**Лабораторная работа №6**

Выполнила: студент группы 4309

Курбанов Д.И.

В программировании есть инструмент, который позволяет экономить память и при этом обрабатывать огромные массивы данных. Это генераторы.

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.

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**.

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде

Код программы:

TreeConverter.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if ((node.Text == "S") || (node.Text == "A"))

{

if (node.Nodes.Count > 0)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

ConvertBinaryToDecimal(childNode);

}

}

private bool IsBin(string value)

{

foreach (var c in value)

if (c != '0' && c != '1')

return false;

return true;

}

}

}

Form1.cs:

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;

namespace WindowsFormsApp4

{

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 Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 caaa cab 010000001 caaaa");

int n = textBox2.Lines.Length;

}

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

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree);

// uSyntAnalyzer Synt2 = new uSyntAnalyzer(SyntTree2);

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

TreeNode parent = new TreeNode("TreeView");

SyntTree.Nodes.Add(parent);

Synt.S(parent);

Synt.Lex.NextToken();

SyntTree.ExpandAll();

Generator.CopyTree(SyntTree, SyntTree2);

Generator.RefactorTree(SyntTree2);

SyntTree2.ExpandAll();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

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

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

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

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

{

}

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

{

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

private void SyntTree2\_AfterSelect(object sender, TreeViewEventArgs e)

{

}

}

}

Результат работы программы:

