Name: Pradyumn R Pai
Roll No: 50
Class: CS7A

PROGRAM CODE

```
#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h> //For isalnum, isalpha, isdigit, isspace


typedef enum {
    TOKEN_KEYWORD,
    TOKEN_IDENTIFIER,
    TOKEN_INT_CONST,
    TOKEN_STRING_LITERAL,
    TOKEN_OPERATOR,
    TOKEN_PUNCTUATOR,
    TOKEN_UNKNOWN,
    TOKEN_DIRECTIVE,
    TOKEN_END
} TokenType;


const char* TokenTypeNames[] = {
    "Keyword",
    "Identifier",
    "Integer Constant",
    "String Literal",
    "Operator",
    "Punctuator",
    "Unknown",
    "Compiler Directive"
};
```

```
/* operators match the regex: (>>=)|(<<=)|(\/=)|(->)|(%=)|(\*=)|(&=)|(--)|(-=)|(\+\+)|(\^=)|
(&&)|(>=)|(<<)|(<=)|(\|\|)|(>>)|(!=)|(\+=)|(==)|(\|=)|[>|+/=%&^*!~.\-<] */

char operators[][10] = {"+","-","*","/","%","<",">","!","&","|","^","~","=",".","+
+","--","<=",">=","==","!=","&&","||","<<",">>","+=","-=","*=","/=","%=","&=","|=","^=","-
>",">>=","<<="};

char punctuators[] = "{}[](),;:.?";


char keywords[][10] = {"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"};


char buffer[1024] = "";

int bufferLength = 0;

bool atNewLine = true;


bool isSubset(char* buffer, char charSet[][10], int len) {
   for (int i=0;i<len;++i){
      if (strcmp(buffer,charSet[i])==0){
         return true;
      }
   }
   return false;
}


bool isInteger(char* buffer){
   if (buffer[0]=='0' && buffer[1]=='\0') return true;
   if (buffer[0]=='0') return false;


   int n = strlen(buffer);
   for (int i=0;i<n;++i){
      if (!isdigit(buffer[i])) return false;
   }
```

```
    return true;
}

bool isKeyword(char* buffer){
    return isSubset(buffer,keywords,sizeof(keywords)/sizeof(keywords[0]));
}

bool isPunctuator(char c){
    int n = sizeof(punctuators);
    for (int i=0;i<n;++i){
        if (c==punctuators[i]) return true;
    }
    return false;
}

bool isOperator(char* buffer){
    return isSubset(buffer,operators,sizeof(operators)/sizeof(operators[0]));
}

TokenType identifierParse(char* buffer){
    int n = strlen(buffer);
    bool validFlag = false;
    for (int i=0;i<n;++i){
        if (!isalnum(buffer[i]) && buffer[i]!='_'){
            if (i==0) return TOKEN_UNKNOWN;
            for (int j=n-1;j>=i;--j){
                ungetc(buffer[j],stdin);
            }
            buffer[i] = '\0';
            validFlag = true;
            break;
```

```
        }
    }
    if (isInteger(buffer)) return TOKEN_INT_CONST;
    if (isKeyword(buffer)) return TOKEN_KEYWORD;
    if (isdigit(buffer[0])) return TOKEN_UNKNOWN; //Ensure first digit is alphabet or _
    return TOKEN_IDENTIFIER;
}

TokenType getToken(){
    //Reset buffer
    buffer[0] = '\0';
    bufferLength = 0;

    while (true){
        char c = getchar();
        if (c=='\n'){
            atNewLine = true;
        }

        if (c==EOF){
            if (bufferLength==0) return TOKEN_END;
            ungetc(c,stdin);
            return identifierParse(buffer);
        }

        //Handle compiler directives
        if (atNewLine && c=='#'){
            if (bufferLength!=0){
                ungetc(c,stdin);
                return identifierParse(buffer);
            }
```

```
    buffer[bufferLength++] = c;
    while (true) {
        c = getchar();
        if (c==EOF) {
            if (bufferLength==0) return TOKEN_END;
            ungetc(c,stdin);
            return identifierParse(buffer);
        }
        if (c=='\n') break;
        buffer[bufferLength++] = c;
    }
    buffer[bufferLength] = '\0';
    return TOKEN_DIRECTIVE;
}

// Handle punctuators
if (isPunctuator(c)){
    if (bufferLength==0){
        buffer[0] = c;
        buffer[1] = '\0';
        return TOKEN_PUNCTUATOR;
    }
    ungetc(c,stdin);
    return identifierParse(buffer);
}

// Handle white space
if (isspace(c)){
    if (bufferLength!=0) {
        return identifierParse(buffer);
```

```
    }
    continue;
}

// Handle comments
if (c=='/'){
    char n = getchar();
    bool commentFlag = true;
    if (n=='/'){
        while (commentFlag){
            c = getchar();
            if (c==EOF) break;
            if (c=='\n') commentFlag = false;
        }
        continue;
    } else if (n=='*'){
        while (commentFlag) {
            c = n;
            n = getchar();
            if (n==EOF) break;
            if (c=='*' && n=='/') commentFlag = false;
        }
        continue;
    } else {
        ungetc(n,stdin);
    }
    if (n==EOF){
        return TOKEN_END;
    }
    if (!commentFlag) continue;
}
```

```
        buffer[bufferLength++] = c;
        buffer[bufferLength] = '\0';

        if (isOperator(buffer)){
            while (bufferLength<3){
                buffer[bufferLength++] = getchar();
                buffer[bufferLength] = '\0';
            }
            while (bufferLength>1 && !isOperator(buffer)){
                ungetc(buffer[--bufferLength],stdin);
                buffer[bufferLength] = '\0';
            }
            return TOKEN_OPERATOR;
        }

        if (c==""){
            while ((c=getchar())!=""){
                if (c==EOF){
                    ungetc(c,stdin);
                    return identifierParse(buffer);
                }
                buffer[bufferLength++] = c;
            }
            buffer[bufferLength] = "";
            buffer[++bufferLength] = '\0';
            return TOKEN_STRING_LITERAL;
        }
    }

}
```

```
int main(){
    TokenType currentToken;
    while ((currentToken=getToken())!=TOKEN_END){
        printf("<%s,%s>\n",TokenTypeNames[currentToken],buffer);
    }
}
```

