دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش فاز اول پروژه‌ی درس طراحی کامپایلر

تحلیلگر لغوی – Lexical Analyzer

مهرآذین مرزوق – ۴۰۰۳۶۱۳۰۵۵

فهرست

منطق پروژه

در این پروژه، ابتدا از کاربر نام فایلی که کد در آن وجود داره گرفته می‌شود. این‌کار تا زمانی ادامه پیدا می‌کند که کاربر عبارت END را وارد کند.

سپس محتویات فایل مذکور، در متغیری به‌نام program ریخته می‌شود تا توابع بتوانند از آنها استفاده کنند.

در مرحله بعدی، دو فایل مربوط به خروجی با whitespace و بدون whitespace ایجاد می‌شود.

برای شروع پروسه‌ی تحلیل لغوی، متغیری به‌نام i درنظر گرفته می‌شود. این متغیر در حقیقت index بخشی از program است که می‌بایست تحلیل لغوی شود. برای انجام تحلیل لغوی، عدد i را به توابع مربوط به هر توکن می‌دهیم تا تابع مربوطه(که با استفاده از دیاگرام گذر هر توکن به وجود آمده است)، آن بخش از program را قبول یا رد کند.

در صورتی که تابع، عبارتی را بپذیرد، توکن مربوطه و index مربوطه (که حالا می‌بایست i را مساوی آن قرار دهیم) را باز می‌گرداند.

در کد زیر، اولویت‌های تابع مربوط به هر توکن با استفاده از if و elif پیاده‌سازی شده است. در هر بخش، در صورتی که توکن با اولویت بیشتر، خالی نباشد، وارد فایل‌های ذکرشده می‌شود.

```
file_name = " "
while file_name != "END":
    print("Please enter the file name")
    file_name = input()
    if file_name != "END":
        with open(file_name, 'r') as file:
            program = file.read()
        with open(f'{file_name}_output.txt', 'w') as f1:
            pass
        with open(f'{file_name}_output_no_whitespace.txt', 'w') as f2:
            pass

        i = 0
        while i != program.__len__():
            A, a = get_ids_or_keywords(i)
            B, b = get_notations(i)
            C, c = get_comments(i)
            D, d = get_numbers(i)
            E, e = get_literals(i)
            F, f = get_operators(i)
            G, g = get_whitespace(i)

            if A is not None:
                write_in_files(A)
                i = a
            elif B is not None:
                write_in_files(B)
```

```
            i = b
        elif C is not None:
            write_in_files(C)
            i = c
        elif D is not None:
            write_in_files(D)
            i = d
        elif E is not None:
            write_in_files(E)
            i = e
        elif F is not None:
            write_in_files(F)
            i = f
        elif G is not None:
            with open(f'{file_name}_output.txt', 'a+') as f1:
                f1.write(f'{i}: {G.display()}\n')
            i = g
        else:
            i += 1

    f1.close()
    f2.close()
    print("The output is ready")
```

توکن

کلاس Token برای تولید توکن‌های تحلیلگر لغوی نوشته شده است.

```
class Token:
    def __init__(self, lexeme, token):
        self.lexeme = lexeme
        self.token = token

    def display(self):
        return f'{self.lexeme} -> {self.token}'
```

ترمینال‌ها

ترمینال‌های مورد استفاده در کدها به شرح زیر می‌باشد.

```
KW = {"bool", "break", "char", "continue", "else", "false", "for", "if", "int", "print", "return", "true"}
digit = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
hex_digit = digit.union({"A", "B", "C", "D", "E", "F", "a", "b", "c", "d", "e", "f"})
digit_but_0 = {"1", "2", "3", "4", "5", "6", "7", "8", "9"}
WS = {" ", "\t", "\n"}
letter_ = set()
for i in range(65, 91):
    letter_.add(chr(i))
for i in range(97, 123):
    letter_.add(chr(i))
letter_.add(chr(95))
letter_digit = letter_.union(digit)
```

نوشتن در فایل‌ها

تابع زیر برای نوشتن در فایلهای خروجی با whitespace و بدون whitescpace استفاده می‌شود.

```
def write_in_files(X: Token):
    with open(f'{file_name}_output.txt', 'a+') as x:
        x.write(f'{i}: {X.display()}\n')
    with open(f'{file_name}_output_no_whitespace.txt', 'a+') as x:
        x.write(f'{i}: {X.display()}\n')
```

کلمات کلیدی

کلمات کلیدی زبان با حروف کوچک نوشته می شوند که در زیر ذکر شده اند.

bool break char continue else false for if int print return true

برای اینکه کلمات کلیدی به عنوان شناسه، شناخته نشوند، پیش از اعلام یک عبارت به عنوان شناسه، آن عبارت را با لیستی از این کلمات کلیدی مقایسه می‌کنیم و در صورتی که این مقایسه به تشابه انجامید، عبارت را به عنوان کلمه کلیدی عنوان می‌کنیم. کد این قسمت در بخش شناسه نشان داده می‌شود.

شناسه‌ها

یک شناسه نامی برای یک موجودیت در یک زبان برنامه نویسی است. دو موجودیت در این زبان برنامه نویسی عبارتند از متغیر و تابع. شناسه‌ها با یک حرف یا علامت زیرخط (_) آغاز می‌شوند و می‌توانند حاوی ارقام ، حروف و علامت های زیرخط باشند. شناسه ها نمی توانند برابر با هیچ یک از کلمات کلیدی باشند

عبارت منظم

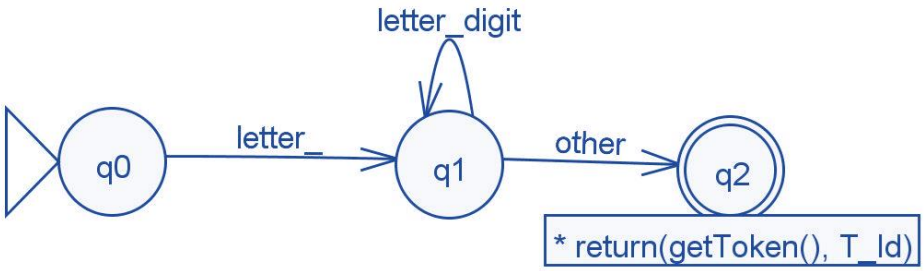
letter_ → [A-Za-z_]

digit → [0-9]

letter_digit → letter_ | digit

id → letter_ (letter_digit)*

دیاگرام گذار



کد

برای این‌که هیچ کلمه‌ی کلیدی‌ای به عنوان شناسه معرفی نشود باید در نهایت همه شناسه‌ها را با لیست کلمات کلیدی مقایسه کرد و در صورت شباهت، عبارت را به عنوان کلمه کلیدی معرفی کنیم.

در این کد، این شرط در state == 2 بررسی شده‌است. مابقی کد بر اساس دیاگرام گذار زده شده‌است.

```
def get_ids_or_keywords(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
        if state == 0:
            if letter_.__contains__(c):
                state = 1
            else:
                return None, None
        elif state == 1:
            if letter_digit.__contains__(c):
                state = 1
            else:
                state = 2
        elif state == 2:
            index -= 1
            token = program[m:index]
            if KW.__contains__(token):
                return Token(token, f'T_{token.capitalize()}'), index
            else:
```

```
            return Token(token, "T_Id"), index
        index += 1
    return None, None
```

علامت‌های نشانه‌گذاری

–علامت های آ کولاد باز } و آ کولاد بسته { در تعریف بلوک ها استفاده می شوند.

–علامت های پرانتز باز) و پرانتز بسته (در تعریف توابع، فراخوانی توابع، و عبارت های محاسباتی استفاده می شوند.

–علامت های کروشه باز] و کروشه بسته [برای تعریف آرایه های استفاده می شوند.

