1. IsKeyword Metodu

```c
int isKeyword(const char *identifier) {
    const char *keywords[] = {
            "int", "text", "is", "loop", "times", "read", "write", "newLine"
    };
    int num_keywords = sizeof(keywords) / sizeof(char *);

    for (int i = 0; i < num_keywords; i++) {
        if (strcmp(identifier, keywords[i]) == 0) {
            return 1;
        }
    }

    return 0;
}
```

Bu fonksiyon, verilen bir dizenin (string) bir anahtar kelime olup olmadığını kontrol eder. Önceden tanımlanmış bir anahtar kelime listesi içinde verilen dizeyi arar. Eğer dize listede bulunursa, fonksiyon 1 değerini döndürür; bulunmazsa 0 değerini döndürür. Bu şekilde, verilen dize bir anahtar kelime mi yoksa normal bir tanımlayıcı mı olduğu belirlenir.

2-Identifiers Metodu

Bu kod, bir dosyadan karakter karakter okuyarak bir tanımlayıcıyı (identifier)

tanımlamak için kullanılır.

```
void Identifiers(FILE *sta, FILE *lex, char char_) {
    int count = 0;
    char identifier[MAX_IDENTIFIER_SIZE + 1] = "";

    while (isalnum(char_) || char_ == '\0') {
        if (count < MAX_IDENTIFIER_SIZE) {
            identifier[count] = tolower(char_);
        }
        count++;
        char_ = fgetc(sta);
    }

    ungetc(char_, sta); // put back the non-alphanumeric character

    if (count > MAX_IDENTIFIER_SIZE) {
        fprintf(lex, "Error: Identifier maximum limit of 30 characters\n", identifier);
    } else if (isKeyword(identifier)) {
        fprintf(lex, "Token: Keyword(%s)\n", identifier);
    } else {
        fprintf(lex, "Token: Identifier(%s)\n", identifier);
    }
}
```

3-IntegerConstants Metodu

```c
void IntConsts(FILE *sta, FILE *lex, char char_) {
    int count = 0;
    char integer_const[MAX_INT_SIZE + 1] = "";

    while (isdigit(char_)) {
        if (count < MAX_INT_SIZE) {
            integer_const[count] = char_;
        }
        count++;
        char_ = fgetc(sta);
    }

    ungetc(char_, sta); // put back the non-digit character

    if (count > MAX_INT_SIZE) {
        fprintf(lex, "Error: Integer constant maximum limit of 10 digits\n", integer_const);
    } else {
        fprintf(lex, "Token: IntConst(%s)\n", integer_const);
    }
}
```

Bu kod, bir dosyadan karakter karakter okuyarak bir tamsayı sabitini (integer constant) tanımlar. İşleyiş şu adımları izler:

1. Bir sayaç (**`count`**) ve bir karakter dizisi (**`integer_const`**) tanımlanır.
2. Karakter bir rakam olduğu sürece döngü devam eder.
3. Karakterler tamsayı sabiti dizisine eklenir ve sayacı artırılır.
4. Eğer tamsayı sabitinin boyutu sınırı aşılırsa döngüden çıkılır.
5. Döngü, bir sonraki karakteri alarak devam eder.

Sonuç olarak, dosyadan bir tamsayı sabiti okunur ve **`integer_const`** dizisinde saklanır.

4-Comments Metodu

```
void Comments(FILE *sta, FILE *lex, char char_) {
    char next_char_ = fgetc(sta);
    if (next_char_ == '*') {
        do {
            while ((char_ = fgetc(sta)) != '*') {
                if (char_ == EOF) {
                    fprintf(lex, "Error: Comment doesn't terminate before end of file\n");
                    return;
                }
            }
            char_ = fgetc(sta);
            if (char_ == EOF) {
                fprintf(lex, "Error: Comment doesn't terminate before end of file\n");
                return;
            }
        } while (char_ != '/');
    } else {
        ungetc(next_char_, sta);
    }
}
```

Bu fonksiyon, bir dosyadan yorumları işler. Eğer bir yorum `/*` ile başlıyorsa, `*/` ile bitişini arar ve yorum bloğunu atlar. Eğer dosya sonu veya yorum bloğunun sonu (*/ karakteri) dosya sonunda bulunamazsa, bir hata mesajı yazdırır.

5-Operators Metodu

```
void Operators(FILE *sta, FILE *lex, char char_) {
    char operator[3] = "";
    operator[0] = char_;
    operator[1] = fgetc(sta);

    if (operator[1] != '+' && operator[1] != '=' && operator[1] != '-') {
        ungetc(operator[1], sta); // put back the non-operator character
        operator[1] = '\0'; // null
    }

    fprintf(lex, "Token: Operator(%s)\n", operator);
}
```

Bu fonksiyon, bir dosyadan operatörleri işler. İlk iki karakteri alır ve birleşik bir operatör olup olmadığını kontrol eder. Eğer birleşik bir operatörse, bunu yazdırır, değilse yalnızca ilk karakteri yazdırır.

6-StringConstant Metodu

```
void StringConsts(FILE *sta, FILE *lex, char char_) {
    char string_const[9999] = "";
    int count = 0;

    while ((char_ = fgetc(sta)) != '"') {
        if (char_ == EOF) {
            fprintf(lex, "Error: String constant doesn't terminate before end of file.\n");
            return;
        }
        string_const[count++] = char_;
    }

    string_const[count] = '\0'; // null
    fprintf(lex, "Token: StringConst(%s)\n", string_const);
}
```

Bu fonksiyon, bir dosyadan karakter karakter okuyarak bir string sabitini (string constant) tanımlar. Çift tırnak ("") karakterlerini bulana kadar karakterleri diziye ekler. Eğer dosya sonuna ulaşılırsa veya çift tırnak karakteri bulunamazsa bir hata mesajı yazdırılır.

7-Comma ve EndOfLine Metodu

```
void Comma(FILE *lex, char char_) {
    const char *token_ = "Comma";
    fprintf(lex, "Token: %s\n", token_);
}



void EndOfLine(FILE *lex, char char_) {
    if (char_ == ';') {
        fprintf(lex, "Token: EndOfLine\n");
    }
}
```

Bu kod, virgül ve noktalı virgül karakterlerini işleyen iki ayrı fonksiyondan oluşuyor:

1. Comma fonksiyonu, bir virgül karakteri bulunduğunda, "Comma" tokenini `lex` dosyasına yazdırır.

2. `EndOfLine` fonksiyonu, bir noktalı virgül karakteri bulunduğunda, "EndOfLine" tokenini `lex` dosyasına yazdırır.

Her iki fonksiyon da karakterin türüne bağlı olarak uygun bir token oluşturur ve bunu `lex` dosyasına yazar.

8-Brackets metodu

```
void Brackets(FILE *lex, char char_) {
    const char *token_;
    switch (char_) {
        case '{':
            token_ = "LeftCurlyBracket";
            break;
        case '}':
            token_ = "RightCurlyBracket";
            break;
        default:
            return;
    }
    fprintf(lex, "Token: %s\n", token_);
}
```

Bu fonksiyon, verilen bir karakterin parantez olup olmadığını kontrol eder. Eğer karakter bir açılış veya kapanış süslü parantez ise, buna uygun bir token oluşturur ve bu tokeni yazdırır. Aksi halde, fonksiyon işlemi sonlandırır.

Dosya Açma ve Dosya Hatası Verme

```c
int main() {
    char char_;

    FILE *sta= fopen("code.sta", "r");
    FILE *lex = fopen("code.lex", "w");

    if (!lex) {
        printf("Error: Cannot open output file code.lex\n");
        return 1;
    }
    if (!sta) {
        printf("Error: Cannot open source file code.sta\n");
        return 1;
    }
```

ÇIKTI ÖRNEKLERİ

Code.sta

```
1  int a=5
2  b="elma"+"armut"
3  int c={}
4  d is "ege universitesi"
5  write "Merhaba"
6  e=15-6
```

Code.lex

```
Token: Keyword(int)
Token: Identifier(a)
Token: IntConst(5)
Token: Identifier(b)
Token: StringConst(elma)
Token: Operator(+)
Token: StringConst(armut)
Token: Keyword(int)
Token: Identifier(c)
Token: LeftCurlyBracket
Token: RightCurlyBracket
Token: Identifier(d)
Token: Keyword(is)
Token: StringConst(ege universitesi)
Token: Keyword(write)
Token: StringConst(Merhaba)
Token: Identifier(e)
Token: IntConst(15)
Token: Operator(-)
Token: IntConst(6)
```