НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кафедра обчислювальної техніки

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни «СИСТЕМНЕ ПРОГРАМУВАННЯ».

(назва дисципліни)

на тему: «Розробка компілятора підмножини команд мови Pascal, що містять арифметичні дії і дужкові форми»

Студентки 3 курсу ФІОТ групи ІВ-71

спеціальності 123 «Комп’ютерна інженерія»

Молчанової В. С.

(прізвище та ініціали)

Керівник доцент Павлов В.Г.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

Київ - 2019 рік

Зміст

[1. Вступ 3](#_Toc27048022)

[2. Блок-схема роботи компілятора 4](#_Toc27048023)

[3. Опис компілятора 5](#_Toc27048024)

[4. Лістинг модулів програми 5](#_Toc27048025)

[5. Результуючий код асемблера (за варіантом) 9](#_Toc27048026)

[6. Тестові приклади та результати їх виконання 11](#_Toc27048027)

## 1. Вступ

**Тема:** розробка компілятора підмножини команд мови С, що містять арифметичні дії та дужкові форми.

**Мета роботи:** Одержання навичок складання системних програм у відповідності до вимог завдання i підготовки комплекту документів на програми для замовника або користувача. Вивчення основних вимог до базових документів системних програм та одержання навичок оформлення таких документів на прикладах реалізації системних програм та їх документів.

**Вхідний текст (за варіантом):**

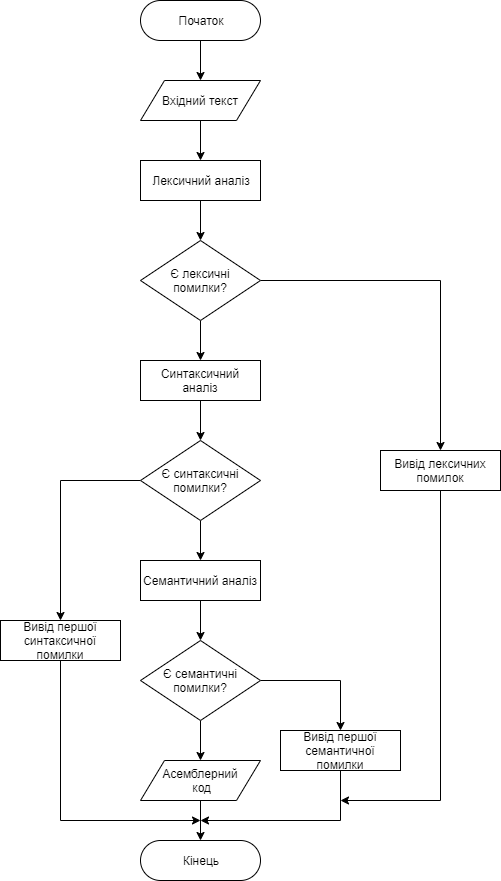
double b, a[4]; short n, d; b=2\*a[n]; b=d;

Для коректної роботи програми треба визначити змінні, тому вхідний текст було доповнено до наступного вигляду:

double b, a[4] = {1, 5, 2, 6}; short n = 2, d = 37; b=2\*a[n]; b=d;

Результатом програми, який треба вивести на екран, будемо вважати останню змінну, якій було присвоєне значення.

## 2. Блок-схема роботи компілятора



## 3. Опис компілятора

Компілятор був написаний на мові C#. Його код було розбито на наступні модулі:

* Лексичний аналізатор – перетворює вхідний текст у список лексем.
* Синтаксичний аналізатор – перетворює список лексем на синтаксичне дерево.
* Семантичний аналізатор-генератор – перевіряє синтаксичне дерево на наявність семантичних помилок та за їх відсутністю генерує код асемблера.
* Класи базових елементів для кожного з аналізаторів – описують відповідно клас лексем для лексичного, вершин для синтаксичного та змінних для семантичного аналізу.
* Допоміжні статичні класи-словники – містять статичні дані, що використовуються під час роботи компілятора, наприклад:
  + Список зарезервованих мовою С слів
  + Список операторів мови С
  + Дані про відповідність типів та їх розмірів

## 4. Лістинг модулів програми

public class SemanticAnalysis

{

public Glossary Glossary { get; }

public Dictionary<string, IVariable> Variables;

public Dictionary<int, string> Errors;

public string AssemblerCode = "";

private int \_firstErrorIndex = -1;

private int \_floatIndex = 10;

private string \_dataSection = "";

private string \_codeSection = "";

private Variable \_lastVariable;

private HashSet<int> freeRegs = new HashSet<int>{0, 1, 2, 3};

public SemanticAnalysis(SyntacticAnalysis syntacticAnalysis)

{

Errors = new Dictionary<int, string>();

Variables = new Dictionary<string, IVariable>();

Glossary = syntacticAnalysis.CurrentGlossary;

Analyze(syntacticAnalysis.Root);

