

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش فاز اول پروژه‌ی درس طراحی کامپایلر

تحلیلگر لغوی – Lexical Analyzer

مهرآذین مرزوق – ۴۰۰۳۶۱۳۰۵۵

# 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.

KW = {"bool", "break", "char", "continue", "else", "false", "for", "if", "int", "print", "return", "true"}

# 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 (ـ) 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

A diagram of a diagram

Description automatically generated

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.

## کد

ترمینال‌های مورد استفاده‌ی تابع get\_ids\_or\_keywords :

KW = {"bool", "break", "char", "continue", "else", "false", "for", "if", "int", "print", "return", "true"}  
digit = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}  
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)

تابع اصلی:

def get\_ids\_or\_keywords(m: int):  
 index = m  
 state = 0  
 while 1:  
 if state == 0:  
 if letter\_.\_\_contains\_\_(program[index]):  
 state = 1  
 else:  
 return None  
 elif state == 1:  
 if letter\_digit.\_\_contains\_\_(program[index]):  
 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

# 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

A diagram of a network

Description automatically generated

## کد

def get\_notations(m: int):  
 index = m  
 state = 0  
 while 1:  
 if state == 0:  
 if program[index] == "{":  
 state = 1  
 elif program[index] == "}":  
 state = 2  
 elif program[index] == "(":  
 state = 3  
 elif program[index] == ")":  
 state = 4  
 elif program[index] == "[":  
 state = 5  
 elif program[index] == "]":  
 state = 6  
 elif program[index] == ",":  
 state = 7  
 elif program[index] == ";":  
 state = 8  
  
 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

# 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 همۀ توکن ها را چاپ می کند.

## عبارت منظم

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

## دیاگرام گذر

A diagram of a diagram

Description automatically generated

## کد

def get\_comments(m: int):  
 index = m  
 state = 0  
 while 1:  
 if state == 0:  
 if program[index] == "/":  
 state = 1  
 else:  
 return None, None  
 elif state == 1:  
 if program[index] == "/":  
 state = 2  
 else:  
 return None, None  
 elif state == 2:  
 if program[index] != "\n":  
 state = 2  
 else:  
 state = 3  
 elif state == 3:  
 token = program[m:index]  
 return Token(token, "T\_Comment"), index  
 else:  
 return None, None  
 index += 1

# 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.

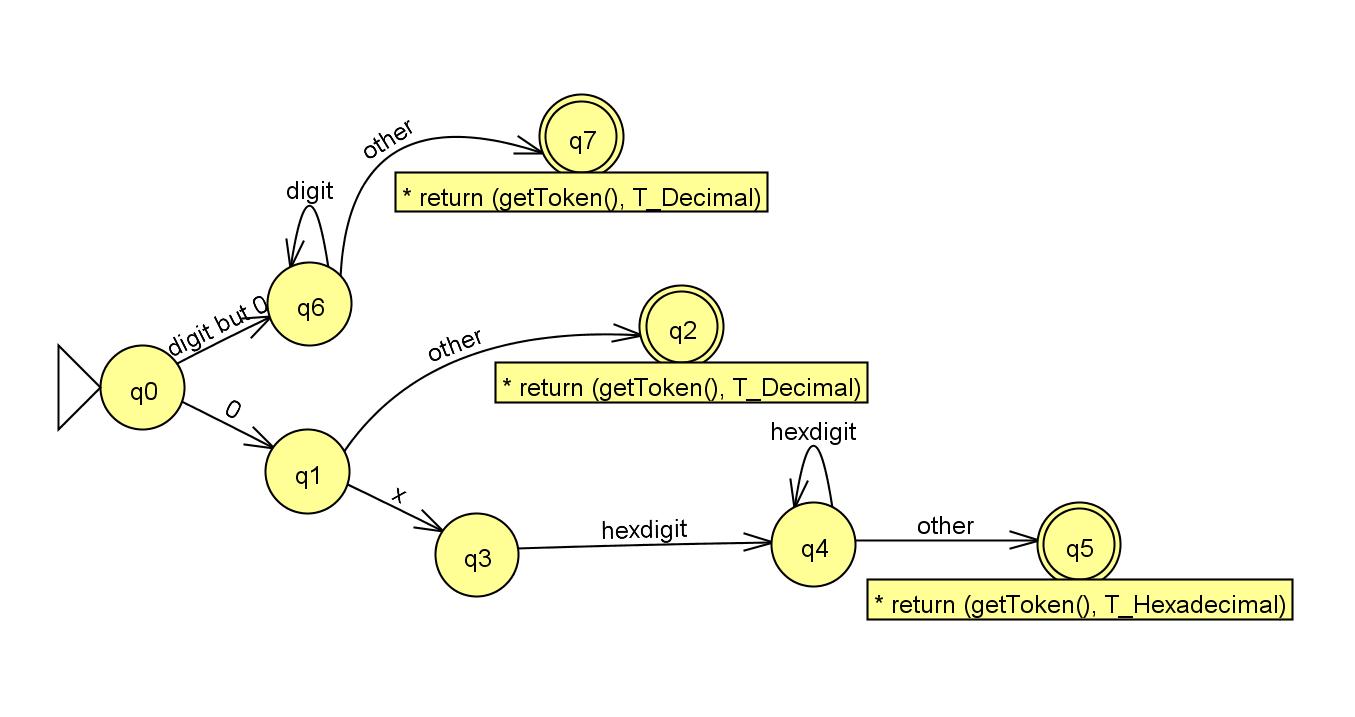
## عبارت منظم

number decimal | 0 (hexadecimal?)

