ՀԱՅԱՍՏԱՆԻ ՀԱՆՐԱՊԵՏՈՒԹՅԱՆ ԿՐԹՈՒԹՅԱՆ, ԳԻՏՈՒԹՅԱՆ,ՄՇԱԿՈՒՅԹԻ ԵՎ ՍՊՈՐՏԻ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ (ՀԻՄՆԱԴՐԱՄ)

ԻՆՍՏԻՏՈՒՏ *ՏԵՂԵԿԱՏՎԱԿԱՆ ԵՎ ՀԵՌԱՀԱՂՈՐԴԱԿՑԱԿԱՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐԻ ՈՒ ԷԼԵԿՏՐՈՆԻԿԱՅԻ*

ԱՄԲԻՈՆ ԱԼԳՈՐԻԹՄԱԿԱՆ ԼԵԶՈՒՆԵՐԻ ԵՎ ԾՐԱԳՐԱՎՈՐՄԱՆ

ՀԵՏԱԶՈՏԱԿԱՆ ԱՇԽԱՏԱՆՔԻ ԱՌԱՋԱԴՐԱՆՔ

«ԻՆՖՈՐՄԱՑԻՈՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐ» առարկայից

Աշխատանքի թեման՝ «Թյուրինգի մեքենայի սիմուլյատորի մշակումը C# լեզվով»_____

_____

_____

_____

_____

_____

_____

ՄՄԹ 340 ակադեմիական խմբի ուսանող Սևոյան Կամո Կորյունի

ազգանուն անուն հայրանուն

Հաշվեբացատրագրի բովանդակությունը

1. Տիտղոսաթերթը
2. ԿԱ առաջադրանքը
3. Ալգորիթմը, ալգորիթմի նկարագրությունը
4. Ծրագիրը, ծրագրի նկարագրությունը
5. Հաշվարկների արդյունքները

Ամբիոնի վարիչի պ/կ՝ ***Ս. Ս. Ավետիսյան***

_____

ամսաթիվ ստորագրություն

Աշխատանքի ղեկավար Սերգեյ Մուշեղի Սարգսյան _____

ազգանուն անուն ամսաթիվ ստորագրություն

Աշխատանքի առաջադրանքը ստացա ***14.02. 2024թ.*** Սևոյան Կամո _____

ազգանուն անուն ամսաթիվ ստորագրություն

ՀԱՅԱՍՏԱՆԻ ՀԱՆՐԱՊԵՏՈՒԹՅԱՆ ԿՐԹՈՒԹՅԱՆ, ԳԻՏՈՒԹՅԱՆ,ՄՇԱԿՈՒՅԹԻ ԵՎ ՍՊՈՐՏԻ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ (ՀԻՄՆԱԴՐԱՄ)

ՏՀՏԷ ինստիտուտ

ԱԼ և Ծ ամբիոն

ՀԱՇՎԵԲԱՑԱՏՐԱԳԻՐ

«ԻՆՖՈՐՄԱՑԻՈՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐ» առարկայի հետազոտական աշխատանքի

Թեմա՝

«Թյուրինգի մեքենայի սիմուլյատորի մշակումը C# լեզվով»_____

_____

_____

_____

Ուսանող _____Սևոյան Կամո Կորյունի_____ _____
ազգանուն, անուն հայրանուն ստորագրություն

Ղեկավար _____Սարգսյան Սերգեյ Մուշեղի_____ _____
ազգանուն, անուն հայրանուն ստորագրություն

Ամբիոնի վարիչի պ/կ ***Ս. Ս. Ավետիսյան*** _____
ստորագրություն

Հանձնաժողովի անդամներ _____ _____
ստորագրություն

_____ _____
ստորագրություն

Երևան 2024

Հայաստանի Ազգային Պոլիտեխնիկական Համալսարան

Կիրառական մաթեմատիկայի և ֆիզիկայի ֆակուլտետ

Կուրսային աշխատանք

Առարկա՝ Ինֆորմացիոն տեխնոլոգիաներ

Դասախոս՝ Սարգսյան Սերգեյ

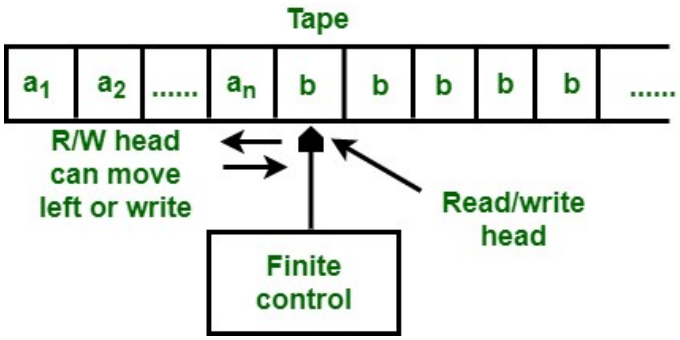
Ուսանող՝ Սևոյան Կամո

Խումբ՝ ՄՄԹ 340

Երևան 2024

Թյուրինգի մեքենա: Ընդհանուր դրույթներ և հատկություններ

Թյուրինգի մեքենան կազմված է երկու կողմից անվերջ ձգվող ժապավենից, գրող-կարդացող գլխիկից և ղեկավարող հարմարանքից՝



Մեքենան աշխատում է ժամանակի առանձին $t = 0, 1, 2, \ldots$ պահերին: Ժապավենը բաղկացած է բջիջներից: Ժամանակի յուրաքանչյուր պահի ամեն մի բջջում գրվում է

$$A = \{a_0, a_1, \ldots, a_{n-1}\}$$

այբուբենի միայն մեկ նշան:

Գրող-կարդացող գլխիկը ժամանակի յուրաքանչյուր պահի գտնվում է, դիտարկում է, ժապավենի բջիջներից միայն մեկը, կարդում է այդ բջջում գրված նշանը, կարող է այդ բջջում գրել A այբուբենի կամայական տառ և հաջորդ պահին դիտարկել դիտարկված բջջին հարևան ձախ կամ աջ բջիջներից որևէ մեկը: Հետագայում A -ն կանվանենք մեքենայի արտաքին այբուբեն:

Ղեկավարող հարմարանքը ժամանակի յուրաքանչյուր պահի գտնվում է

$$Q = \{q_0, q_1, \ldots, q_{r-1}, p_1, \ldots, p_k\}$$

այբուբենի, որտեղ $r > 0, k > 0$՝ ներքին վիճակների բազմության, վիճակներից մեկում և միայն մեկում: Հետագայում Q-ն կանվանենք մեքենայի ներքին այբուբեն:

$\bar{Q} = \{q_0, \ldots, q_{r-1}\}$ բազմությունը կանվանենք աշխատանքային վիճակների բազմություն, իսկ $\{p_1, \ldots, p_k\}$ բազմությունը՝ եզրափակիչ վիճակների բազմություն: q_0-ն կանվանենք սկզբնական վիճակ:

Մեքենայի աշխատանքի ընթացքում ղեկավարող հարմարանքը, ելնելով ներքին վիճակից և գրող-կարդացող գլխիկի դիտարկված բջջում գրված նշանից, կարող է՝

ա) փոխել ներքին վիճակը,

բ) փոխել գլխիկի կողմից դիտարկվող բջջի նշանը,

գ) փոխել գլխիկի դիրքը՝ հաջորդ պահին՝ այն թողնելով տեղում կամ տեղափոխելով դիտարկվող բջջին հարևան ձախ կամ աջ բջիջը:

Թյուրինգի մեքենայի ա) , բ) և գ) գործողությունները միարժեքորեն որոշվում են՝

$$\lambda : \overline{Q} \times A \to Q,$$

$$\delta: \overline{Q} \times A \to A,$$

$$\nu: \overline{Q} \times A \to \{\text{S}, \text{Ձ}, \text{Ա}\}$$

արտապատկերումներով, որտեղ $A = \{a_0, a_1, \dots, a_{n-1}\}$-ն արտաքին այբուբենն է,

$Q = \{q_0, q_1, \dots, q_{r-1}, p_1, \dots, p_k\} -$ ն՝ ներքին այբուենը, $\bar{Q} = \{q_0, \dots, q_{r-1}\}$-ն՝ աշխատանքային վիճակների բազմությունը, Տ՝ գրող կարդացող գլխիկը պետք է մնա տեղում, Ձ՝ գլխիկը պետք է գնա ձախ և Ա ՝ գլխիկը պետք է գնա աջ:

Սահմանում: A, Q բազմությունների և λ, δ, ν արտապատկերումների հնգյակը անվանում են Թյուրինգի մեքենա / ԹՄ / և նշանակում՝

$$\boldsymbol{T_{q_0} =< A, Q, \lambda, \delta, \nu >}$$

արտահայտությամբ կամ, պարզապես T -ով:

Բարձր կարգի լեզվի նախագծում

Առավել բարդ ալգորիթմերի համար Թյուրինգի մեքենայի ծրագրի կազմումը առաջացնում է բարդություններ, որն առավելապես կապված է վիճակների բազմության մեծացման հետ: Ուստի անհրաժեշտություն է առաջանում ինչ-որ կերպ հեշտացնել այդ գործընթացը: Խնդիրը լուծելու համար կստեղծենք լեզու, որը կունենա օգտագործողի համար պարզ հրամաններ և հեշտ քերականություն: Դրա համար անհրաժեշտ է ստեղծել թարգմանող ծրագիր այդ լեզվից դեպի Թյուրինգի մեքենայի համար հասկանալի հրամաններ: Ստեղծվող լեզուն նկարագրելու համար կօգտվենք Չոմսկիի նկարագրման եղանակից:

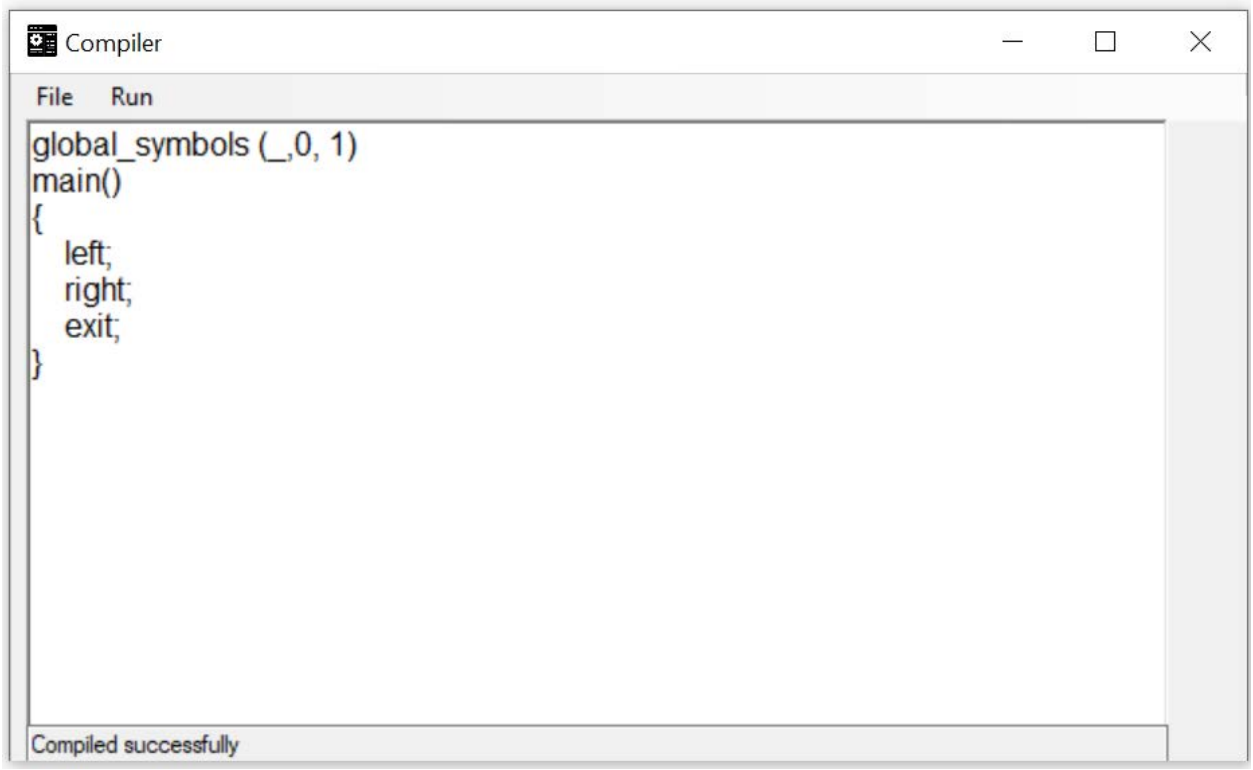
- <program> := <symbol list> <main-statement>
- <main-statement> := <statement>
- <statement> := <block-statement> | <if-statement> | <if_else-statement> | <while-statement> | <do_while-statement>| <repeat_until-statement> | <write-statement> | <left-statement> | <right-statement> | <exit-statement> | <error-statement> | <continue-statement> | <break-statement> | <switch-statement>
- <block-statement> := '{' <statement>+ '}'
- <if-statement> := 'if' <symbol list> <statement>
- <if_else-statement> := 'if' <symbol list> <statement> 'else' <statement>
- <while-statement> := 'while' <symbol list> <statement>
- <do_whlie-statement> := 'do' <statement> 'while' <symbol list> ';'
- <repeat_until-statement> := 'repeat' <statement> 'until' <symbol list> ';'
- <write-statement> := 'write' <symbol> ';'
- <left-statement> := 'left' ';'
- <right-statement> := 'right' ';'
- <exit-statement> := 'exit' ';'
- <error-statement> := 'error' ';'
- <continue-statement> := 'continue' ';'
- <break-statement> := 'break' ';'
- <switch-statement> ::= 'switch' '()' <case-statements>
- <case-statements> ::= <case-statement>+
- <case-statement> := 'case' '(' <symbol> ')' <statement>
- <symbol list> := <not> '(' <symbol> (<comma> <symbol>)* ')' | '(' <symbol> (<comma> <symbol>)* ')'
- <symbol> = ASCII printable character
- <not> := 'not'
- <comma> := ','

Լեզուն ունի հինգ պարզ հրաման որոնք են՝ **left**, **right**, **write**, **exit**, **error**: Լեզվի բաղադրյալ հրամաններն են՝ **if**, **if-else**, **while**, **do-while**, **repeat-until**, **switch-case**:

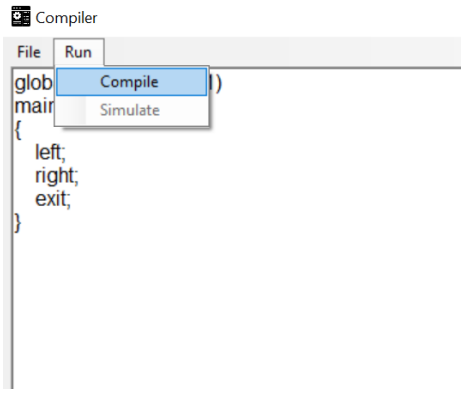
Լեզվում հնարավոր է օգտագործել բլոկային արտահայտություններ կամայական խորությամբ: Նշենք նաև, որ լեզվի ճանաչման **parse** փուլը կատարվում է top-down (բարդից պարզ) եղանակով: Թարգմանիչ ծրագիրը ընդունում է ստեղծված լեզվով գրված ծրագիրը որի հաջող թարգմանելու դեպքում գեներացվում է Թյուրինգի մեքենայի սիմուլյատորի համար նախորդիվ սահմանված ձևաչափով ծրագիր: Մուտքային ծրագրում (syntaxic) քերականական և (semantic) իմաստային սխալների դեպքում թարգմանիչ ծրագիրը տալիս է սխալի համապատասխան հաղորդագրություն:

Գրաֆիկական ինտերֆեյսի նախագծում (GUI)

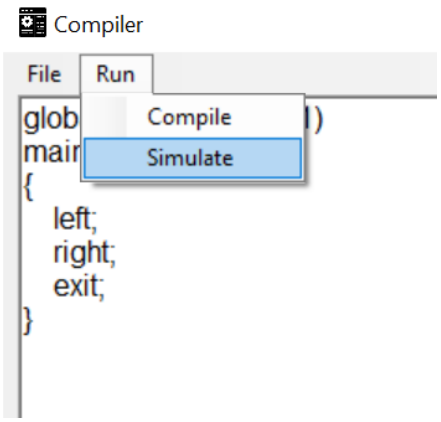
Windows Forms գրաֆիկական ինտերֆեյսի միջոցով ստեղծվել է տրված ծրագրի գրաֆիկական մասը:



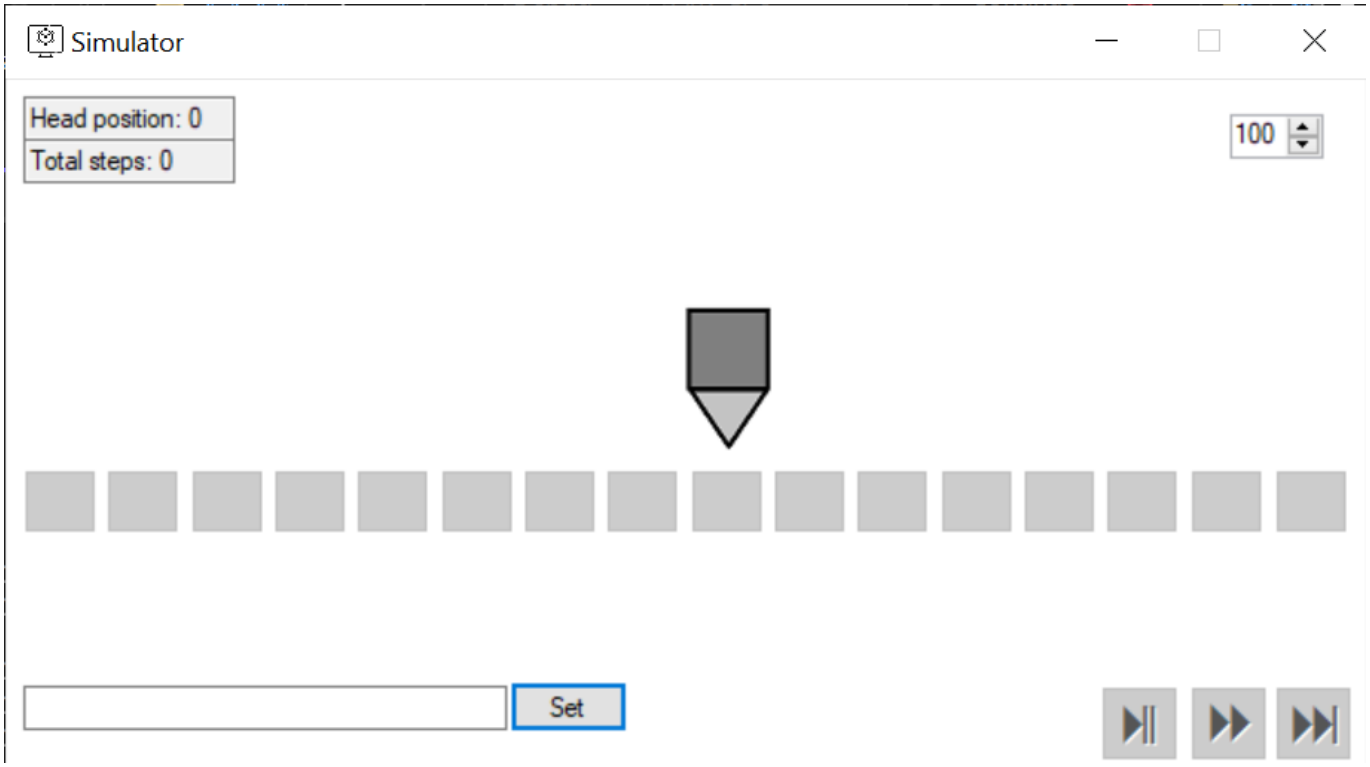
Բացված պատուհանում մուտքագրվում է ծրագիրը, կամ բացվում է ծրագիր պարունակող ֆայլը: **Compile** կոճակի միջոցով բարձր մակարդակով գրված ծրագիրը թարգմանվում է Թյուրինգի մեքենայի աղյուսակային ներկայացման:



Սիմուլյատորի աշխատանքը սկսվում է **Simulate** կոճակի սեղմումով:



Նոր պատուհանում մուտքագրվում են սիմուլյատորի մուտքային տվյալները: Սիմուլյատորը աշխատանքը սկսելու համար անհրաժեշտ է սեղմել ներքևի աջ անկյունում գտնվող երեք կոճակներից որևէ մեկը:



Հատված ծրագրից

Compiler.cs

```
public string Compile()
{
    MainNode node = parser.Parse();

    string symbols = parser.GlobalSymbols.Symbols;
    globalSymbols = symbols;

    List<string> lines = new List<string>();

    string temp = "";
    int stateNumber = 0;

    CompileNode(node, lines, ref stateNumber, symbols);

    temp += symbols[0];

    for (int i = 1; i < symbols.Length; ++i)
    {
        temp += "," + symbols.Substring(i, 1);
    }
    temp += "\n";

    temp += "q0";
    for (int i = 1; i < stateNumber; ++i)
    {
        temp += ",q" + i.ToString();
    }
    temp += ",h\n";

    for (int i = 0; i < lines.Count; ++i)
    {
        temp += lines[i];
    }

    temp += "q0\nh\n_\n";

    return temp;
}

private void CompileIfNode(Node node, List<string> lines, ref int stateNumber, string globalSymbols)
{
    string temp = "";
    int oldStateNumber;
    int currentLine;

    ++stateNumber;
    oldStateNumber = stateNumber;
    lines.Add("");
    currentLine = lines.Count - 1;
    IfNode ifNode = node as IfNode;
    bool hasNot = ifNode.Symbols.HasNegation;

    CompileNode(ifNode.Statement, lines, ref stateNumber, globalSymbols);

    if (hasNot ^ ifNode.Symbols.Symbols.Contains(globalSymbols[0]))
    {
        temp += globalSymbols.Substring(0, 1) + ",q" + oldStateNumber.ToString() + ",@";
    }
    else
    {
        temp += globalSymbols.Substring(0, 1) + ",q" + stateNumber.ToString() + ",@";
    }

    for (int i = 1; i < globalSymbols.Length; ++i)
    {
        if (hasNot ^ ifNode.Symbols.Symbols.Contains(globalSymbols[i]))
        {
            temp += "\t|\t" + globalSymbols.Substring(i, 1) + ",q" + oldStateNumber.ToString() + ",@";
        }
        else
        {
            temp += "\t|\t" + globalSymbols.Substring(i, 1) + ",q" + stateNumber.ToString() + ",@";
        }
    }

    temp += "\n";
    lines[currentLine] = temp;
}
```

Compiler.cs

```
private void CompileBlockNode(Node node, List<string> lines, ref int stateNumber, string globalSymbols)
{
    BlockNode blockNode = node as BlockNode;
    for (int i = 0; i < blockNode.Statements.Count; ++i)
    {
        CompileNode(blockNode.Statements[i], lines, ref stateNumber, globalSymbols);
    }
}



private void CompileLeftNode(Node node, List<string> lines, ref int stateNumber, string globalSymbols)
{
    string temp = "";
    ++stateNumber;

    temp += globalSymbols.Substring(0, 1) + ",q" + stateNumber.ToString() + ",<";
    for (int i = 1; i < globalSymbols.Length; ++i)
    {
        temp += "\t|\t" + globalSymbols.Substring(i, 1) + ",q" + stateNumber.ToString() + ",<";
    }
    temp += "\n";
    lines.Add(temp);
}



private void CompileWriteNode(Node node, List<string> lines, ref int stateNumber, string globalSymbols)
{
    ++stateNumber;
    string currentSymbol = (node as WriteNode).Symbol.ToString();
    string temp = currentSymbol;

    temp += ",q" + stateNumber.ToString() + ",@";

    for (int i = 1; i < globalSymbols.Length; ++i)
    {
        temp += "\t|\t" + currentSymbol + ",q" + stateNumber.ToString() + ",@";
    }
    temp += "\n";
    lines.Add(temp);
}



private void CompileContinueNode(Node node, List<string> lines, ref int stateNumber, string globalSymbols)
{
    ++stateNumber;

    lines.Add("");
    ((node as FlowControllNode).OwnerLoop as LoopNode).ContinueStates.Add(stateNumber);
}


private void CompileBreakNode(Node node, List<string> lines, ref int stateNumber, string globalSymbols)
{
    ++stateNumber;

    lines.Add("");
    ((node as FlowControllNode).OwnerLoop as LoopNode).BreakStates.Add(stateNumber);
}
```

