МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

ОТЧЕТ
по лабораторной работе № 2
на тему: «Построение лексического анализатора»
по дисциплине: «Теория автоматов и формальных языков»

Выполнили: Кожухова О.А., Шорин В.Д.
Институт приборостроения, автоматизации и информационных технологий
Направление: 09.03.04 «Программная инженерия»
Группа: 71ПГ
Проверили: Гордиенко А.П., Чижов А.В.

Отметка о зачете:

Дата: «____» _____ 2021 г.

Орел, 2021 г.

Задание на лабораторную работу:

Построить лексический анализатор для распознавания идентификатора, зарезервированного слова if и числа.

Выполнение работы:

КА для распознавания идентификатора:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ATaFL_Labs
{
   class IdAutomaticMachine : AutomaticMachine
   {
      public new enum States
      {
         start, state_1, state_2, state_3, state_finish, state_void
      }

      private new enum Actions
      {
         actionGetLetter, actionGetDigit, actionVoid, actionFinish
      }

      private new enum Signals
      {
         letter, digit, other
      }

      // Таблица переходов
      private new States[,] tableTransitions = new States[6, 3]
      {
                  //  letter              digit               other
         /*start*/      { States.state_1,     States.state_void,   States.state_void   },
         /*state_1*/    { States.state_2,     States.state_3,      States.state_finish },
         /*state_2*/    { States.state_2,     States.state_3,      States.state_finish },
         /*state_3*/    { States.state_2,     States.state_3,      States.state_finish },
         /*state_finish*/{ States.state_finish, States.state_finish, States.state_finish },
         /*state_void*/  { States.state_void,   States.state_void,   States.state_void   }
      };
```

```
// Таблица состояний (действий?)
private new Actions[,] tableActions = new Actions[6, 3]
{
                // letter                digit              other
   /*start*/       { Actions.actionGetLetter, Actions.actionVoid,     Actions.actionVoid   },
   /*state_1*/     { Actions.actionGetLetter, Actions.actionGetDigit, Actions.actionFinish },
   /*state_2*/     { Actions.actionGetLetter, Actions.actionGetDigit, Actions.actionFinish },
   /*state_3*/     { Actions.actionGetLetter, Actions.actionGetDigit, Actions.actionFinish },
   /*state_finish*/ { Actions.actionFinish,    Actions.actionFinish,   Actions.actionFinish },
   /*state_void*/   { Actions.actionVoid,      Actions.actionVoid,     Actions.actionVoid },
};

private new States currentState;
private new Actions currentAction;
private new Signals currentSignal;

public IdAutomaticMachine() { }

public override void NextStep(char symbol)
{
   if (char.IsLetter(symbol))
   {
      currentSignal = Signals.letter;
   }
   else if (char.IsDigit(symbol))
   {
      currentSignal = Signals.digit;
   }
   else
   {
      currentSignal = Signals.other;
   }
   currentState = tableTransitions[(int)currentState, (int)currentSignal];
}

public override void Initialize()
{
   currentState = States.start;
   currentAction = Actions.actionVoid;
}

public override bool IsFinishState => (currentState == States.state_finish);
```

```
    public override bool IsVoidState => (currentState == States.state_void);
  }
}
```

КА для распознавания зарезервированного слова if:

```
namespace ATaFL_Labs
{
  class IfAutomaticMachine : AutomaticMachine
  {
    public new enum States
    {
      start, state_1, state_finish, state_void
    }

    private new enum Signals
    {
      //letter_i, letter_I, letter_f, letter_F, other,
      letter_i_I, letter_f_F, other,
    }

    //// Таблица переходов
    //private new States[,] tableTransitions = new States[4, 5]
    //{
    //              // letter_i          letter_I          letter_f          letter_F          other
    //  /*start*/    { States.state_1,    States.state_1,    States.state_void,  States.state_void,  States.state_void },
    //  /*state_1*/    { States.state_void,    States.state_void,    States.state_finish,  States.state_finish,  States.state_void },
    //  /*state_finish*/{  States.state_finish,  States.state_finish,  States.state_finish,  States.state_finish,  States.state_finish },
    //  /*state_void*/  { States.state_void,    States.state_void,    States.state_void,    States.state_void,  States.state_void,  },
    //};

    // Таблица переходов
    private new States[,] tableTransitions = new States[4, 3]
    {
                  // letter_i_I          letter_f_F          other
      /*start*/    { States.state_1,    States.state_void,  States.state_void  },
      /*state_1*/    { States.state_void,  States.state_finish, States.state_void  },
      /*state_finish*/{ States.state_finish, States.state_finish, States.state_finish },
      /*state_void*/  { States.state_void,  States.state_void,  States.state_void,  },
    };
```

```
        private new States currentState;
        private new Signals currentSignal;

        public IfAutomaticMachine() { }

        public override void NextStep(char symbol)
        {
            if (symbol.ToString().ToLower() == "i")
            {
                currentSignal = Signals.letter_i_I;
            }
            else if (symbol.ToString().ToLower() == "f")
            {
                currentSignal = Signals.letter_f_F;
            }
            else
            {
                currentSignal = Signals.other;
            }
            currentState = tableTransitions[(int)currentState, (int)currentSignal];
        }

        public override void Initialize() => currentState = States.start;
        public override bool IsFinishState => (currentState == States.state_finish);
        public override bool IsVoidState => (currentState == States.state_void);
    }
}
```

КА для распознавания числа:

```
namespace ATaFL_Labs
{
    class DigitAutomaticMachine: AutomaticMachine
    {
        public new enum States
        {
            start, state_1, state_2, state_3, state_4, state_5, state_finish, state_void
        }

        private new enum Signals
        {
            digit, dot, comma, minus, other,
```

```
    }

    // Таблица переходов
    private new States[,] tableTransitions = new States[8, 5]
    {
                // digit            dot              comma            minus            other
      /*start*/    { States.state_1,    States.state_void, States.state_void, States.state_2,    States.state_void
},
      /*state_1*/  { States.state_1,    States.state_3,    States.state_4,    States.state_void, States.state_finish
},
      /*state_2*/        { States.state_1,        States.state_void,    States.state_void,    States.state_void,
States.state_void  },
      /*state_3*/        { States.state_5,        States.state_void,    States.state_void,    States.state_void,
States.state_void  },
      /*state_4*/        { States.state_5,        States.state_void,    States.state_void,    States.state_void,
States.state_void  },
      /*state_5*/         { States.state_5,         States.state_finish, States.state_finish, States.state_finish,
States.state_finish },
      /*state_finish*/{   States.state_finish,   States.state_finish,  States.state_finish,  States.state_finish,
States.state_finish },
      /*state_void*/   { States.state_void,    States.state_void,    States.state_void,    States.state_void,
States.state_void,  },
    };


    private new States currentState;
    private new Signals currentSignal;

    public DigitAutomaticMachine() { }

    public override void NextStep(char symbol)
    {
      if (char.IsDigit(symbol))
      {
        currentSignal = Signals.digit;
      }
      else if (symbol == '.')
      {
        currentSignal = Signals.dot;
      }
      else if (symbol == ',')
      {
```

```
            currentSignal = Signals.comma;
        }
        else if (symbol == '-')
        {
            currentSignal = Signals.minus;
        }
        else
        {
            currentSignal = Signals.other;
        }
        currentState = tableTransitions[(int)currentState, (int)currentSignal];
    }

    public override void Initialize()
    {
        currentState = States.start;
    }

    public override bool IsFinishState => (currentState == States.state_finish);
    public override bool IsVoidState => (currentState == States.state_void);
  }
}
```

КА:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ATaFL_Labs
{
  abstract class AutomaticMachine
  {
    public enum States { start };
    protected enum Actions { actionVoid };
    protected enum Signals { };
    protected States[,] tableTransitions;
    protected Actions[,] tableActions;

    protected States currentState;
    protected Actions currentAction;
    protected Signals currentSignal;

    public AutomaticMachine() { }
```

```
        public abstract void NextStep(char symbol);
        public abstract void Initialize();

        public abstract bool IsFinishState { get; }
        public abstract bool IsVoidState { get; }
    }
}
```

Программа:

```
using System;
using System.Collections.Generic;

namespace ATaFL_Labs
{
    class Program
    {
        private enum Tokens
        {
            id, digit, keyWord, unknown
        }

        private enum Priority
        {
            keyWord, id, digit, unknown,
        }

        private static IdAutomaticMachine idAutomaticMachine;

        private static List<AutomaticMachine> machines;

        static void Main(string[] args)
        {
            machines = new List<AutomaticMachine>()
            {
                new IdAutomaticMachine(),
                new DigitAutomaticMachine(),
                new IfAutomaticMachine(),
            };

            string userString = "asd qwe1 123 -12 32.4 if -54.3 86,2 -75.1";
            Console.WriteLine($"String: {userString}");

            string[] words = userString.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
```

```
    foreach (var word in words)
    {
        string s = word + " ";
        InitializeAll();
        foreach (var symbol in s)
        {
            DoAllAM(symbol);
        }
        bool isAllMachinesEndWork = IsAllMachinesEndWork();
        if (isAllMachinesEndWork)
        {
            Tokens token = CheckPriority();
            PrintResult(word, token);
        }
    }
}

private static Tokens CheckPriority()
{
    List<Priority> priorities = new List<Priority>();

    foreach (var machine in machines)
    {
        if (machine.IsFinishState)
        {
            priorities.Add(GetPriority(machine.GetType().Name));
        }
        else if (machine.IsVoidState)
        {
            priorities.Add(Priority.unknown);
        }
    }

    priorities.Sort();

    return GetTokenByPriority(priorities[0]);
}

private static Priority GetPriority(string name)
{
    switch (name)
```

```
    {
        case "IdAutomaticMachine":      return Priority.id;
        case "DigitAutomaticMachine":   return Priority.digit;
        case "IfAutomaticMachine":      return Priority.keyWord;
        default:                        return Priority.unknown;
    }
}

private static Tokens GetTokenByPriority(Priority priority)
{
    switch (priority)
    {
        case Priority.keyWord:  return Tokens.keyWord;
        case Priority.id:       return Tokens.id;
        case Priority.digit:    return Tokens.digit;
        case Priority.unknown:  return Tokens.unknown;
        default:                return Tokens.unknown;
    }
}

private static void PrintResult(string word, Tokens token) => Console.WriteLine($"{word} is {token}\n");

private static bool IsAllMachinesEndWork()
{
    int count = 0;
    foreach (var machine in machines)
    {
        if (machine.IsFinishState || machine.IsVoidState)
        {
            count++;
        }
    }

    return (count == machines.Count);
}

private static void InitializeAll()
{
    foreach (var machine in machines)
    {
        machine.Initialize();
    }
```

```
        }

        private static void DoAllAM(char symbol)
        {
            foreach (var machine in machines)
            {
                machine.NextStep(symbol);
            }
        }
    }
}
```