hexadecimal x (hex\_digit)+

decimal (digit but 0) (digit)\*

## دیاگرام گذر



## کد

ترمینال‌های استفاده شده در تابع get\_numbers :

digit = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}  
hex\_digit = digit.union({"A", "B", "C", "D", "E", "F"})  
digit\_but\_0 = {"1", "2", "3", "4", "5", "6", "7", "8", "9"}

تابع اصلی:

def get\_numbers(m: int):  
 index = m  
 state = 0  
 while 1:  
 if state == 0:  
 if digit\_but\_0.\_\_contains\_\_(program[index]):  
 state = 6  
 elif program[index] == "0":  
 state = 1  
 else:  
 return None, None  
 elif state == 1:  
 if program[index] == "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\_\_(program[index]):  
 state = 4  
 else:  
 return None, None  
 elif state == 4:  
 if hex\_digit.\_\_contains\_\_(program[index]):  
 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\_\_(program[index]):  
 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

# 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 رشتۀ 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.

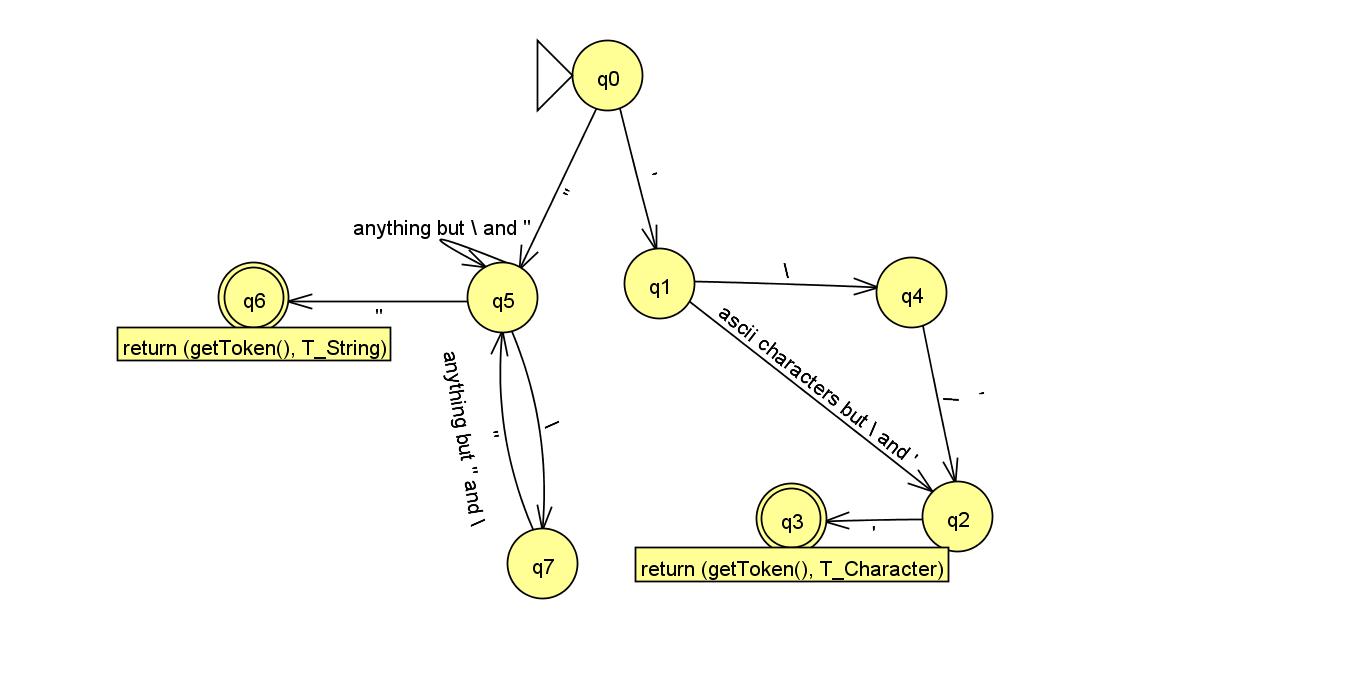
## عبارت منظم

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 1:  
 if state == 0:  
 if program[index] == "'":  
 state = 1  
 elif program[index == "\""]:  
 state = 5  
 else:  
 return None, None  
 elif state == 1:  
 if program[index] == "\\":  
 state = 4  
 elif program[index].isascii() and program[index] != "'":  
 state = 2  
 else:  
 return None  
 elif state == 2:  
 if program[index] == "'":  
 state = 3  
 else:  
 return None  
 elif state == 3:  
 token = Token(program[m:index], "T\_Character")  
 return token, index  
 elif state == 4:  
 if program[index] == "'" or program[index] == "\\":  
 state = 2  
 else:  
 return None, None  
 elif state == 5:  
 if program[index].isascii() and program[index] == "\"" and program[index] == "\\":  
 state = 5  
 elif program[index] == "\"":  
 state = 6  
 elif program[index] == "\\":  
 state = 7  
 else:  
 return None, None  
 elif state == 6:  
 token = Token(program[m:index], "T\_String")  
 return token, index  
 elif state == 7:  
 if program[index].isascii() and program[index] != "\\":  
 state = 5  
 else:  
 return None, None  
 else:  
 return None, None  
 index += 1

# 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 درجۀ 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.

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

A diagram of a network

Description automatically generated

# فاصله‌های خالی

توکن ها توسط یک فاصلۀ 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

A diagram of a diagram of a flowchart

Description automatically generated