OUTPUT:

Input.txt:

```
#include <stdio.h>

/* Multi
Line Comment*/
int main() {
    //Single line comment
    int arr[2] = {1, 2};
    int a,b,c;
    int *p = &a;
    c = a + b;
    a++;
    c += a;
    c = (a && b);
    return 0;
}
```

output:

```
<Compiler Directive,#include <stdio.h>>
<Keyword,int>
<Identifier,main>
<Punctuator,(>
```

<Punctuator,)>

<Punctuator,{>

<Keyword,int>

<Identifier,arr>

<Punctuator,[>

<Integer Constant,2>

<Punctuator,]>

<Operator,=>

<Punctuator,{>

<Integer Constant,1>

<Punctuator,,>

<Integer Constant,2>

<Punctuator,}>

<Punctuator,;>

<Keyword,int>

<Identifier,a>

<Punctuator,,>

<Identifier,b>

<Punctuator,,>

<Identifier,c>

<Punctuator,;>

<Keyword,int>

<Operator,*>

<Identifier,p>

<Operator,=>

<Operator,&>

<Identifier,a>

<Punctuator,;>

<Identifier,c>

<Operator,=>

<Identifier,a>

<Operator,+>

<Identifier,b>

<Punctuator,;>

<Identifier,a>

<Operator,++>

<Punctuator,;>

<Identifier,c>

<Operator,+=>

<Identifier,a>

<Punctuator,;>

<Identifier,c>

<Operator,=>

<Punctuator,(>

<Identifier,a>

<Operator,&&>

<Identifier,b>

<Punctuator,)>

<Punctuator,;>

<Keyword,return>

<Integer Constant,0>

<Punctuator,;>

<Punctuator,}>