}

public SemanticAnalysis(string expression, Glossary glossary)

{

Errors = new Dictionary<int, string>();

Variables = new Dictionary<string, IVariable>();

Glossary = glossary;

var lexicalAnalysis = new LexicalAnalysis(expression, glossary);

var syntacticAnalysis = new SyntacticAnalysis(lexicalAnalysis.Result, glossary);

Analyze(syntacticAnalysis.Root);

}

protected void Analyze(Node rootNode)

{

DoStatement(rootNode);

if (\_lastVariable.Type < VariableType.Float)

{

AddIntegerOutput();

}

else

{

AddFloatOutput();

}

AssemblerCode += AssemblerHelper.Prologue;

AssemblerCode += $"\tCaption db \"{\_lastVariable.Name.Substring(4, \_lastVariable.Name.Length - 4)} =\", 0\n\n";

AssemblerCode += \_dataSection;

AssemblerCode += AssemblerHelper.Main;

AssemblerCode += \_codeSection;

AssemblerCode += AssemblerHelper.Epilogue;

if (!Errors.Any())

{

Console.WriteLine("\nNo errors were found\n");

Console.WriteLine("Assembler code:\n");

Console.WriteLine(AssemblerCode+"\n\n");

var writePath = @"D:\University\SP\CourseWork\func.asm";

try

{

using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default))

{

sw.WriteLine(AssemblerCode);

}

Console.WriteLine($"Code is written to file '{writePath}'");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

else

{

PrintErrors();

}

}

protected void DoStatement(Node node)

{

if (node.Type == NodeType.Root)

{

DoStatement(node.Left ?? node.Right);

return;

}

if (node.Value == ";")

{

DoStatement(node.Left);

if (node.Right != null) DoStatement(node.Right);

return;

}

if (node.Type == NodeType.Type && node.Parent.Type != NodeType.AssignationOperator)

{

HandleType(node, out var \_var);

return;

}

if (node.Type == NodeType.AssignationOperator) HandleAssignationOperator(node);

}

protected void AddError(int index, string error)

{

Errors.Add(index, error);

if (\_firstErrorIndex == -1) \_firstErrorIndex = index;

}

private void Mov(Variable target, Variable source)

{

if (!target.IsTemp)

{

if (!source.IsTemp)

{

VarToVar(target, source);

return;

}

if (target.IsFloat)

{

if (source.IsFloat)

{

AddCode($"fld float\_const{source.Reg}");

}

else

{

if (source.IsTemp)

{

AddCode($"mov int\_buf, {source.Value}");

AddCode($"fild int\_buf");

}

else

{

AddCode($"fild {source.Name}");

}

FreeReg(source.Reg);

}

AddCode($"fstp {target.Name}");

return;

}

var targetType = AssemblerHelper.SizesInAssembler[target.Type];

var sourceReg = AssemblerHelper.Registers[targetType][source.Reg];

AddCode($"mov {target.Name}, {sourceReg}");

FreeReg(source.Reg);

}

}

private Variable AddBinAction(string action, Variable left, Variable right)

{

var resultType = TypesToBigger(left.Type, right.Type);

if (resultType > VariableType.LongLong)

{

var resultVariable = new Variable("", VariableType.Float, \_floatIndex);

AddData($"float\_const{\_floatIndex++} dd 0");

if (action == "imul") action = "mul";

if (left.IsFloat)

{

if (left.IsTemp)

{

AddCode($"fld float\_const{left.Reg}");

}

else

{

AddCode($"fld {left.Name}");

}

}

else

{

if (left.IsTemp)

{

AddCode($"mov int\_buf, {left.Value}");

AddCode($"fild int\_buf");

}

else

{

AddCode($"fild {left.Name}");

}

}

if (right.IsFloat)

{

if (right.IsTemp)

{

AddCode($"fld float\_const{right.Reg}");

}

else

{

AddCode($"fld {right.Name}");

}

}

else

{

if (right.IsTemp)

{

AddCode($"mov int\_buf, {right.Value}");

AddCode($"fild int\_buf");

}

else

{

AddCode($"fild {right.Name}");

}

}

AddCode($"f{action}");

AddCode($"fstp float\_const{resultVariable.Reg}");

FreeReg(left.Reg);

FreeReg(right.Reg);

return resultVariable;

}

string leftValue;

var leftIndex = left.Reg;

if (left.IsTemp)

{

var leftType = AssemblerHelper.SizesInAssembler[resultType];

leftValue = AssemblerHelper.Registers[leftType][left.Reg];

}

else

{

leftIndex = PickReg();

var bigType = AssemblerHelper.SizesInAssembler[resultType];

var bigReg = AssemblerHelper.Registers[bigType][leftIndex];

if (left.Type == resultType)

{

AddCode($"mov {bigReg}, {left.Name}");

