ", end='')
                output+=f"[id {ids[s]}] "
        elif c.isdigit() or (c == '.' and i + 1 < len(code) and code[i + 1].isdigit()):
            s = ''
            while i < len(code) and (code[i].isdigit() or code[i] == '.'):
                s += code[i]
                i += 1
            print(f"[num {s}] ", end='')
            output+=f"[num {s}] "
        else:
            i += 1
            err += 1

    print("\nErrors:", err)
    print("Line number:", line_number)
    return output
  def main():
    read_file()
    xoutput=lexemes()
    write_to_file("lexemes.txt",xoutput)
    #read_and_print_file("lexemes.txt")

  if __name__ == "__main__":
    main()

def assignment3():
  #variable
  lexemes=""
  copy_lexemes=""
```

```
    tokenToBeRemove=["kw","op","num","sep","par","brc"]
    dataType=["double","int","float"]
    id_names_withDataType=[]
    id_names_withType=[]
    id_names_withValue=[]
    symbol_table=[]

    #read file
    file = open("lexemes.txt","r")
    lexemes=file.read()

    #create space between [ ]
    for i in range(len(lexemes)):
        if lexemes[i]=="[":
            copy_lexemes+=lexemes[i]+" "
        elif lexemes[i]=="]":
             copy_lexemes+=" "+lexemes[i]
        else:
            copy_lexemes+=lexemes[i]

    #print(copy_lexemes)
    print()

    #seperate every keyword
    lexemes_list=copy_lexemes.split()

    #only identifiers are kept
    for items in  lexemes_list:
        if(items in tokenToBeRemove):
            lexemes_list.remove(items)

    for items in lexemes_list:
        print(items,end=" ")
    print()
def assignment4():
    def tokenize_code(code):
        tokens = []
        token = ""
        in_multi_line_comment = False
        line = 1
    for i in range(len(code)):
        if in_multi_line_comment:
            if code[i] == '*' and i + 1 < len(code) and code[i + 1] == '/':
                in_multi_line_comment = False
                i += 1  # Skip the '/'
        else:
            if code[i] == '/' and i + 1 < len(code):
                if code[i + 1] == '*':
                    in_multi_line_comment = True
                    i += 1  # Skip the '*'
                elif code[i + 1] == '/':
                    # Skip the rest of the line (including the newline character)
                    while i < len(code) and code[i] != '\n':
                        i += 1
            elif code[i] == ' ':
                if token:
                    tokens.append(token)
                    token = ""
            elif code[i] == '\n':
                if token:
                    tokens.append(token)
                    token = ""
                tokens.append("\n")
                line += 1
            elif code[i:i+2] == ";;":
                print(f"Duplicate semicolons ';;' at line {line}")
                i += 1  # Skip the second semicolon
            elif code[i:i+2] == "}}":
                print(f"Duplicate closing curly braces '}}' at line {line}")
                i += 1  # Skip the second closing curly brace
            else:
                token += code[i]

    if token:
        tokens.append(token)
    return tokens

def detect_duplicate_keywords(tokens):
    prev_keyword = None
    line = 1
    for token in tokens:
            if token in {"if", "else", "for", "while", "do"}:
                if prev_keyword == token:
                    print(f"Duplicate keyword '{token}' at line {line}")
                prev_keyword = token
            elif token == "\n":
                line += 1
                prev_keyword = None

    def detect_unbalanced_braces(tokens):
        stack = []
        line = 1
        for token in tokens:
            if token == "{":
                stack.append(("brace", line))
            elif token == "}":
                if not stack:
                    print(f"Unmatched '}}' at line {line}")
                elif stack[-1][0] == "brace":
                    stack.pop()
                else:
                    print(f"Unmatched '{{' at line {line}")
            elif token == "\n":
                line += 1

    def main():
        # Read input from the source file
        with open("input.c", "r") as input_file:
            code = input_file.read()

        tokens = tokenize_code(code)
        detect_duplicate_keywords(tokens)
        detect_unbalanced_braces(tokens)

    if __name__ == "__main__":
        main()


print("Assignment 1")
assingment1()
print("--------------------------------------------")
print("Assignment 2")
```

```
assignment2()
print("--------------------------------------------")
print("Assignment 3")
assignment3()
print("--------------------------------------------")
print("Assignment 4")
assignment4()
print("--------------------------------------------")
```