上 海 交 通 大 学 试 卷

（ 2024 至 2025 学年 第 1 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称_____编译原理与技术_____成绩 _____

Problem 1: Lexical Analysis (30 points)

1.

(1.1)

(1.2)

(1.3)

2.

我承诺，我将严格遵守考试纪律。

承诺人： _____

题号									
得分									
批阅人(流水阅卷教师签名处)									

3.

Problem 2: Grammar and AST (50 points)

1.

2.

	a	=	;	lambda	{	}	+	$	S′	S	EL	E	T	F
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														

3.

Step	Stack	Action
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		
40		
41		
42		

43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		

Problem 3: Semantic Analysis (20 points)

1.

2.

Problem 1: Lexical Analysis (30 points)

```
match request with
| LocalDate ->
   "2024-11-22"
| LocalDateTime ->
   "2024-11-22T14:00:00"
| OffsetDateTime ->
   "2024-11-22T14:00:00+08:00";;  // "2024-11-22T14:00:00Z" is also ok
```

```
localDateStr                                        (1.1)
localDateTimeStr                                    (1.2)
offsetDateTimeStr                                   (1.3)
matchType
["LocalDate"|"LocalDateTime"|"OffsetDateTime"]
SPACE                                               [\t]
ENTER                                               [\n]
%x INIT MatchContent MatchRequest MatchType MatchArrow MatchStr
%%
/** Note: We use macro to replace some function like ADJ -> adjust(),
  *
  */
<INIT>"match"                    {ADJ; cout << "begin match:” << endl;
                                  BEGIN(MatchRequest);}
<INIT MatchContent MatchRequest MatchType MatchArrow MatchStr>{SPACE}
                                 {ADJ;}
<MatchStr>{ENTER}                {ADJ;}
<INIT MatchContent>{ENTER}
                                 {ADJ;
                                  cout << "next:" << endl }
<INIT>.                          {ADJ; cout << "error init" << endl;
                                  ERROR;}
```

```
<MatchRequest>"request with"    {ADJ; cout << "find match request" << endl;
                                 BEGIN(MatchContent);}
<MatchRequest>.                 {ADJ; cout << "invalid match request" << endl;
                                 ERROR;}
<MatchContent>"|"               {ADJ;
                                 BEGIN(MatchType);}
<MatchContent>";;"              {ADJ; cout << "end match" << endl;
                                 Return;}
<MatchContent>.                 {ADJ; cout << "error in |" << endl;
                                 ERROR;}
<MatchType>{matchType}          {ADJ;
                                 cout << "begin find arrow" << endl;
                                 BEGIN(MatchArrow);}
<MatchType>.                    {ADJ; cout << "error in matchType" << endl;
                                 ERROR;}
<MatchArrow> "->"               {ADJ;
                                 cout << "begin str" << endl;
                                 BEGIN(MatchStr);}
<MatchArrow>.                   {ADJ; cout << "error in find arrow" << endl;
                                 ERROR;}
<MatchStr> {localDateStr}       {ADJ;
                                 cout << "find localDateStr:" << MATCHED
                                   << endl;
                                 BEGIN(MatchContent);}
<MatchStr> {localDateTimeStr}   {ADJ;
                                 cout << "find localDateTimeStr:" << MATCHED
                                   << endl;
                                 BEGIN(MatchContent);}
```

```
<MatchStr> {offsetDateTimeStr}  {ADJ;
                                cout << "find offsetDateTimeStr:" << MATCHED
                                  << endl;
                                BEGIN(MatchContent);}
<MatchStr>.                     {ADJ; cout << "invalid matchStr" << endl;
                                 ERROR;}
```

Here is a simplified implementation of a pattern matching in Ocaml, a functional language.

In this problem, let's consider an interesting lexical analyzer, which can analyze localized time, localized datetime and datetime with offset. With the related lex code listed above, please answer the following questions.

Note: We've changed some of the lexical rules to make it easier for you to write, so localDate, localDateTime and offsetDateTime are slightly different from the real format.

1. Regular Expressions (8')

   At first, you need to write down the regular expressions of the localDateStr, localDateTimeStr and offsetDateTimeStr (1.1 ~ 1.3), Their descriptions and examples are as follows:

   - A localDateStr starts with a """ and ends with a """, and is composed of three parts, each part is divided by a "-":
     - Year part: The year part has four digits, and only non-zero digit is allowed as the first character of the year part.
     - Month part: The month part has two digits, and must be a valid month number, for example 01 means January.
     - Date part: The date part also has two digits, and the number must between 01 and 31. (**Hint: You don’t need to consider leap year and the difference between months.**)
   - A localDateTimeStr starts with a """ and ends with a """, firstly contains all the three parts of the localDateStr, and then follows a "T", then has three parts, each part is divided by a ":":
     - Hour part: The hour part has two digits, and the number must between 00 and 23.
     - Minutes part: The minutes part has two digits, and the number must between 00 and 59.

- Seconds part: The seconds part has two digits, and the number must between 00 and 59.