}

else

{

var smallType = AssemblerHelper.SizesInAssembler[resultType];

var smallReg = AssemblerHelper.Registers[smallType][leftIndex];

AddCode($"mov {smallReg}, {left.Name}");

}

leftValue = bigReg;

}

string rightValue;

var rightIndex = right.Reg;

if (right.IsTemp)

{

var rightType = AssemblerHelper.SizesInAssembler[resultType];

rightValue = AssemblerHelper.Registers[rightType][right.Reg];

}

else

{

if (right.Type == resultType)

{

rightValue = right.Name;

}

else

{

var smallType = AssemblerHelper.SizesInAssembler[resultType];

rightIndex = PickReg();

var smallReg = AssemblerHelper.Registers[smallType][rightIndex];

AddCode($"mov {smallReg}, {right.Name}");

var bigType = AssemblerHelper.SizesInAssembler[resultType];

var bigReg = AssemblerHelper.Registers[bigType][rightIndex];

rightValue = bigReg;

}

}

AddCode($"{action} {leftValue}, {rightValue}");

FreeReg(rightIndex);

AddCode("\n");

return new Variable("", resultType, leftIndex);

}

private void AddBinAction(string action, int reg, Variable source)

{

freeRegs.Remove(reg);

source.Reg = reg;

var sourceType = AssemblerHelper.SizesInAssembler[source.Type];

var targetReg = AssemblerHelper.Registers[sourceType][reg];

var sourceValue = source.IsTemp ? source.Value : source.Name;

AddCode($"{action} {targetReg}, {sourceValue}");

}

private void VarToVar(Variable target, Variable source)

{

var regIndex = PickReg();

var targetType = AssemblerHelper.SizesInAssembler[target.Type];

var targetReg = AssemblerHelper.Registers[targetType][regIndex];

var sourceType = AssemblerHelper.SizesInAssembler[source.Type];

var sourceReg = AssemblerHelper.Registers[sourceType][regIndex];

AddCode($"mov {sourceReg}, {source.Name}");

AddCode($"mov {target.Name}, {targetReg}");

}

## 5. Результуючий код асемблера (за варіантом)

.586

.model flat, stdcall

include D:\masm32\include\msvcrt.inc

include D:\masm32\include\kernel32.inc

include D:\masm32\include\user32.inc

includelib D:\masm32\lib\msvcrt.lib

includelib D:\masm32\lib\kernel32.lib

includelib D:\masm32\lib\user32.lib

.data

int\_fmt db " % d",10, 0

flt\_fmt db " % f",10, 0

buff dq 0

int\_buf dd 0

Caption db "b =", 0

var\_b dd ? ; Double b

arr\_a dd 4 dup(?) ; Double \*a[4]

var\_n dw ? ; Short n

var\_d dw ? ; Short d

float\_const10 dd 0

.code

main:

xor eax, eax

xor ebx, ebx

mov bx, 1

mov int\_buf, 1

fild int\_buf

fstp dword ptr[arr\_a + 0]

xor ebx, ebx

mov bx, 5

mov int\_buf, 5

fild int\_buf

fstp dword ptr[arr\_a + 4]

xor ebx, ebx

mov bx, 2

mov int\_buf, 2

fild int\_buf

fstp dword ptr[arr\_a + 8]

xor ebx, ebx

mov bx, 6

mov int\_buf, 6

fild int\_buf

fstp dword ptr[arr\_a + 12]

xor ebx, ebx

mov bx, 2

mov var\_n, bx

xor ebx, ebx

mov bx, 37

mov var\_d, bx

xor ebx, ebx

mov bx, var\_n

imul ebx, 4

xor ecx, ecx

mov cx, 2

mov int\_buf, 2

fild int\_buf

fld dword ptr[arr\_a + ebx]

fmul

fstp float\_const10

fld float\_const10

fstp var\_b

xor ebx, ebx

mov bx, var\_d

mov var\_b, ebx

fld var\_b

fstp buff

invoke crt\_printf, addr Caption

invoke crt\_printf, addr flt\_fmt, buff

invoke ExitProcess, 0

end main

## 6. Тестові приклади та результати їх виконання

**Приклад 1:** “double b, a[4] = {1, 5, 2, 6}; short n = 2, d = 37; b=2\*a[n]; b=d;”

**Результат:**

PS D:\University\SP\CourseWork> ./func.exe

b = 37

**Приклад 2:** “double b, a[4] = {1, 5, 2, 6}; short n = 2, d = 37; b=2\*a[n]; d=b;”

**Результат:**

Entered expression: double b, a[4] = {1,5,2,6};short n = 2, d = 37; b=2\*a[n]; d=b;

^

SEMANTIC ERROR: Cannot convert 'Double' to 'Short'

**Приклад 3:** “double b = 2.4 \* (1.1 + 1.3);”

**Результат:**

PS D:\University\SP\CourseWork> ./func.exe

b = 5.760000