

به نام خدا

بهاره کاوسی نژاد – 99431217

تمرین سری سوم درس کامپایلر – شبیهساز میاترم

سوال 1

در این سوال قصد داریم یک زبان ساده برنامه نویسی به نام زبان چالوس-اوتاق را طراحی کنیم. این زبان شامل دستورات زیر است:

1. انواع جملات تخصیصی (Assignment)، شرطی (if)، حلقه (while)، جملات ترکیبی (Compound)
2. تعریف یا declaration نوع متغیرها شامل int و boolean.
3. قوانین لغوی شامل تعریف اعداد، کامنت، عملگرهای محاسباتی، عملگرهای مقایسه‌ای، عملگرهای منطقی و مقادیر true و false.

- تعریف و مقدار دهی متغیرها (این زبان تنها از متغیرهای عدد صحیح و بولین پشتیبانی می‌کند)

```
int x;  
x=10;  
boolean b;  
b=true;  
int y=76;  
boolean c=false;
```

- وجود عملگرهای مقایسه‌ای بزرگتر، مساوی و کوچکتر در زبان برای متغیرهای عدد صحیح. خروجی این عملگرها یک boolean است.
- وجود عملگرهای محاسبه‌گر جمع و تفریق برای اعداد صحیح.
- وجود دستور شرطی تنها به شکل زیر:

```
if(boolean condition) {  
    State;  
}
```

- وجود دستور برای ایجاد حلقه تنها به شکل زیر:

```
while(boolean condition) {
    State;
}
```

- وجود کامنت تنها به صورت زیر:

```
#comment
```

یک گرامر با فرمت g4 برای زبان چالوس-اوتاق طراحی نمایید.

پس از ساخت پروژه، یک فایل جدید با نام Chaloos_Otagh.g4 می‌سازیم و گرامر را تشکیل می‌دهیم:

• قوانین Parser:

- **program**: یک program با یک statement یا بیشتر
- **statement**: انواع مختلف statement یعنی تعریف متغیرها، جملات تخصیصی، شرطی، حلقه و جملات ترکیبی
- **assignment**: مقداردهی متغیرها
- **variableDeclaration**: تعریف متغیرها (نوع و نام)
- **variableType**: نوع متغیرها که می‌تواند int یا boolean باشد.
- **variableName**: تعیین چگونگی نامگذاری متغیرها
- **expression**: انواع مختلف expression شامل محاسباتی، مقایسه‌ای و مقادیر boolean
- **additionExpression**: expression جمع کردن تشکیل شده از expression‌های اولیه (primary)
- **comparisonExpression**: expression مقایسه‌ای تشکیل شده از expression‌های اولیه (primary) و عملگر مقایسه‌ای
- **booleanValue**: مقادیر boolean که می‌توانند true یا false باشند
- **ifStatement**: تعریف یک جمله شرطی با شرط boolean و جمله ترکیبی
- **whileStatement**: تعریف یک حلقه با شرط boolean و جمله ترکیبی
- **compoundStatement**: تعریف یک جمله ترکیبی داخل {} شامل یک یا دو عبارت (statement)

• قوانین Lexer:

- قوانین lexer، token‌هایی که در زبان مورد استفاده قرار می‌گیرند را تعریف می‌کند؛ مانند عملگرهای حسابی (+، -، * و /)، عملگرهای مقایسه‌ای (=، > و <)، پرانتزها، نقطه ویرگول، عملگر تخصیصی، keywordهای integer، مقادیر boolean (true و false) و integer literals.
- همچنین whitespace‌ها و کامنت‌ها در حین parse کردن skip می‌شوند.

```

grammar Chaloos_Otagh;

// Parser rules
program: statement+;

statement: variableDeclaration
         | assignment
         | ifStatement
         | whileStatement
         | compoundStatement;

assignment: variableDeclaration '=' expression ';'
          | variableName '=' expression ';';

variableDeclaration: variableType variableName;

variableType: 'int' | 'boolean';

variableName: (Letter | Digit | '_') (Letter | Digit | '_')*;

expression: additionExpression | comparisonExpression |
booleanValue;

additionExpression: primaryExpression (ADD primaryExpression)*;

comparisonExpression: primaryExpression comparisonOperator
primaryExpression;

primaryExpression: '(' expression ')' | variableName | INT;

comparisonOperator: '>' | '==' | '<';

booleanValue: 'true' | 'false';

ifStatement: 'if' '(' booleanCondition ')' compoundStatement;

whileStatement: 'while' '(' booleanCondition ')'
compoundStatement;

compoundStatement: '{' statement+ '}';

booleanCondition: '(' expression ')';

// Lexer rules
ADD: '+';
SUB: '-';
MUL: '*';
DIV: '/';

EQ: '==';
GT: '>';
LT: '<';

LPAREN: '(';
RPAREN: ')';
LBRACE: '{';
RBRACE: '}';
SEMI: ';';
ASSIGN: '=';
BOOL: 'boolean';
INT_TYPE: 'int';
TRUE: 'true';
FALSE: 'false';

INT: [+-]? [0-9]+;

Letter: [a-zA-Z];
Digit: [0-9];

WS: [ \t\r\n]+ -> skip;
COMMENT: '#' ~[\r\n]* -> skip;

```

سوال 2

با کمک ابزار Antlr و زبان پایتون برنامه‌ای بنویسید که یک رشته حاوی یک فایل دستورات زبان چالوس-اوتاق را در ورودی گرفته، در ادامه تغییراتی را روی آن اعمال کند. این تغییرات بدین شکل است که در ابتدای هر کامنت نام خانوادگی شما و در انتهای هر کامنت شماره دانشجویی شما را نوشته و کامنت بدست آمده را جایگزین کامنت قبلی کند.

#comment


میشود:

#<your_lastname>comment<your_student_id>

این سوال در فایل main.py نوشته شده است.


```
if __name__ == '__main__':  
    # Read the input file  
    input_file = "input.txt"  
    with open(input_file, "r") as file:  
        content = file.readlines()  
  
    last_name = "Kavousi"  
    StudentID = "99431217"  
    # Modify the comments and create the updated content  
    updated_content = []  
    for line in content:  
        if line.startswith("#"):  
            updated_line = "# " + last_name + " " +  
line.lstrip("#").rstrip("\n") + " " + StudentID + "\n"  
            updated_content.append(updated_line)  
        else:  
            updated_content.append(line)  
  
    # Write the updated content to a new file  
    output_file = "output.txt"  
    with open(output_file, "w") as file:  
        file.writelines(updated_content)
```

- ابتدا فایل input.txt را به عنوان ورودی می‌خوانیم:



```
#This is the first comment
int x;
x=10;
boolean b;
b=true;
#This is the second comment
int y=76;
boolean c=false;
#This is the third comment
```

- این فایل حاوی سه خط کامنت است.
- دو متغیر برای ذخیره نام خانوادگی و شماره دانشجویی تعریف می‌کنیم.
- در یک حلقه for، در هر خط فایل بررسی می‌کنیم که آیا خط با علامت # آغاز شده است یا خیر.
- هر خطی که با # آغاز شده باشد به عنوان کامنت شناخته می‌شود و نام خانوادگی به ابتدای آن و شماره دانشجویی به انتهای آن اضافه می‌شود.
- هر خط در انتهای حلقه for به updated_content اضافه می‌شود.
- در انتها، نتیجه را در فایل output.txt ذخیره می‌کنیم.



```
# Kavousi This is the first comment 99431217
int x;
x=10;
boolean b;
b=true;
# Kavousi This is the second comment 99431217
int y=76;
boolean c=false;
# Kavousi This is the third comment 99431217
```

سوال 3

با استفاده از زبان پایتون و ابزار انتلر برنامه ای بنویسید که یک رشته حاوی دستورات زبان چالوس-اوتاق را دریافت کرده و بیشترین عمق دستورات را در خروجی چاپ کند. هر موقع وارد بدنه یک دستور شرطی یا یک حلقه می‌شویم، عمق کد یک درجه افزایش یافته و وقتی از هر کدام خارج می‌شویم عمق کد یک درجه افزایش میابد. به عنوان مثال حداکثر عمق قطعه کد زیر برابر 2 است.

```
if(condition) {  
    State;  
}  
  
while(condition) {  
    if(condition) {  
        State;  
    }  
}
```

این سوال در فایل Q3.py نوشته شده است. همچنین می‌توان در listener نیز آن را پیاده سازی کرد؛ به این صورت که مانند همین فایل Q3.py یک متغیر max_depth و یک متغیر current_depth با مقدار اولیه 0 داریم و در هر enterIfStatement یا enterWhileStatement، یک واحد به current_depth اضافه شده و max_depth به روز رسانی می‌شود و در هر exitIfStatement یا exitWhileStatement، یک واحد از current_depth کم می‌شود. همچنین می‌توان با استفاده از attribute در گرامر این سوال را پیاده سازی کرد.

```
def count_max_depth(code):  
    max_depth = 0  
    current_depth = 0  
  
    for line in code:  
        line = line.strip()  
  
        if line.startswith("if") or line.startswith("while"):  
            current_depth += 1  
            max_depth = max(max_depth, current_depth)  
  
        if line.endswith("}"):  
            current_depth -= 1  
  
    return max_depth  
  
if __name__ == '__main__':  
    # Read the input file  
    input_file = "input3.txt"  
    with open(input_file, "r") as file:  
        content = file.readlines()  
  
    max_depth = count_max_depth(content)  
    print("Maximum Depth:", max_depth)
```

- در تابع `count_max_depth`، به ازای هر خط ورودی بررسی می‌کنیم که آیا خط با `if` یا `while` آغاز شده است یا خیر. اگر این شرط برقرار باشد، به عمق کنونی (`current_depth`)، یک واحد اضافه می‌شود و عمیق بیشینه (`max_depth`) آپدیت می‌شود.
- در قسمت `main` نیز فایل ورودی را می‌خوانیم و بعد از فراخوانی تابع `count_max_depth` خروجی را چاپ می‌کنیم.

ورودی اول:

```
#This is the first comment
int x;
x=10;
boolean b;
b=true;
#This is the second comment
int y=76;
boolean c=false;
#This is the third comment
```

خروجی اول:

```
Maximum Depth: 0
```

```
Process finished with exit code 0
```

ورودی دوم:

```
if(condition) {  
    State;  
}  
while(condition) {  
    if (condition) {  
        State;  
    }  
}
```

خروجی دوم:

```
Maximum Depth: 2
```

```
Process finished with exit code 0
```


ورودی سوم:

```
if(condition) {  
    State;  
}  
while(condition) {  
    if (condition) {  
        while(condition) {  
            }  
        }  
    }  
}
```

خروجی سوم:

```
Maximum Depth: 3
```

```
Process finished with exit code 0
```