- A offsetDateTimeStr starts with a """ and ends with a """, firstly contains all parts of localDateTimeStr, and then has two forms:
  - Zero form: only contains a "Z".
  - Zone form: first starts with a "+" or a "-", and then two digits between 00 and 12. And then follows a ":", and ends with two digits. If the last two digits is 12, then these two digits must be 00, otherwise can be between 00 and 59.

  Valid Examples:

  ```
  localDateStr:
      "2024-11-22", "2024-02-31", "1999-01-01"
  localDateTimeStr:
      "2024-11-22T14:00:00", "2024-02-31T00:00:00",
      "1999-01-01T23:59:59"
  offsetDateTimeStr:
      "2024-11-22T14:00:00Z", "2024-02-31T00:00:00-08:59",
      "2024-02-31T00:00:00+12:00"
  ```

  Invalid Example:

  ```
  localDateStr:
      "2024-11-22 (not end with """),
      "0024-02-31" (year part starts with a 0)
      "2024-13-32" (month and date are out of range)
  localDateTimeStr:
      "2024-11-22T1:00:00" (hour part only have one digit)
      "2024-11-22T00:5959" (lost a ":")
      "2024-11-22T00:60:60" (minutes and seconds are out of range)
  offsetDateTimeStr:
      "2024-11-22T14:00:0012:00" (missing a "+" or "-")
      "2024-11-22T14:00:00+12:30" (zone part minutes are out of range)
      "2024-11-22T14:00:00+13:60" (zone part hours and minutes are out
          of range)
  ```

2. What will be the output on the following input? The input is as follows. (10’)

（**Hint: You can use MATCHED to get the string you just matched. ERROR will simply shut down the program.**）

```
Input 1:

match request with

| LocalDateTime ->

   "2024-11-22T12:00:00"

| OffsetDateTime ->

   "2024-11-22T14:00:00+08:00";;

Output 1:




Input 2:

match request with

| LocalDateTime ->

   "0024-11-22T12:00:00";;

Output 2:



Input 3:

match request with

| LocalDateTime

   "2024-11-22T12:00:00Z"

Output 3:
```

3. Draw the minimized DFA that accepts a simple pattern matching for authentication on the following requirements. For any regular expression, the minimized DFA is a unique DFA having the smallest number of states that accept it. You will get part of the scores if your DFA is not minimized. (12’)

Problem 2: Grammar (50 points)

The following is a context-free grammar for a function closure syntax represented using lambda, here are the rules for the grammar:

Grammar rules
0. S'-> S$
1. S -> EL
2. EL-> E EL
3. EL-> E
4. E -> a = T ;
5. T -> lambda { EL }
6. T -> F
7. F -> a + F
8. F -> a

There are 7 terminals: "a", "+", "=", ";", "{", "}", "lambda".
Here, the "lambda{ }" expression represents a closure function that can be assigned to a variable. The contents within the "{ }" denote the body of the closure. For simplicity, all variable names in this syntax are replaced with "a".
Using the above grammar, answer the following questions:

1. Construct the state graph (DFA) for this grammar using LR(1) items. Is it an LR(1) grammar? (20')

2. Construct the LR(1) parsing table for the above DFA. Does there exist any conflict? If so, please list out the conflict(s) and the type(s) of the conflict(s). If not, please write "No conflict".(20')

3. Fill the table to show the operations of such a parser on the input string (10'):

"a = lambda { a = lambda { a = a + a ; } ; a = a + a ; } ;"

Problem 3: Semantic Analysis (20 points)

```
 1  function IsPrime(num:int) : int =
 2  let
 3    var isPrime : int := 1
 4    function checkDivisor(x:int, limit:int) : int =
 5    let
 6      var flag : int := 0
 7    in
 8      if x > limit then flag := 1
 9      else if num % x == 0 then flag := 0
10      else flag := checkDivisor(x + 1, limit)
11      flag
12    end
13  in
14    if num <= 1 then (
15      isPrime := 0
16    )
17    else (
18      let
19        var halfNum : int := num / 2
20      in
21        if checkDivisor(2, halfNum) == 0 then isPrime := 0
22      end
23    )
24    isPrime
25  end
```

In Tiger compiler's semantic analysis phase, we use **symbol tables** (also called **environments**) mapping identifiers to their types. An environment is a set of bindings denoted by an arrow(->). For example, **E**={g -> string, a -> int} means that in environment **E**, the identifier a is an **integer** variable and g is a **string** variable.

1. How many **environments** will be generated in the semantic analysis phase of this **program**? Please write them in detail. (You can answer like this: E0 = {id1->type1}, E1=E0 + {id2->type2, id3->type3},…) Note that a new symbol table will be generated when any identifier **enters or leaves** the current scope. Please refer to Appendix A for scoping rules when necessary. (10')

2. Each local variable in a program has a **scope** in which it is visible. Through looking into the **active environments**, the compiler can check whether the symbol exists. For the above **program**, you have written the **generated environments** in **Question1**. Please write all **active environments** for the following lines:2, 5, 8, 15, 21, 24**.** (10')