**COMSATS UNIVERSITY OF ISLAMABAD,**

**ATTOCK CAMPUS**

**LAB TERMINAL**

**Submitted by: Ahmed Ali/Nadeem Mumtaz**

**Registration No: FA20-bcs-016/FA20-bcs-060**

**Class: BCS-7B**

**Submitted to: Sir Bilal Bukhari**

**Date: 27-12-2023**

**QUESTION NO 02:**

**functionalities**

**1. Tokenization:** The primary functionality is to analyze the input code and tokenize each component. Based on the categorization, the code labels each component appropriately:

- Identifiers

- Symbols

- Reversed words (like "for", "while")

- Variables

- Numbers

- Pointers

- Errors

**2. Error Display:** There's also a feature to display any detected errors in the code. These errors can be displayed in a label that changes color based on whether the code compiled successfully or not.

**Initialization:** The code initializes various lists to store identifiers, symbols, reversed words, and labels for displaying parsed code.

**Parsing:** The button1\_Click method contains the logic to parse the text entered in a textbox (textBox1). It checks each token (separated by spaces) to determine its type (Identifier, Symbol, Reversed Word, Variable, Number, Pointer, or **Error).** Based on this determination, the code displays a corresponding label in a flow layout panel (flowLayoutPanel1).

**Utility Functions:** The utility functions like isIdentifier, isSymbol, and isReversedWord are presumably used to determine the type of a given token.

**Functionalities(CODE)**

public void createMemoryLabels()

{

this.Size = new Size(1304, 1087); // 559 + ((memoryList.Count + calcList.Count) \* 58)

for (int j = 0; j < labelsList.Count; j++)

{

flowLayoutPanel1.Controls.Remove(labelsList[j]);

}

for (int j = 0; j < memoryLabels.Count; j++)

{

flowLayoutPanel1.Controls.Remove(memoryLabels[j]);

}

flowLayoutPanel1.Controls.Remove(errLabel);

for (int j = 0; j < finalMemoryList.Count; j++)

{

Label label = new Label();

memoryLabels.Add(label);

}

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

{

memoryLabels[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

memoryLabels[i].ForeColor = System.Drawing.Color.White;

memoryLabels[i].Name = "newLabel" + i;

memoryLabels[i].Size = new System.Drawing.Size(1000, 36);

memoryLabels[i].Text = finalMemoryList[i].name + " = " + finalMemoryList[i].value;

memoryLabels[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(memoryLabels[i]);

}

}

public void printErrors(String error, bool isCompiled)

{

this.Size = new Size(1304, 1087); // 559 + ((memoryList.Count + calcList.Count) \* 58)

for (int j = 0; j < labelsList.Count; j++)

{

flowLayoutPanel1.Controls.Remove(labelsList[j]);

}

for (int j = 0; j < memoryLabels.Count; j++)

{

flowLayoutPanel1.Controls.Remove(memoryLabels[j]);

}

flowLayoutPanel1.Controls.Remove(errLabel);

errLabel.Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

if(isCompiled)

errLabel.ForeColor = System.Drawing.Color.LimeGreen;

else

errLabel.ForeColor = System.Drawing.Color.Red;

errLabel.Name = "errLabel";

errLabel.Size = new System.Drawing.Size(1050, 36);

errLabel.Text = error;

errLabel.Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(errLabel);

}

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

{

string[] code = textBox1.Text.Split(' ');

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

{

flowLayoutPanel1.Controls.Remove(labelsList[i]);

}

for (int j = 0; j < memoryLabels.Count; j++)

{

flowLayoutPanel1.Controls.Remove(memoryLabels[j]);

}

flowLayoutPanel1.Controls.Remove(errLabel);

for (int i = 0; i < code.Length; i++)

{

Label label = new Label();

labelsList.Add(label);

}

if (!String.IsNullOrEmpty(textBox1.Text) && code[code.Length - 1] != "")

{

this.Size = new Size(1304, 1087); //559 + (code.Length \* 16)

var regexItem = new Regex("^[a-zA-Z0-9 ]\*$");

for (int i = 0; i < code.Length; i++)

{

double test;

if (isIdentifier(code[i]))

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Identifier";

labelsList[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

else if (isSymbol(code[i]))

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Symbol";

labelsList[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

else if (isReversedWord(code[i]))

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Reversed Word";

labelsList[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

else if (!isIdentifier(code[i]) && !isSymbol(code[i]) && !isReversedWord(code[i]) && !code[i].All(char.IsDigit) && !Double.TryParse(code[i], out test) && (regexItem.IsMatch(code[i])))

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Variable";

labelsList[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

else if (!isIdentifier(code[i]) && !isSymbol(code[i]) && !isReversedWord(code[i]) && (code[i].All(char.IsDigit) || Double.TryParse(code[i], out test)) && !String.IsNullOrEmpty(code[i]))

