**Лабораторная работа №7**

Выполнила: студент группы 4309

Канькина Ольга Александровна

Генератор - это подпрограмма, которая может возвращать очередное значение и автоматически сохранять и возобновлять своё состояние для возврата следующего значения. Генератор похож на функцию, возвращающую массив, поскольку он имеет параметры, может быть вызван и возвращает последовательность значений.

Как выглядит цикл с генератором:

1. Цикл выполняется нужное количество раз.
2. На каждом шаге цикла генератор получает какое-то значение, отдаёт его в нужное место и забывает всё напрочь.
3. Генератор не помнит значение, которое он отдавал до этого, и не знает, что он будет отдавать на следующем шаге. Всё, что у него есть, — данные, которые нужно обработать на текущем шаге.
4. Память под работу генератора выделяется, только когда он генерирует новые данные. Пока генератор стоит или не выдаёт данные — память не выделяется.

Чаще всего генераторы используют как функции. Каждый раз, когда обращаются к такой функции-генератору, она делает так:

1. Берёт новую порцию данных из указанного ей источника.
2. Обрабатывает данные.
3. Возвращает результат.
4. Забывает про всё до следующего вызова.

Главный плюс генераторов — их можно указывать в качестве диапазона в циклах. На каждом шаге цикл получает новое значение от генератора и работает уже с ним. Как только у генератора заканчиваются варианты и он останавливается — цикл тоже останавливается.

Генератор — функция, которая генерирует последовательность результатов вместо одного значения. Вместо того, чтобы возвращать значение, мы создаём серию значений. Вызов функции генератора создает объект-генератор. Однако функция не запускается. Функция генератор выполняется только при вызове. Функция генератор возобновляется при следующем вызове. При завершении итератора возбуждается исключение.

Атрибутивная грамматика - это формальный способ дополнить формальную грамматику обработкой семантической информации. Семантическая информация хранится в атрибутах, связанных с терминальными и нетерминальными символами грамматики. Значения атрибутов являются результатом правил оценки атрибутов, связанных с созданием грамматики.

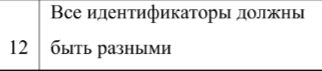
Генераторы в программировании представляют собой мощный инструмент, позволяющий работать с последовательностями данных. Однако, у генераторов есть свои ограничения и недостатки, которые могут затруднять их применение при решении сложных задач. Одно из ограничений генератора заключается в его однонаправленности. Генераторы позволяют передвигаться только вперед по последовательности данных, что ограничивает возможность обратного прохода или повторного прохода.

Атрибуты позволяют передавать информацию из любого места в абстрактном синтаксическом дереве в любое другое место контролируемым и формальным способом.

Каждая семантическая функция имеет дело с атрибутами символов, встречающимися только в одном производственном правиле: и параметры семантической функции, и ее результат являются атрибутами символов из одного конкретного правила. Когда семантическая функция определяет значение атрибута символа в левой части правила, атрибут называется синтезированным; в противном случае он называется унаследованным. Таким образом, синтезированные атрибуты служат для передачи семантической информации вверх по дереву синтаксического анализа, в то время как унаследованные атрибуты позволяют передавать значения от родительских узлов вниз и по всему синтаксическому дереву.

В простых приложениях, таких как вычисление арифметических выражений, атрибутивная грамматика может использоваться для описания всей задачи, которая должна быть выполнена, помимо прямого синтаксического анализа; в сложных системах, например, при создании средства языкового перевода, такого как компилятор, она может использоваться для проверки семантики, связанной с грамматикой, представляющей правила языка, которые явно не передаются определением синтаксиса. Она также может использоваться парсерами или компиляторами для перевода синтаксического дерева непосредственно в код для некоторой конкретной машины или на некоторый промежуточный язык.

**Задание**. Для предложенного преподавателем варианта контекстного условия расширить атрибутную грамматику из лабораторной работы № 4 добавлением атрибутов, правил их вычисления, правил вычисления контекстных условий. Включить в программу синтаксического анализатора из лабораторной работы № 4 действия по вычислению атрибутов и проверки контекстных условий.



Код программы:

Form1.cs:

using nsLex;

using nsLexMainForm;

using nsSynt;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using WindowsFormsApp26;

namespace CHashTableList

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public CHashTableList htl = new CHashTableList(2);

public Form1()

{

InitializeComponent();

tbFSource.AppendText("100\_a\_10010\_a\_" + "\r\n");

int n = tbFSource.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

public void TablesToMemo(object sender, System.EventArgs e)

{

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

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

htl.TableToStringList(0, listTable);

//for (int i = 0; i < listTable.Count; i++)

// listBox1.Items.Add(listTable[i]);

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

htl.TableToStringList(1, listTable);

//for (int i = 0; i < listTable.Count; i++)

// listBox2.Items.Add(listTable[i]);

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

listTable.Clear();

}

private void btnFStart\_Click\_1(object sender, EventArgs e)

{

tbFMessage.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree); // Используем существующий экземпляр treeView2