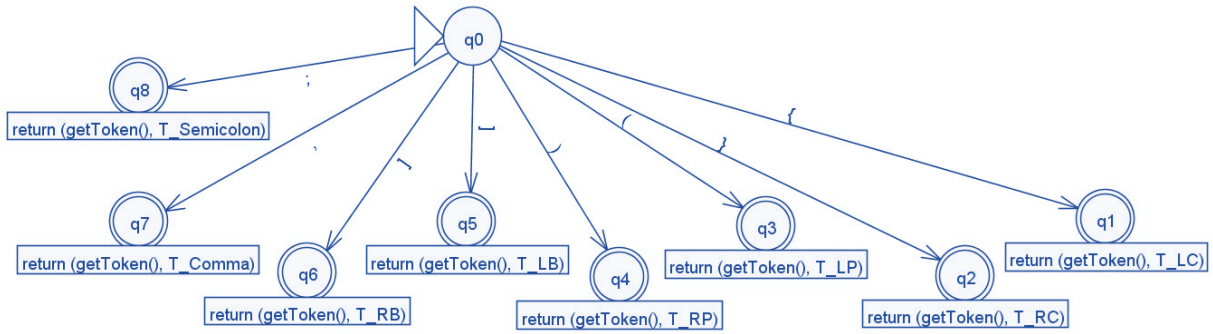
–علامت ویرگول , برای جدا کردن ورودی های تابع از یکدیگر در تعریف و فراخوانی تابع استفاده می شود.

–علامت نقطه ویرگول ; در پایان تعریف متغیرها، دستورات محاسبه ای و فراخوانی توابع، و همچنین در تعریف حلقه ها استفاده می شوند.

عبارت منظم

Notations -> { | } | (|) | [|] | , | ;

دیاگرام گذار



کد

کد زیر بر اساس دیاگرام گذار بالا زده شده‌است.

```
def get_notations(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
```

```
    if state == 0:
        if c == "{":
            state = 1
        elif c == "}":
            state = 2
        elif c == "(":
            state = 3
        elif c == ")":
            state = 4
        elif c == "[":
            state = 5
        elif c == "]":
            state = 6
        elif c == ",":
            state = 7
        elif c == ";":
            state = 8
        else:
            return None, None

    elif state == 1:
        return Token("{", "T_LC"), index
    elif state == 2:
        return Token("}", "T_RC"), index
    elif state == 3:
        return Token("(", "T_LP"), index
    elif state == 4:
        return Token(")", "T_RP"), index
    elif state == 5:
        return Token("[", "T_LB"), index
    elif state == 6:
        return Token("]", "T_RB"), index
    elif state == 7:
        return Token(",", "T_Comma"), index
    elif state == 8:
        return Token(";", "T_Semicolon"), index
    else:
        return None, None
    index += 1
return None, None
```

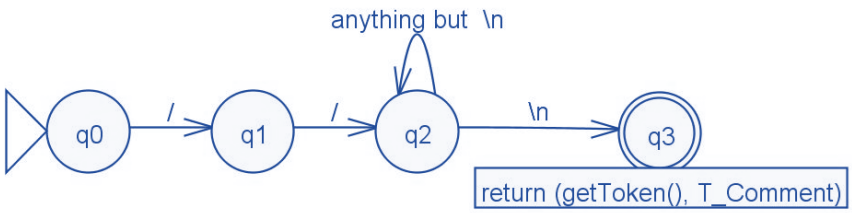
توضیحات

توضیحات با دو علامت اسلش یا خط اریب // آغاز می شوند و با کاراکتر پایان خط معادل کد اسکی ۱۰ یا \n پایان می یابند. تحلیل گر لغوی توضیحات را به تحلیل گر نحوی ارسال نمی کند، اما پس از تحلیل لغات لیست همهٔ توکن ها را چاپ می کند.