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Number";

labelsList[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

else

{

if (code[i][0] != '\*' && code[i][code[i].Length - 1] == '\*')

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Pointer";

labelsList[i].Margin = new System.Windows.Forms.Padding(6, 6, 6, 8);

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

else

{

labelsList[i].Font = new System.Drawing.Font("Calibri", 12.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

labelsList[i].ForeColor = System.Drawing.Color.White;

labelsList[i].Name = "newLabel" + i;

labelsList[i].Size = new System.Drawing.Size(1000, 36);

labelsList[i].Text = code[i] + " -> Error";

labelsList[i].Margin = new System.Windows.Forms.Padding();

flowLayoutPanel1.Controls.Add(labelsList[i]);

}

}

}

}

}

public bool isIdentifier(String code)

{

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

{

if(code == iList[i])

{

return true;

}

}

return false;

}

public bool isSymbol(String code)

{

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

{

if (code == sList[i].symbol)

{

return true;

}

}

return false;

}

public bool isReversedWord(String code)

{

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

{

if (code == rList[i])

{

return true;

}

}

return false;

}

public bool isVariable(String code)

{

double test;

var regexItem = new Regex("^[a-zA-Z0-9 ]\*$");

if (!isIdentifier(code) && !isSymbol(code) && !isReversedWord(code) && !code.All(char.IsDigit) && !Double.TryParse(code, out test) && (regexItem.IsMatch(code)))

return true;

else

return false;

}

public bool isNumber(String code)

{

double test;

if (!isIdentifier(code) && !isSymbol(code) && !isReversedWord(code) && (code.All(char.IsDigit) || Double.TryParse(code, out test)) && !String.IsNullOrEmpty(code))

return true;

else

return false;

}

public bool isOperator(String code)

{

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

{

if (code == oList[i])

{

return true;

}

}

return false;

}

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

{

textBox1.Text = "";

this.Size = new Size(1304, 559);

}

public bool analyze1a(int i, string[] code, int inCondition)

{

MemorySaver pnn = new MemorySaver();

if (i == 0 || inCondition == 1)

{

try

{

if (isIdentifier(code[i]))

{

if (isVariable(code[i + 1]))

{

if (code[i + 2] == ";")

{

// well

pnn.name = code[i + 1];

pnn.value = "null";

memoryList.Add(pnn);

}

else

{

if (code[i + 2] == "=")

{

if (isVariable(code[i + 3]) || isNumber(code[i + 3]))

{

if (code[i + 4] == ";")

{

// then well

pnn.name = code[i + 1];

pnn.value = code[i + 3];

memoryList.Add(pnn);

}

else

{

for (; ; i += 2)

{

if (isOperator(code[i + 4])) // || isOperator(code[i + 3][0]) && code[i + 3] == "="

{

// ---->>>>> = if the doctor wants to start with =

if (isVariable(code[i + 5]) || isNumber(code[i + 5]))

{

if (code[i + 6] == ";")

{

break;

}

else

{

if (isOperator(code[i + 6]))

{

}

else

{

f = 0;

error = "near " + code[i + 6] + " Word Number " + (i + 6);

break;

}

}

}

else

{

f = 0;

error = "near " + code[i + 5] + " Word Number " + (i + 5);

break;

}

}

else

{

f = 0;

error = "near " + code[i + 4] + " Word Number " + (i + 4);

break;

}

}

if (f == 0) return false;

}

}

else

{

f = 0;

error = "near " + code[i + 3] + " Word Number " + (i + 3);

return false;

}

}

else

{

f = 0;

error = "near " + code[i + 2] + " Word Number " + (i + 2);

return false;

}

}

}

else

{

f = 0;

error = "near " + code[i + 1] + " Word Number " + (i + 1);

return false;

}

}

else

{

f = 0;

error = "near " + code[i] + " Word Number " + (i);

return false;

}

}

catch (Exception ex)

{

f = 0;

error = "Catched Error.";

return false;

}

}

else

{

return false;

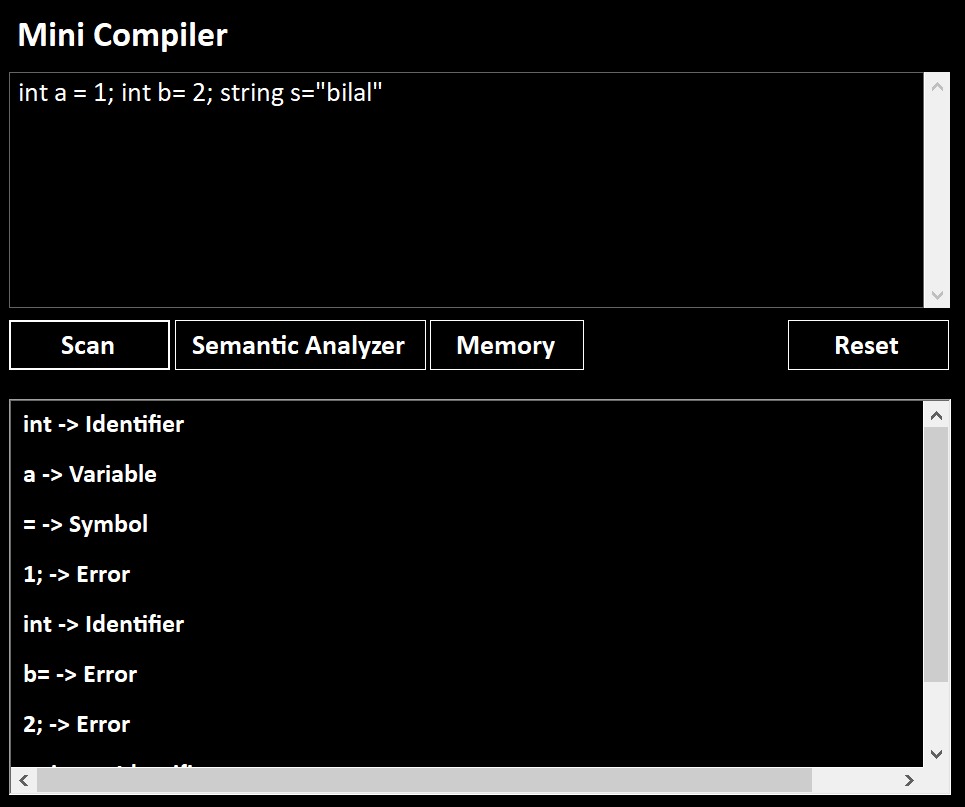
}

return true;

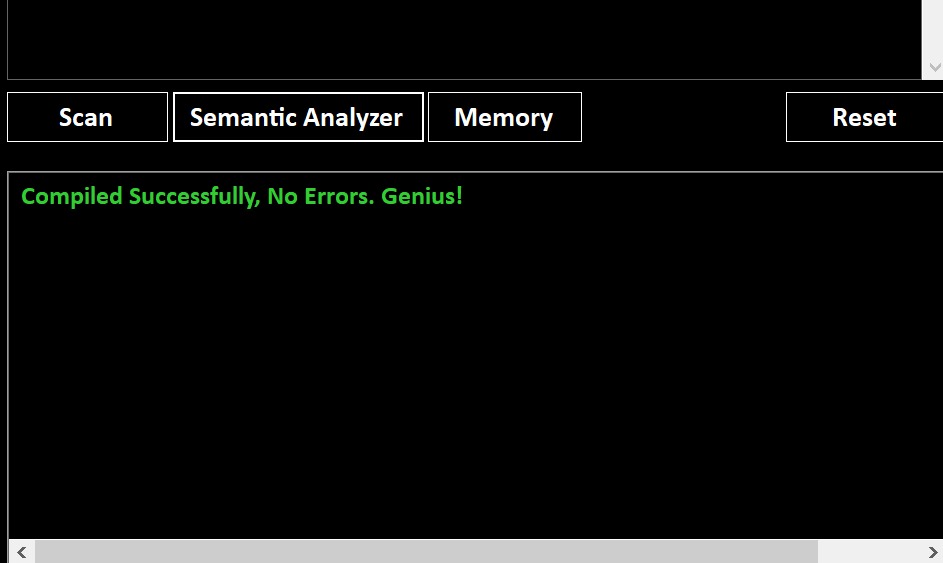
}

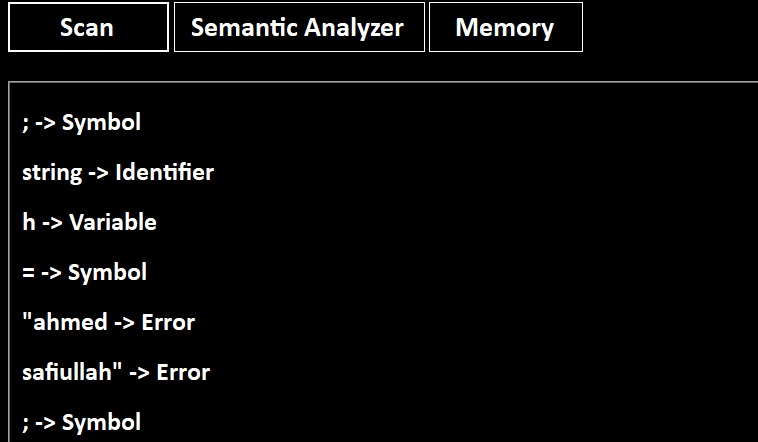
**Functionalities(OUTPUT)**





****

****

****