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.enumPState = TState.Start;

try

{

Synt.Lex.NextToken();

Synt.S();

SyntTree.ExpandAll();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void btnFRecord\_Click\_1(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

//Lex.intPSourceColSelection = 0;

//Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 0, ref b))

{

TablesToMemo(this, e);

}

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmdt):

{

hashFunction.AddWord(hashTableRezerv, ":");

s1 = "rez " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmComma):

{

hashFunction.AddWord(hashTableRezerv, ",");

s1 = "rez " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

case (TToken.lxmmul):

{

hashFunction.AddWord(hashTableRezerv, "\_");

s1 = "rez " + Lex.strPLexicalUnit; int b = 0;

if (htl.AddLexicalUnit(Lex.strPLexicalUnit, 1, ref b))

{

TablesToMemo(this, e);

}

break;

}

}

String m = "(" + s + "" + s1 + ")";

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += tbFSource.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void searchBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox1.SelectedItem.ToString()) == 1)

{

searchBtn.BackColor = Color.Green;

}

else

{

searchBtn.BackColor = Color.Red;

}

}

private void addBtn\_Click\_1(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

}

private void deleteBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

deleteBtn.BackColor = Color.Green;

}

else

{

deleteBtn.BackColor = Color.Red;

}

}

private void changeBtn\_Click\_1(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox1.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox1.Text.ToString());

changeBtn.BackColor = Color.Green;

}

else

{

changeBtn.BackColor = Color.Red;

}

}

private void reloadBtn\_Click\_1(object sender, EventArgs e)

{

listBox1.Items.Clear();

listBox2.Items.Clear();

listBox3.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox1.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox2.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox3.Items.Add(string.Join(", ", entry.Value));

}

}

private void convertBtn\_Click(object sender, EventArgs e)

{

CopyTreeView(SyntTree, SyntTree2);

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree2);

}

private void CopyTreeNodes(TreeNodeCollection sourceNodes, TreeNodeCollection targetNodes)

{

foreach (TreeNode sourceNode in sourceNodes)

{

TreeNode newNode = new TreeNode(sourceNode.Text);

targetNodes.Add(newNode);

CopyTreeNodes(sourceNode.Nodes, newNode.Nodes); // Рекурсивно копируем дочерние узлы

}

}

private void CopyTreeView(TreeView sourceTreeView, TreeView targetTreeView)

{

targetTreeView.Nodes.Clear(); // Очищаем существующие узлы в целевом TreeView

CopyTreeNodes(sourceTreeView.Nodes, targetTreeView.Nodes);

}

private void duplicateBtn\_Click(object sender, EventArgs e)

{

try

{

Duplicate duplicate = new Duplicate(SyntTree);

duplicate.ChekingForDuplicate(SyntTree);

throw new Exception("Текст верный");

}

catch (Exception exc)

{

tbFMessage.Text = "";

tbFMessage.Text += exc.Message;

}

}

private void treeView1\_AfterSelect(object sender, TreeViewEventArgs e)

{

}

public void SemantFunction()

{

try

{

uSemantAnalyzer Semant = new uSemantAnalyzer(SyntTree);

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

}

}

}

}

Duplicate.cs:

using nsLex;

using nsLexMainForm;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp26

{

public class Duplicate

{

MyHashFunction myHashFunction = new MyHashFunction();

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>(); // ключом является целое число, а значением список строк

private TreeView tree;

CLex Lex = new CLex();

public Duplicate(TreeView treeView)

{

tree = treeView;

hashTableIdentifier = new Dictionary<int, List<string>>();

}

public void ChekingForDuplicate(TreeView tree)

{

foreach (TreeNode node in tree.Nodes) // Рассматриваем всевозможные ветви (node) нашего TreeView

{

ChekingForDuplicate(node);

}

}

public void ChekingForDuplicate(TreeNode node)

{

if (node.Text == "B") // Если мы попали в ветку под названием B

{

if (node.Nodes.Count > 0) // Если эта ветка не пустая

{

foreach (TreeNode node2 in node.Nodes) // Перебираем все подветви нода с названием B, тем самым рассматривая каждый лексический юнит

{

string binaryValue = node2.Text;

if (IsString(binaryValue))

{

if (myHashFunction.SearchWord(hashTableIdentifier, binaryValue) == 0) //Если текст подузла является строкой и отсутствует, то он добавляется

{

myHashFunction.AddWord(hashTableIdentifier, binaryValue);

}

else

{

tree.Nodes.Clear();

throw new Exception("Слово "+binaryValue + " повторяется");

}

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

ChekingForDuplicate(childNode); // Тут мы разделяем опять ветвь (node) на ещё более мелкие, то есть углубляемся

}

}

private bool IsString(string value) //функция, которая проверяет, является ли заданное значение строкой

{

foreach (var c in value)

if (c == 'a' || c == 'b' || c == 'c' || c == 'd')

return true;

return false;

}

}

}

Результат работы программы:

