COMSATS Institute of Information Technology
Department of Computer Science, Attock Campus

"CC Mid Lab"

Name: **ASMA AKHTAR**
M.Saif Ullah

Reg No: **Fa20-BCS-003**
Fa20-BCS-039

Course: **CC (lab)**

PROGRAM: **BS(CS)-7A**

Course Instructor: **Mr. Bilal Haider**

Date:**28-12-2023**

QUESTION NO.2

Scanner Functionality:

Tokens are the smallest units of meaning in a programming language, and they are created by the scanner, also referred to as the lexical analyzer. It can identify operators, literals, identifiers, and keywords, among other things. By transforming the source code into a stream of tokens that the parser can process further, the scanner is essential to the first stage of the compilation process.

Semantic Analysis Functionality:

Beyond syntax, semantic analysis verifies the meaning of the source code, making it a crucial step in the compilation process. It entails making sure the software complies with the semantics and conventions of the language. The functions of the semantic analyzer include type checking, verifying that variables are used appropriately, and creating a symbol table to record identifiers and their properties. It is essential for identifying problems with the program's logical structure and guaranteeing that the generated code will function as intended.

Scanner Function:

```
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(6, 6, 6, 8);
                flowLayoutPanel1.Controls.Add(labelsList[i]);
            }
        }
```

```
        }
    }

}
```

Semantics Analysis Function:

```
private void button3_Click(object sender, EventArgs e)
{
    public bool mainAnalyze(int whichButton)
{
    string[] code = textBox1.Text.Split(' ');
    f = 1;
    error = "";
    double test;
    var regexItem = new Regex("^[a-zA-Z0-9 ]*$");

    if (code.Length >= 3)
    {

        for (int i = 0; i < code.Length; i++)
        {
            if (isIdentifier(code[i]))
            {
                analyze1a(i, code, 0);
                analyze1b(i, code, 0);
            }

            else if (isVariable(code[i]))
            {
```

```
            analyze2a(i, code, 0);
            analyze2b(i, code, 0);
        }

        else if (code[i] == "if")
        {
            analyze3a(i, code);
            analyze3b(i, code);

        }

        if (!code[i].All(char.IsLetter) || Double.TryParse(code[i], out test) ||
String.IsNullOrEmpty(code[i]) || !regexItem.IsMatch(code[i]))
        {
            if (i == 0)
            {
                f = 0;
                error = "Unexpected Error ";
                break;
            }
            else if (i > 0)
            {
                if ((code[i - 1] == ";" && code[i] != "}") || code[i - 1] == "}")
                {
                    f = 0;
                    error = "Unexpected Error ";
                    break;
                }
            }
        }
```

```
            if (f == 0) break;
        }




    }
    else
    {
        f = 0;
        error = "Error Occurred -> Too little code to compile";
    }


    if (f == 1)
    {
        if(whichButton == 3)
        {
            //MessageBox.Show("Compiled Successfully", "Run", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            printErrors("Compiled Successfully, No Errors. Perfect", true);
        }
        return true;
    }
    else
    {
        //MessageBox.Show("Error Occurred " + error, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        printErrors("Error Occurred " + error, false);
        return false;
```

```
    }
}
}
```

Memory Analysis Function:

```
private void button4_Click(object sender, EventArgs e)
 {
    string[] code = textBox1.Text.Split(' ');
    memoryList.Clear();
    calcList.Clear();
    finalMemoryList.Clear();
    if (mainAnalyze(4) || true)
    {
       for (int i = 0; i < memoryList.Count; i++)
       {
          //MessageBox.Show("" + memoryList[i].name + " = " + memoryList[i].value, "Memory
Output", MessageBoxButtons.OK, MessageBoxIcon.Information);
          MemorySaver identifier = new MemorySaver();
          identifier.name = memoryList[i].name;
          identifier.value = memoryList[i].value;
          finalMemoryList.Add(identifier);
       }

       Console.WriteLine();
       Console.WriteLine();



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