عبارت منظم

comments -> // (anything but \n)* (\n)

دیاگرام گذر



کد

کد زیر بر اساس دیاگرام گذار بالا زده شده‌است.

```
def get_comments(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
        if state == 0:
            if c == "/":
                state = 1
            else:
                return None, None
        elif state == 1:
            if c == "/":
                state = 2
```

```
        else:
            return None, None
    elif state == 2:
        if c != "\n":
            state = 2
        else:
            state = 3
    elif state == 3:
        index -= 1
        token = program[m:index]
        return Token(token, "T_Comment"), index
    else:
        return None, None
    index += 1
return None, None
```

مقادیر عددی

مقادیر عددی می توانند در مبنای ده (دهدهی یا دسیمال) یا در مبنای شانزده (شانزده شانزدهی یا هگزادسیمال) باشند. یک عدد دهدهی می تواند مثبت یا منفی باشد که در صورت منفی بودن با علامت – آغاز می شوند. اعداد هگزادسیمال با دو کاراکتر 0x آغاز می شوند.

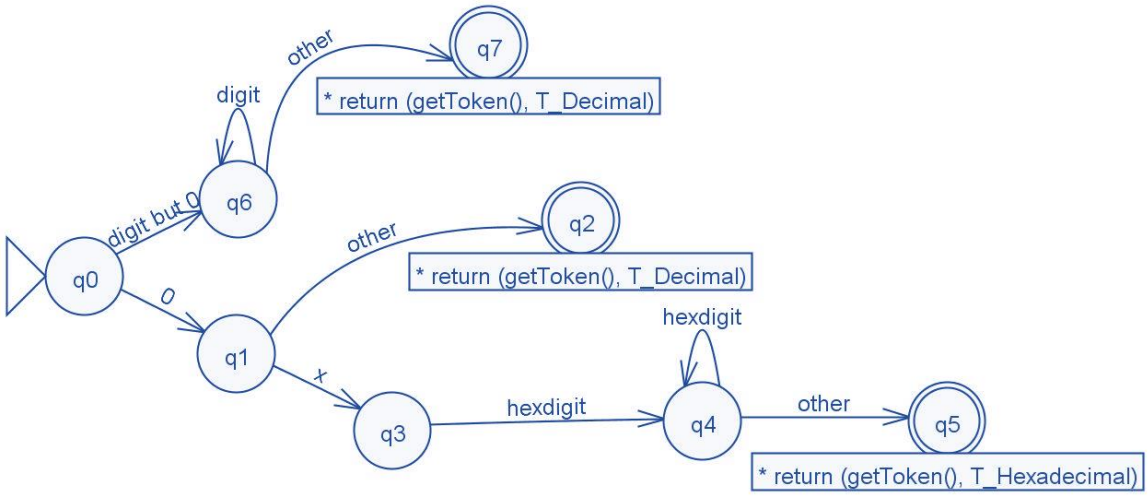
عبارت منظم

number → decimal | 0 (hexadecimal?)

hexadecimal → x (hex_digit)+

decimal → (digit but 0) (digit)*

دیاگرام گذر



کد

کد زیر بر اساس دیاگرام گذار بالا زده شده‌است.

```
def get_numbers(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
        if state == 0:
            if digit_but_0.__contains__(c):
                state = 6
```

```
        elif c == "0":
            state = 1
        else:
            return None, None
    elif state == 1:
        if c == "X" or c == "x":
            state = 3
        else:
            state = 2
    elif state == 2:
        index -= 1
        token = Token(program[m:index], "T_Decimal")
        return token, index
    elif state == 3:
        if hex_digit.__contains__(c):
            state = 4
        else:
            return None, None
    elif state == 4:
        if hex_digit.__contains__(c):
            state = 4
        else:
            state = 5
    elif state == 5:
        index -= 1
        token = Token(program[m:index], "T_Hexadecimal")
        return token, index
    elif state == 6:
        if digit.__contains__(c):
            state = 6
        else:
            state = 7
    elif state == 7:
        index -= 1
        token = Token(program[m:index], "T_Decimal")
        return token, index
    else:
        return None, None
    index += 1
return None, None
```

کاراکترها و رشته‌های ثابت

یک کاراکتر ثابت، یکی از حروف الفبای اسکی است. یک کاراکتر ثابت بین دو علامت آپوستروف ’ ‘ قرار می گیرد. برای نشان دادن علامت آپوستروف در یک کاراکتر ثابت از ’\’ و برای نشان دادن کاراکتر خط اریب وارون (بک اسلش) از ’\\‘ استفاده می‌شود.

یک رشتهٔ ثابت را با دنباله ای از کاراکترها که در بین دو علامت نقل قول " " قرار گرفته اند، نشان می دهیم. برای نشان دادن علامت نقل قول در یک رشته ثابت از "\ استفاده می شود.

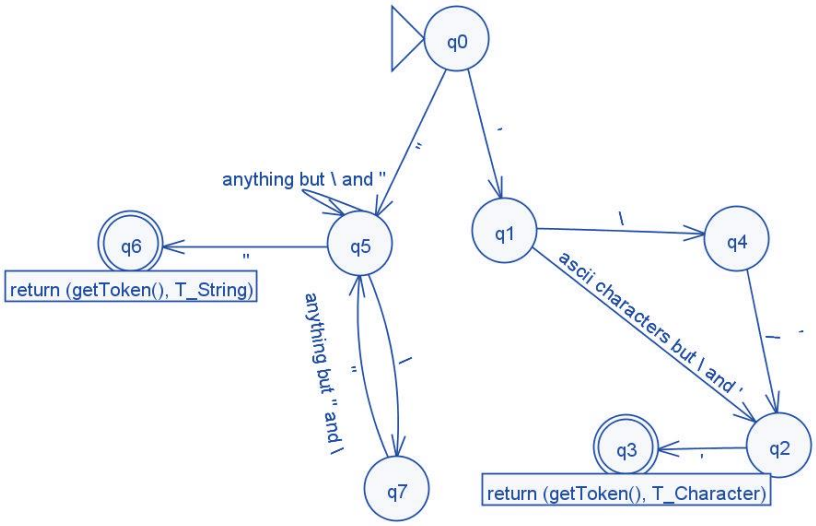
عبارت منظم

litaral → character | string

character → ’ (anything but \ and ‘ | \\ | \’) \’

string → “ (anything but \ and “ | \” | \(anything but \ and “))* “

دیاگرام گذار



کد

```
def get_literals(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
```

```
    if state == 0:
        if c == "'":
            state = 1
        elif c == "\"":
            state = 5
        else:
            return None, None
    elif state == 1:
        if c == "\\":
            state = 4
        elif c.isascii() and c != "'":
            state = 2
        else:
            return None, None
    elif state == 2:
        if c == "'":
            state = 3
        else:
            return None, None
    elif state == 3:
        token = Token(program[m:index], "T_Character")
        return token, index
    elif state == 4:
        if c == "'" or c == "\\":
            state = 2
        else:
            return None, None
    elif state == 5:
        if c.isascii() and c != "\"" and c != "\\":
            state = 5
        elif c == "\"":
            state = 6
        elif c == "\\":
            state = 7
        else:
            return None, None
    elif state == 6:
        token = Token(program[m:index], "T_String")
        return token, index
    elif state == 7:
        if c.isascii():
            state = 5
        else:
            return None, None
    else:
        return None, None
    index += 1
return None, None
```

عملگرها

عملگرهای حسابی مانند + ، - ، * ، / ، % استفاده می شود.

عملگرهای یگانی + و - برای تعیین مثبت و منفی بودن اعداد به کار می روند. عملگرهای یگانی + و - بالاترین اولویت را دارند و پس از آنها * ، / ، % هم اولویت بوده و در درجهٔ دوم اولویت قرار دارند و در نهایت + و - اولویت سوم قرار می گیرند.

عملگرهای رابطه‌ای < ، =< ، > ، >= ، == ، =! برای مقایسه دو مقدار به کار می روند.

عملگرهای منطقی عطف && و فصل || و نقیض ! نیز در عبارات منطقی به کار می روند.

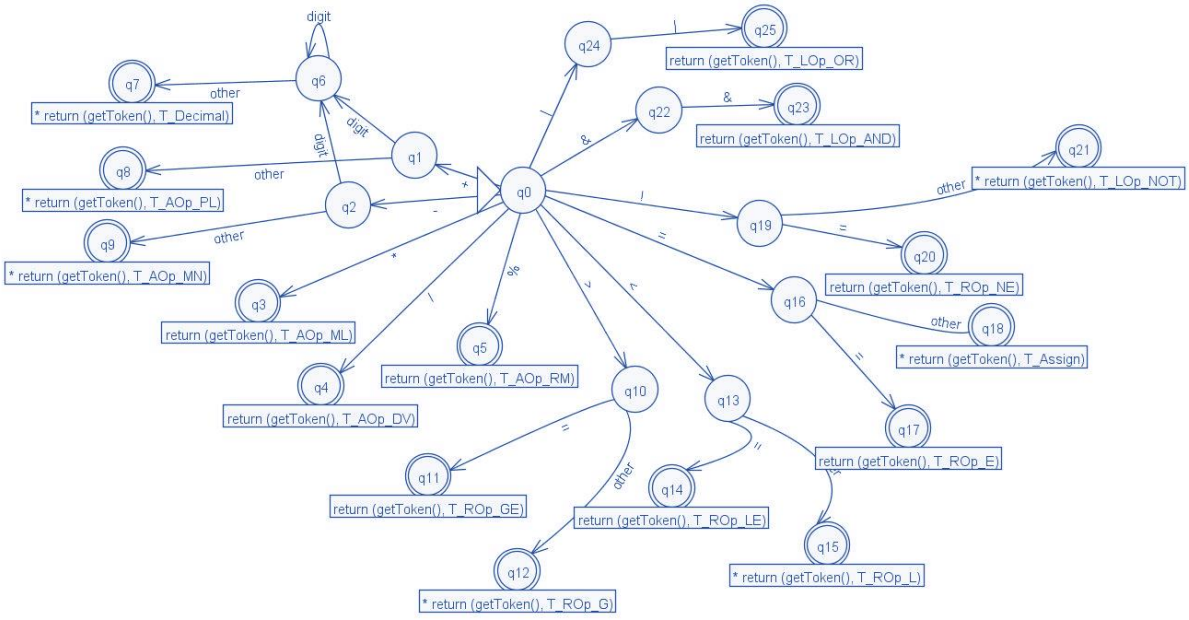
عملگر = هم به معنای اختصاص است.

نکته: از آنجایی که اولویت عملگرهای یگانی + و - از عملگرهای حسابی + و - بیشتر است، می‌بایست بخش اعداد دسیمال را وارد این دیاگرام گذار کنیم تا بتوانیم اولویت‌ها را پیاده‌سازی کنیم. همچنین در کد باید این اولویت‌ها را به صورت if های پشت‌سرهم پیاده‌سازی کنیم.

عبارت منظم

operators → +(digits?) | -(digits?) | * | / | % | <(=?) | >(=?) | =(=?) | !(=?) | || | &&

دیاگرام گذار



کد

کد زیر بر اساس دیاگرام گذار بالا زده شده‌است.

```
def get_operators(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
        if state == 0:
            if c == "+":
                state = 1
            elif c == "-":
                state = 2
            elif c == "*":
                state = 3
            elif c == "/":
                state = 4
            elif c == "%":
                state = 5
            elif c == ">":
                state = 10
            elif c == "<":
                state = 13
            elif c == "=":
                state = 16
            elif c == "!":
                state = 19
            elif c == "&":
                state = 22
            elif c == "|":
                state = 24
            else:
                return None, None
        elif state == 1:
            if digit.__contains__(c):
                state = 6
            else:
                state = 8
        elif state == 2:
            if digit.__contains__(c):
                state = 6
            else:
                state = 9
        elif state == 3:
            token = Token(program[m:index], "T_AOp_ML")
            return token, index
        elif state == 4:
            token = Token(program[m:index], "T_AOp_DV")
            return token, index
        elif state == 5:
            token = Token(program[m:index], "T_AOp_RM")
```

```
        return token, index
    elif state == 6:
        if digit.__contains__(c):
            state = 6
        else:
            state = 7
    elif state == 7:
        index -= 1
        token = Token(program[m:index], "T_Decimal")
        return token, index
    elif state == 8:
        index -= 1
        token = Token(program[m:index], "T_AOp_PL")
        return token, index
    elif state == 9:
        index -= 1
        token = Token(program[m:index], "T_AOp_MN")
        return token, index
    elif state == 10:
        if c == "=":
            state = 11
        else:
            state = 12
    elif state == 11:
        token = Token(program[m:index], "T_ROp_GE")
        return token, index
    elif state == 12:
        index -= 1
        token = Token(program[m:index], "T_ROp_G")
        return token, index
    elif state == 13:
        if c == "=":
            state = 14
        else:
            state = 15
    elif state == 14:
        token = Token(program[m:index], "T_ROp_LE")
        return token, index
    elif state == 15:
        index -= 1
        token = Token(program[m:index], "T_ROp_L")
        return token, index
    elif state == 16:
        if c == "=":
            state = 17
        else:
            state = 18
    elif state == 17:
        token = Token(program[m:index], "T_ROp_E")
        return token, index
    elif state == 18:
        index -= 1
        token = Token(program[m:index], "T_Assign")
        return token, index
```

```
        elif state == 19:
            if c == "=":
                state = 20
            else:
                state = 21
        elif state == 20:
            token = Token(program[m:index], "T_ROp_NE")
            return token, index
        elif state == 21:
            index -= 1
            token = Token(program[m:index], "T_LOp_NOT")
            return token, index
        elif state == 22:
            if c == "&":
                state = 23
            else:
                return None, None
        elif state == 23:
            token = Token(program[m:index], "T_LOp_AND")
            return token, index
        elif state == 24:
            if c == "|":
                state = 25
            else:
                return None, None
        elif state == 25:
            token = Token(program[m:index], "T_LOp_OR")
            return token, index
        else:
            return None, None
        index += 1
    return None, None
```

فاصله‌های خالی

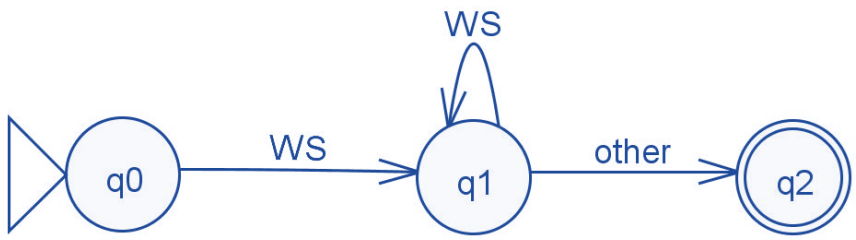
توکن ها توسط یک فاصلهٔ خالی و یا ترکیبی از فاصله های خالی از یکدیگر جدا می شوند. یک فاصله خالی شامل کاراکتر فاصله با کد اسکی ۳۲ ، کاراکتر خط جدید با کد اسکی ۱۰ ، و کاراکتر ستون جدید با کد اسکی ۹ می شود.

عبارت منظم

Whitespace → (WS)+

WS = {“\n”, “\t”, “ ”}

دیاگرام گذار



کد

```
def get_whitespace(m: int):
    index = m
    state = 0
    while index != program.__len__()+1:
        if index < program.__len__():
            c = program[index]
        if state == 0:
            if WS.__contains__(c):
                state = 1
            else:
                return None, None
        elif state == 1:
            if WS.__contains__(c):
                state = 1
            else:
                state = 2
```

```
    elif state == 2:
        index -= 1
        token = Token("whitespace", "T_Whitespace")
        return token, index
    else:
        return None, None
    index += 1
return None, None
```

منابع

اسلایدهای درس طراحی کامپایلر

Compilers: principals, techniques and tools – Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffery D. Ullman