Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота**

**з дисципліни**

**«Теорія формальних мов та компіляцій»**

**Тема: «Розробка генератору коду»**

Виконав:

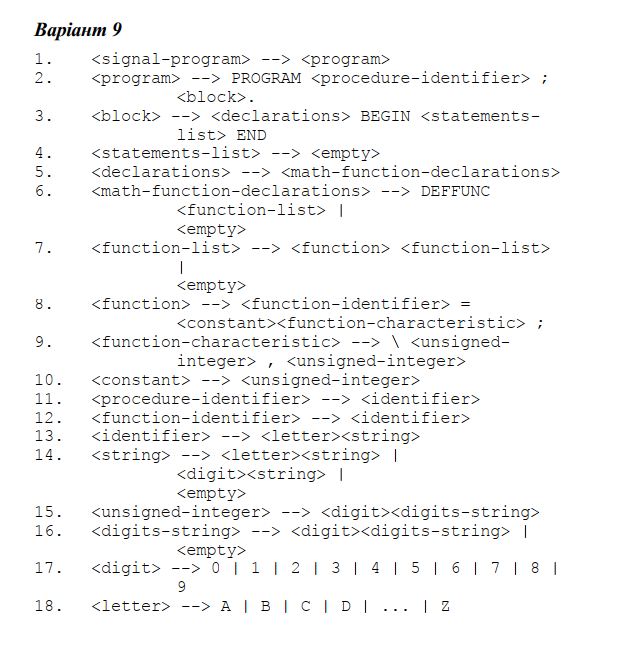
студент групи КП-91мп

Бабенко Валерій Павлович

Київ 2019

**Постановка завдання:**

Розробити програму генератору коду (ГК) для підмножини мови програмування SIGNAL.



**Лістинг**

*Файл Program.cs*

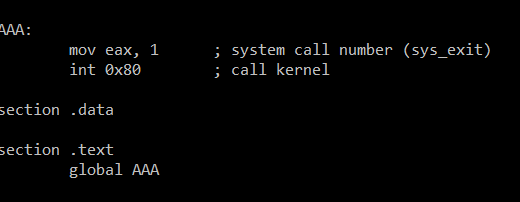
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Marchenko\_Lab3  {  class Program  {  static void Main(string[] args)  {  Console.WindowWidth = 150;  Console.WindowHeight = 35;  testFiveFiles();  }  private static void testOneFile()  {  displayFile("input5.txt");  Console.WriteLine();  LexicAnalyst la = new LexicAnalyst("input5.txt", "keywords.txt", "delimiters.txt");  la.analyze();  la.display();  Console.WriteLine();  Dictionary<string, Dictionary<string, int>> tables = la.getTables();  SyntaxAnalyst synta = new SyntaxAnalyst(tables, la.getLexemList());  synta.start();  synta.display();  Console.ReadKey();  Console.WriteLine();  if (!synta.IsError)  {  SemanticAnalyst sema = new SemanticAnalyst(tables, synta.getTree());  sema.launch();  sema.display();  Console.ReadKey();  }  }  private static void testFiveFiles()  {  for (int i = 1; i <= 5; i++)  {  Console.WriteLine("\nTEST {0}:\n", i);  string fileName = "input" + i + ".txt";  displayFile(fileName);  Console.WriteLine();  LexicAnalyst la = new LexicAnalyst(fileName, "keywords.txt", "delimiters.txt");  la.analyze();  la.display();  Console.WriteLine();  Dictionary<string, Dictionary<string, int>> tables = la.getTables();  SyntaxAnalyst synta = new SyntaxAnalyst(tables, la.getLexemList());  synta.start();  synta.display();  Console.ReadKey();  Console.WriteLine();  if (!synta.IsError)  {  SemanticAnalyst sema = new SemanticAnalyst(tables, synta.getTree());  sema.launch();  sema.display();  Console.ReadKey();  }  }  }  private static void displayFile(string filePath)  {  using (System.IO.StreamReader file = new System.IO.StreamReader(filePath))  {  string line;  while ((line = file.ReadLine()) != null)  {  Console.WriteLine(line);  }  file.Close();  }  }  }  } |

*Файл SemanticAnalyst.cs*

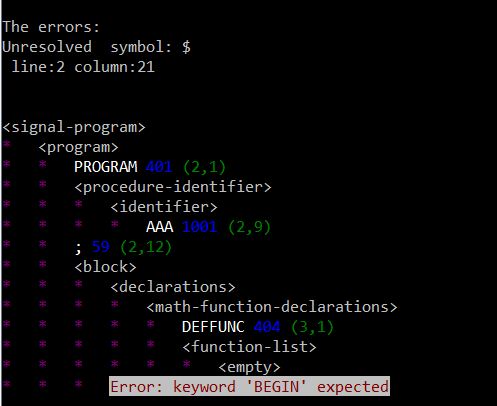
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Marchenko\_Lab3  {  public class AssemblerCode{  public string code;  public string programName;  private Dictionary<string, Dictionary<string, int>> tables;  public AssemblerCode(Dictionary<string, Dictionary<string, int>> tables)  {  this.tables = tables;  }  public void display()  {  Console.WriteLine(code);  }  }  public class SemanticAnalyst  {  private AssemblerCode ac;  private Node tree;  private List<string> usedIdentifiers;  private List<string> errors;  public SemanticAnalyst(Dictionary<string, Dictionary<string, int>> tables, Node tree)  {  this.tree = tree;  this.ac = new AssemblerCode(tables);  usedIdentifiers = new List<string>();  errors = new List<string>();  }  public void display()  {  this.ac.display();  Console.WriteLine();  Console.ForegroundColor = ConsoleColor.DarkRed;  Console.BackgroundColor = ConsoleColor.Gray;  foreach (string errStr in errors)  {  Console.WriteLine(errStr);  }  Console.ForegroundColor = ConsoleColor.Gray;  Console.BackgroundColor = ConsoleColor.Black;  }  public void launch()  {  signalProgram(this.tree);  }  private void signalProgram(Node node)  {  program(node.Children[0]);  }  private void program(Node node)  {  procedureIdentifier(node.Children[1]);  ac.code += ":\n";  ac.code += "\tmov eax, 1\t; system call number (sys\_exit)\n";  ac.code += "\tint 0x80\t; call kernel\n";  ac.code += "\n";  block(node.Children[3]);  }  private void block(Node node)  {  declarations(node.Children[0]);  ac.code += "section .text\n";  ac.code += "\tglobal ";  ac.code += ac.programName;  }  private void declarations(Node node)  {  mathFunctionDeclarations(node.Children[0]);  }  private void mathFunctionDeclarations(Node node)  {  if(node.Children[0].Nnorm != "<empty>")  {  ac.code += "section .data\n";  functionList(node.Children[1]);  ac.code += "\n";  }  }  private void functionList(Node node)  {  if (node.Children[0].Nnorm != "<empty>")  {  function(node.Children[0]);  ac.code += "\t; dt for 10 bytes\n";  functionList(node.Children[1]);  }  }  private void function(Node node)  {  ac.code += "\t";  string ident = functionIdentifier(node.Children[0]);  foreach (string usedIdent in usedIdentifiers)  {  if(ident == usedIdent)  {  string errMessage = "ERROR: one more declaration of identifier detected";  ac.code += "; " + errMessage + " ";  errors.Add(errMessage);  return;  }  }  usedIdentifiers.Add(ident);  ac.code += " dt ";  constant(node.Children[2]);  ac.code += ", ";  functionCharacteristic(node.Children[3]);  }  private void functionCharacteristic(Node node)  {  unsignedInt(node.Children[1]);  ac.code += ", ";  unsignedInt(node.Children[3]);  }  private void constant(Node node)  {  unsignedInt(node.Children[0]);  }  private void procedureIdentifier(Node node)  {  ac.programName = identifier(node.Children[0]);  usedIdentifiers.Add(ac.programName);  }  private string functionIdentifier(Node node)  {  return identifier(node.Children[0]);  }  private string identifier(Node node)  {  ac.code += node.Children[0].Lexem.value;  return node.Children[0].Lexem.value;  }  private void unsignedInt(Node node)  {  ac.code += node.Children[0].Lexem.value;  }  }  } |

**Приклади виводу**

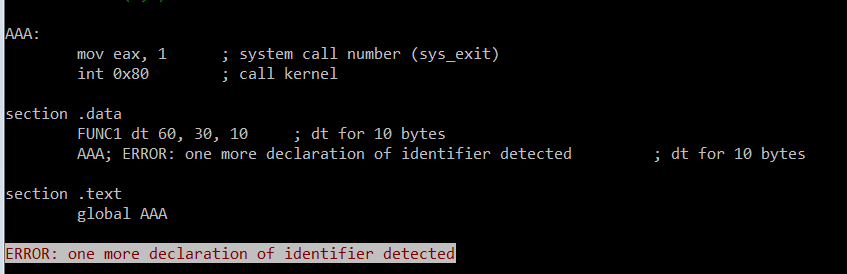
**1.**



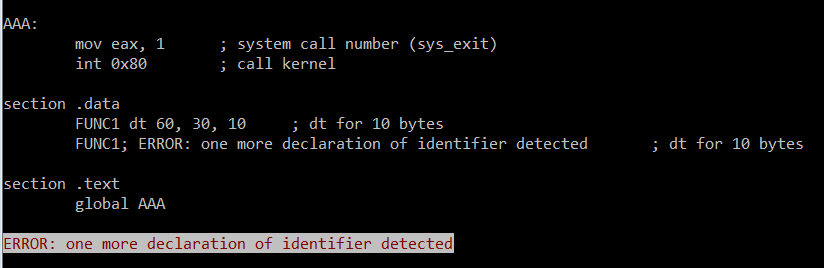
**2.**



**3.**



**4.**



**5.**

