COMPILER CONSTRUCTION LAB TERMINAL

Muhammad Hammad Nazir
Fa21-Bcs-048

QUESTION NO.2

Describe the 2 main core functions of mini compiler

Scanner Functionality:

The scanner, also known as the lexical analyzer, is responsible for breaking down the source code into tokens, which are the smallest units of meaning in the programming language. It performs tasks such as recognizing keywords, identifiers, operators, and literals. The scanner plays a crucial role in the initial phase of the compilation process by converting the source code into a stream of tokens that can be further processed by the parser.

Semantic Analysis Functionality:

Semantic analysis is a critical stage in the compilation process that goes beyond syntax and checks the meaning of the source code. It involves verifying that the program adheres to the language's semantics and rules. The semantic analyzer performs tasks such as type checking, ensuring that variables are used correctly, and building a symbol table to keep track of identifiers and their attributes. It plays a vital role in catching errors related to the logical structure of the program and ensures that the generated code will behave 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]); }
}
} }
```