SimulatorForm.cs

```
void SimulationTimer_Tick(object sender, EventArgs e)
{
    if (IsContinuouslyRunning)
    {
        StepWrapper(visualize: true);

        if (_simulatorState == Simulator.MachineState.Terminated)
        {
            IsContinuouslyRunning = false;
            FinishSimulation();
        }
    }
}


async void InstantaneousEvaluateButton_Click(object sender, EventArgs e)
{
    singleStepButton.Enabled = false;
    continiousStepButton.Enabled = false;

    await Task.Run(() =>
    {
        while (true)
        {
            StepWrapper(visualize: false);
            if (_simulatorState == Simulator.MachineState.Terminated)
            {
                break;
            }
        }
        _parent.VisualizeResult();
        positionText.Text = $"Head position: {_parent.simulator.tape.Position}";
        numStepsText.Text = $"Total steps: {_parent.simulator.NumSteps}";
        FinishSimulation();
    });

}



void StepWrapper(bool visualize)
{

    _simulatorState = _parent.StepSimulator(visualize);
    if (visualize)
    {
        positionText.Text = $"Head position: {_parent.simulator.tape.Position}";
        numStepsText.Text = $"Total steps: {_parent.simulator.NumSteps}";
    }

    if (_simulatorState == Simulator.MachineState.Failed)
    {
        continiousStepButton.Text = "⏩";
        DisableStepButtons();
        inputSetButton.Enabled = true;
        IsContinuouslyRunning = false;
        MessageBox.Show("The simulator terminated with error.");
    }

}
```

Parser.cs

```
private Node ParseBlockStatement(Node root)
{
    BlockNode node = new BlockNode{Root = root};

    NextToken();

    if (GetOffsetToken(0) == TokenType.RightBrace)
    {
        throw new SyntaxErrorException($"Empty block statement in {GetCurrentTokenPosition()}.");
    }

    while (GetOffsetToken(0) != TokenType.RightBrace)
    {
        Node node0 = ParseStatement(node);
        node.Statements.Add(node0);
        NextToken();
    }

    if ((node as BlockNode).Root.Type == NodeType.Main)
    {
        int j = (node as BlockNode).Statements.Count - 1;

        if (((node as BlockNode).Statements[j].Type != NodeType.Exit) &&
        ((node as BlockNode).Statements[j].Type != NodeType.Error))
        {
            node.Statements.Add(new PrimaryNode(NodeType.Exit));
        }
    }
    return node;

}



private bool IsInLoop(Node node)
{
    Node it = node;

    while (
        (it.Root.Type != NodeType.While) &&
        (it.Root.Type != NodeType.DoWhile) &&
        (it.Root.Type != NodeType.RepeatUntil) &&
        (it.Root.Type != NodeType.Main)
        )
    {
        it = it.Root;
    }

    if (it.Root.Type == NodeType.Main)
    {
        return false;
    }

    (node as FlowControllNode).OwnerLoop = it.Root;
    return true;
}



private Node ParseContinueStatement(Node root)
{
    NextToken();

    if (GetOffsetToken(0) != TokenType.Semicolon)
    {
        throw new SyntaxErrorException($"Expected semicolon in {GetCurrentTokenPosition()}.");

    }

    FlowControllNode node = new FlowControllNode(NodeType.Continue){Root = root};

    if (!IsInLoop(node))
    {
        throw new SyntaxErrorException("Flow control statement continue should only be within a loop");
    }

    return node;
}
```

Simulator.cs

```
public MachineState Step()
{
    _currentSymbol = Tape.Get(Tape.Position);

    (string, char) key = (_currentState, _currentSymbol);

    if (!(_lambda.ContainsKey(key) && _delta.ContainsKey(key) && _nyu.ContainsKey(key)))
    {
        return MachineState.Failed;
    }

    string newState = _lambda[key];
    string newSymbol = _delta[key];
    string move = _nyu[key];

    Tape.Write(newSymbol);
    Tape.Move(move);

    _currentState = newState;

    if (newState == _haltState)
    {
        return MachineState.Terminated;
    }

    NumSteps++;
    return MachineState.Running;
}
```