Compiler Design

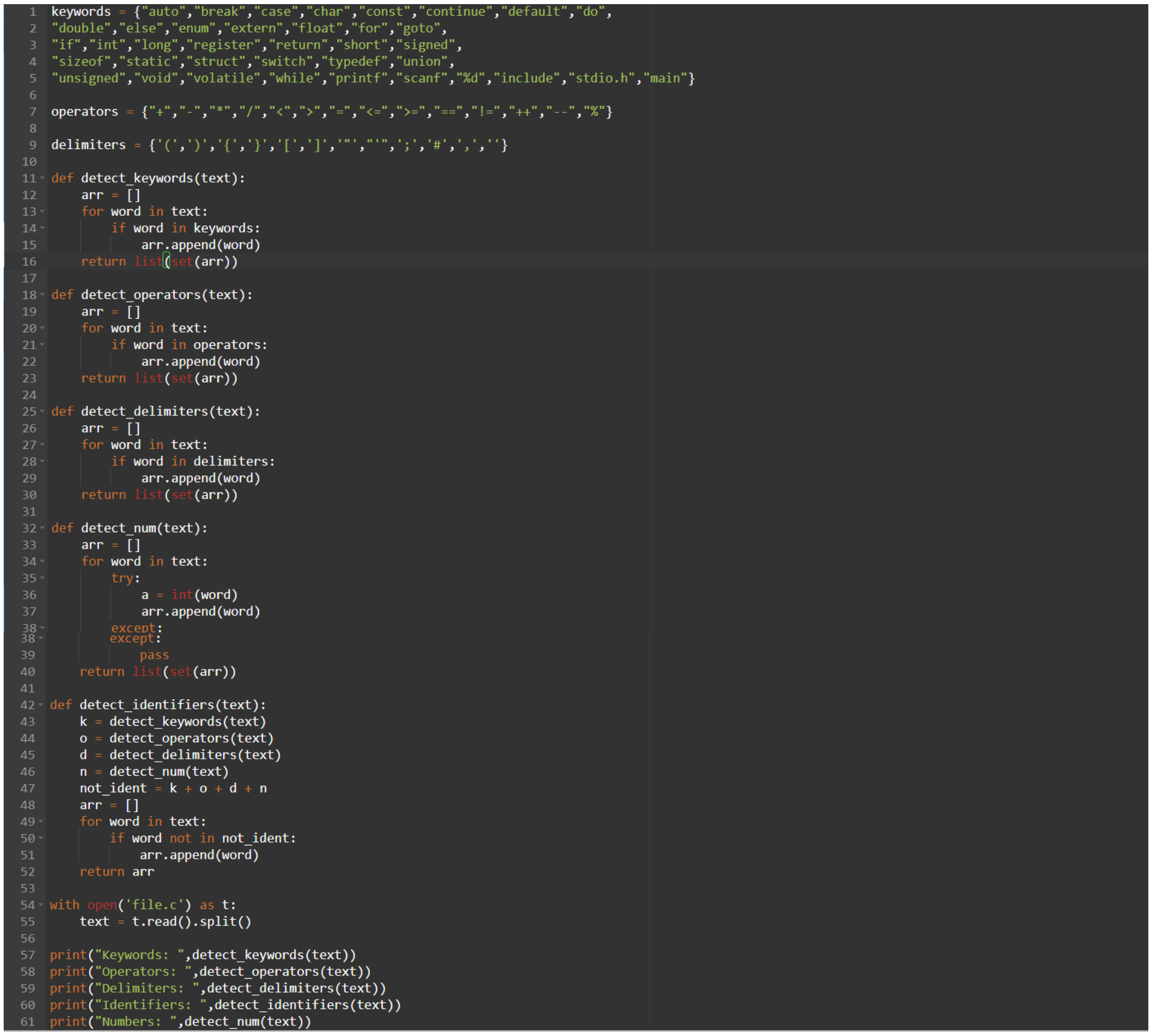
Exp_01 - Implementation of Lexical Analyser

Ayush Jindal

RA1911003010308

Source Code:

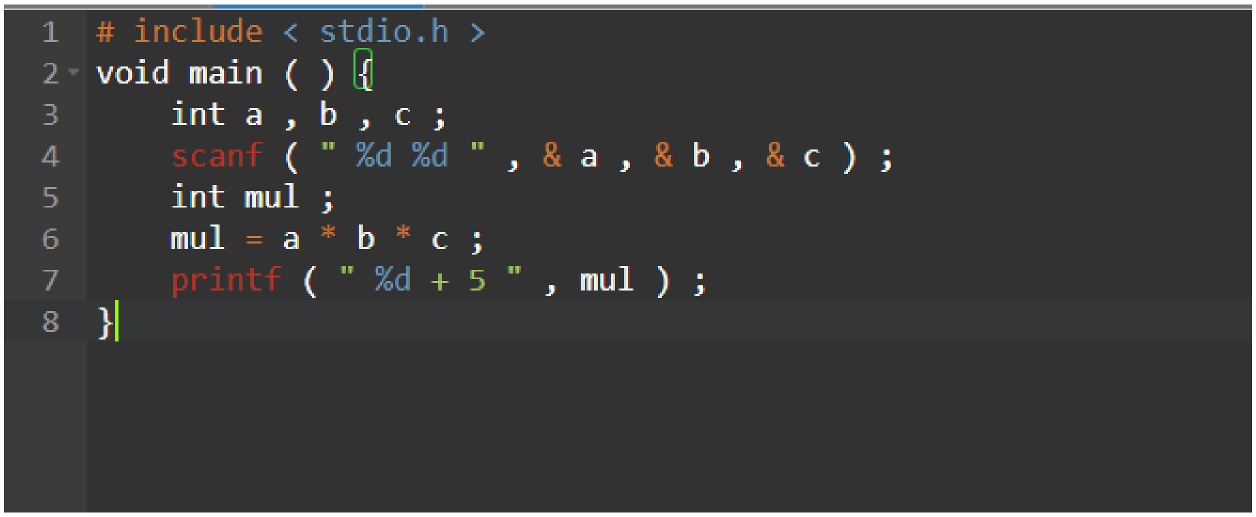
```
keywords = {"auto","break","case","char","const","continue","default","do",
"double","else","enum","extern","float","for","goto",
"if","int","long","register","return","short","signed",
"sizeof","static","struct","switch","typedef","union",
"unsigned","void","volatile","while","printf","scanf","%d","include","stdio.h","main"}

operators = {"+","-","*","/","<",">","=","<=",">=","==","!=","++","--","%"}

delimiters = {'(',')','{','}','[',']','"',"'",';','#',',',''}

def detect_keywords(text):
	arr = []
	for word in text:
		if word in keywords:
			arr.append(word)
	return list(set(arr))

def detect_operators(text):
	arr = []
	for word in text:
		if word in operators:
			arr.append(word)
	return list(set(arr))

def detect_delimiters(text):
	arr = []
	for word in text:
		if word in delimiters:
			arr.append(word)
	return list(set(arr))

def detect_num(text):
        arr = []
        for word in text:
                try:
                        a = int(word)
                        arr.append(word)
                except:
                        pass
        return list(set(arr))

def detect_identifiers(text):
        k = detect_keywords(text)
        o = detect_operators(text)
        d = detect_delimiters(text)
        n = detect_num(text)
        not_ident = k + o + d + n
        arr = []
        for word in text:
                if word not in not_ident:
                        arr.append(word)
        return arr

with open('file.c') as t:
        text = t.read().split()

print("Keywords: ",detect_keywords(text))
print("Operators: ",detect_operators(text))
print("Delimiters: ",detect_delimiters(text))
print("Identifiers: ",detect_identifiers(text))
print("Numbers: ",detect_num(text))
```



```python
keywords = {"auto","break","case","char","const","continue","default","do",
"double","else","enum","extern","float","for","goto",
"if","int","long","register","return","short","signed",
"sizeof","static","struct","switch","typedef","union",
"unsigned","void","volatile","while","printf","scanf","%d","include","stdio.h","main"}

operators = {"+","-","*","/","<",">","=","<=",">=","==","!=","++","--","%"}

delimiters = {'(',')','{','}','[',']','"',"'",';','#',',',''}

def detect_keywords(text):
    arr = []
    for word in text:
        if word in keywords:
            arr.append(word)
    return list(set(arr))

def detect_operators(text):
    arr = []
    for word in text:
        if word in operators:
            arr.append(word)
    return list(set(arr))

def detect_delimiters(text):
    arr = []
    for word in text:
        if word in delimiters:
            arr.append(word)
    return list(set(arr))

def detect_num(text):
    arr = []
    for word in text:
        try:
            a = int(word)
            arr.append(word)
        except:
            pass
    return list(set(arr))

def detect_identifiers(text):
    k = detect_keywords(text)
    o = detect_operators(text)
    d = detect_delimiters(text)
    n = detect_num(text)
    not_ident = k + o + d + n
    arr = []
    for word in text:
        if word not in not_ident:
            arr.append(word)
    return arr

with open('file.c') as t:
    text = t.read().split()

print("Keywords: ",detect_keywords(text))
print("Operators: ",detect_operators(text))
print("Delimiters: ",detect_delimiters(text))
print("Identifiers: ",detect_identifiers(text))
print("Numbers: ",detect_num(text))
```

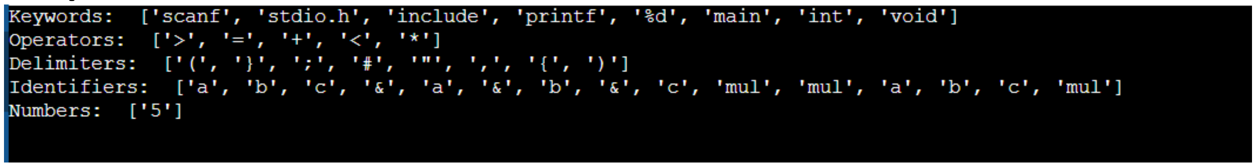
Input Code:

```c
# include < stdio.h >
void main ( ) {
   int a , b , c ;
   scanf ( " %d %d " , & a , & b , & c ) ;
   int mul ;
   mul = a * b * c ;
   printf ( " %d + 5 " , mul ) ;
}
```



```
# include < stdio.h >
void main ( ) {
    int a , b , c ;
    scanf ( " %d %d " , & a , & b , & c ) ;
    int mul ;
    mul = a * b * c ;
    printf ( " %d + 5 " , mul ) ;
}
```

Output:



```
Keywords:  ['scanf', 'stdio.h', 'include', 'printf', '%d', 'main', 'int', 'void']
Operators:  ['>', '=', '+', '<', '*']
Delimiters:  ['(', '}', ';', '#', '"', ',', '{', ')']
Identifiers:  ['a', 'b', 'c', '&', 'a', '&', 'b', '&', 'c', 'mul', 'mul', 'a', 'b', 'c', 'mul']
Numbers:  ['5']
```