```
        {
            for(int j = 0; j < tempCalcList[i].statement.Count; j++)
            {
                //MessageBox.Show("" + tempCalcList[i].statement[j]);
            }
        }



        int value = 0;
        int f2 = 0;
        for (int i = 0; i < calcList.Count; i++)
        {
            updateValues(i);        /// <====================
            for (int j = 0; j < calcList[i].statement.Count; j++)
            {
                if (j == 1)
                {
                    try
                    {
                        if (calcList[i].statement[j] == "+")
                        {
                            value += Int32.Parse(calcList[i].statement[j - 1]) +
Int32.Parse(calcList[i].statement[j + 1]);
                        }
                        else if (calcList[i].statement[j] == "-")
                        {
                            value += Int32.Parse(calcList[i].statement[j - 1]) -
Int32.Parse(calcList[i].statement[j + 1]);
                        }
                        else if (calcList[i].statement[j] == "*")
```

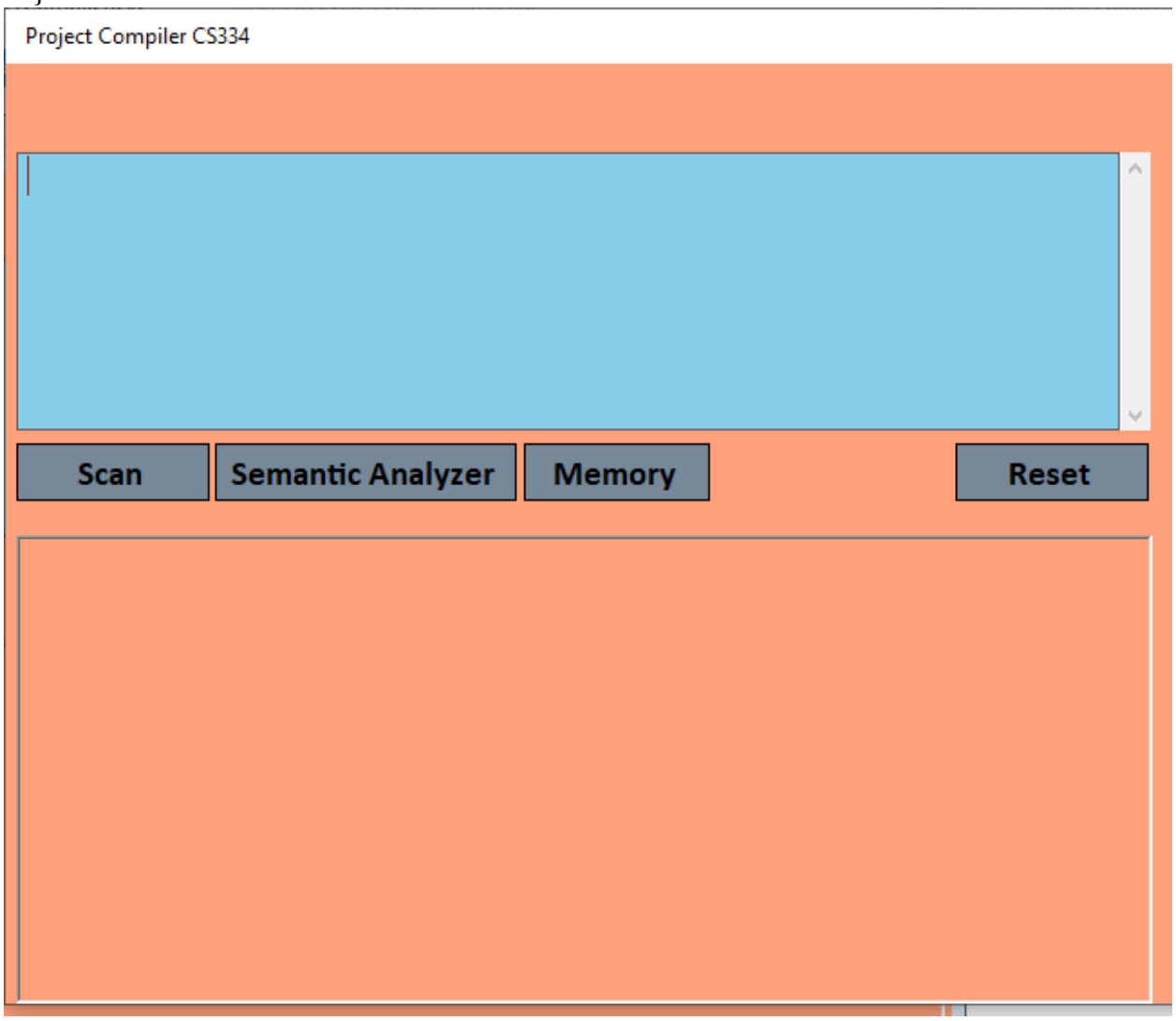
```
                {
                    value += Int32.Parse(calcList[i].statement[j - 1]) *
Int32.Parse(calcList[i].statement[j + 1]);
                }
                else if (calcList[i].statement[j] == "/")
                {
                    value += Int32.Parse(calcList[i].statement[j - 1]) /
Int32.Parse(calcList[i].statement[j + 1]);
                }
                else if (calcList[i].statement[j] == "%")
                {
                    value += Int32.Parse(calcList[i].statement[j - 1]) %
Int32.Parse(calcList[i].statement[j + 1]);
                }
            }
            catch(Exception ex)
            {
                printErrors(calcList[i].name + " Can't be Calculated because it includes one or
more unidentified variable", false);///
                f2 = 1;
            }

        }
        else
        {
            try
            {
                if (calcList[i].statement[j] == "+")
                {
                    value += Int32.Parse(calcList[i].statement[j + 1]);
                }
```

```
                else if (calcList[i].statement[j] == "-")
                {
                    value -= Int32.Parse(calcList[i].statement[j + 1]);
                }
                else if (calcList[i].statement[j] == "*")
                {
                    value *= Int32.Parse(calcList[i].statement[j + 1]);
                }
                else if (calcList[i].statement[j] == "/")
                {
                    value /= Int32.Parse(calcList[i].statement[j + 1]);
                }
                else if (calcList[i].statement[j] == "%")
                {
                    value %= Int32.Parse(calcList[i].statement[j + 1]);
                }
            }
            catch (Exception ex)
            {
                printErrors(calcList[i].name + " Can't be Calculated because" +
calcList[i].statement[j + 1] + "is unidentified variable", false);///
                f2 = 1;
            }
        }
    }
    for (int t = 0; t < memoryList.Count; t++)
    {
        if (memoryList[t].name == calcList[i].name)
        {
            memoryList[t].value = value.ToString(); // <====================
```

```
                break;
            }
        }

        //MessageBox.Show("" + calcList[i].name + " = " + value, "Memory Output",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        MemorySaver identifier = new MemorySaver();
        identifier.name = calcList[i].name;
        if (f2 == 1)
            identifier.value = "Undefined";
        else
            identifier.value = value.ToString();



        finalMemoryList.Add(identifier);
        value = 0;
    }
    // if f2 == 0     <============================
    createMemoryLabels();

    /////

    for (int i = 0; i < calcList.Count; i++)
    {
        Console.WriteLine(calcList[i].name);
        for (int j = 0; j < calcList[i].statement.Count; j++)
        {
            Console.WriteLine(calcList[i].statement[j]);
        }
    }
```

}

}


Project Compiler CS334
Scan
Semantic Analyzer
Memory
Reset

Project Compiler CS334

```
int a = 10;
int b = 20;
```

Scan | Semantic Analyzer | Memory | Reset

```
int -> Identifier

a -> Variable

= -> Symbol

10;
int -> Error
b -> Variable
```