НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА**

з дисципліни **«**Системне програмування 2**»**

Варіант 3

Виконала:

студентка 3 курсу гр. ІО-64

Бровченко А. В.

Перевірив:

Павлов В. Г.

Київ 2018 р.

**Зміст завдання:** За основу береться результат виконання лабораторної роботи 6, у якої крім лексичного та синтаксичного аналізу здійснюється семантичний аналіз. У рамках РГР усі перелічені етапи компіляції доповнюються генерацією ассемблерного коду, який виконує дії, за допомогою виразу, позначеного у варіанті завдання. Для перевірки працездатності цього коду він повинен бути вбудований у програму на мові високого рівня (мова програмування вибирається самостійно) та уявляти собою лише певну частину обчислення. Усі інші необхідні дії для підготовки та виконання вказаних у завданні обчислень, а також вивід результатів цих обчислень виконуються у програмі на мові високого рівня. При внесенні змін у вираз, який визначається за варіантом завдання, повинен бути згенерований новий ассемблерний код.

Таким чином, у межах РГР повинні бути розроблені дві програми:

* одна, яка генерує ассемблерний код для заданого виразу, або його варіантів;
* друга, яка використовує згенерований код як частку загальних обчислень, які мають закінчений вигляд та мають бути перевірені не менш ніж п’яти контрольних прикладах з порівнянням з контрольними результатами.

**Варіант:** 

**Лістинг програми 1:**

**Analizer.py**

**from** re **import** search  
**from** Stack **import** Stack  
**from** Utils **import** is\_var, brackets\_checker, is\_num, tokens  
**from** copy **import** deepcopy  
  
**class** Var(object):  
 **def** \_\_init\_\_(self, var\_type, name, value=**None**):  
 self.var\_type = var\_type  
 self.name = name  
 self.value = value  
  
 **def** \_\_repr\_\_(self):  
 **return "{0} {1} ({2})"**.format(self.var\_type, self.name, self.value)  
  
 **def** \_\_str\_\_(self):  
 **return "{0} {1} ({2})"**.format(self.var\_type, self.name, self.value)  
  
**class** Array(object):  
 **def** \_\_init\_\_(self, var\_type, name, length=0, values=[]):  
 self.var\_type = var\_type  
 self.name = name  
 self.length = length  
 self.values = values[:]  
  
 **def** \_\_repr\_\_(self):  
 **return "{0} {1} ({2})"**.format(self.var\_type, self.name, self.values)  
  
 **def** \_\_str\_\_(self):  
 **return "{0} {1} ({2})"**.format(self.var\_type, self.name, self.values)  
  
**def** is\_valid\_var\_name(s):  
 **if** search(**r'\W[^,]]'**, s):**return False  
  
 if** search(**r'[^A-Za-z]'**, s[0]):**return False  
  
 return True****def** count\_value(expr, type):value = postfix\_eval(infix\_to\_postfix(expr))  
  
 **if** type == **"float"**:  
 value = float(value)  
 **elif** type == **"int"**:  
 value = int(value)**return** value  
  
**def** infix\_to\_postfix(infixexpr):  
 prec = {}  
 prec[**"\*"**] = 3  
 prec[**"/"**] = 3  
 prec[**"+"**] = 2  
 prec[**"-"**] = 2  
 prec[**"("**] = 1  
 op\_stack = Stack()  
 postfix\_list = []  
 token\_list = infixexpr.split()  
  
 **for** token **in** token\_list:  
 **if** search(**r"[A-Za-z0-9]"**, token):  
 postfix\_list.append(token)  
 **elif** token == **'('**:  
 op\_stack.push(token)  
 **elif** token == **')'**:  
 top\_token = op\_stack.pop()  
 **while** top\_token != **'('**:  
 postfix\_list.append(top\_token)  
 top\_token = op\_stack.pop()  
 **else**:  
 **while** (**not** op\_stack.isEmpty()) **and** \  
 (prec[op\_stack.peek()] >= prec[token]):  
 postfix\_list.append(op\_stack.pop())  
 op\_stack.push(token)  
  
 **while not** op\_stack.isEmpty():  
 postfix\_list.append(op\_stack.pop())  
 **return " "**.join(postfix\_list)  
  
**def** postfix\_eval(postfix\_expr):  
 operand\_stack = Stack()  
 token\_list = postfix\_expr.split()  
  
 **for** token **in** token\_list:  
 **if** token **in "0123456789"**:  
 operand\_stack.push(float(token))  
 **elif** token **in "+-\*/"**:  
 operand2 = operand\_stack.pop()  
 operand1 = operand\_stack.pop()  
 result = do\_math(token, operand1, operand2)  
 operand\_stack.push(result)  
 **else**:  
 **if** token.find(**"["**) != -1:  
 arr\_name = token[:token.index(**"["**)]  
 ind = token[token.index(**"["**) + 1]  
 **if** ind **in** var\_dict.keys():  
 ind = var\_dict[ind].value  
 ind = int(ind)  
 **if** ind >= var\_dict[arr\_name].length:  
 print(**"Error: Index out of bounds"**)  
 operand\_stack.push(float(var\_dict[arr\_name].values[ind]))  
 **else**:  
 operand\_stack.push(float(var\_dict[token].value))  
 **return** operand\_stack.pop()  
  
**def** check\_syntax(input\_string):  
 input\_list = input\_string.split(**" "**)  
 has\_errors = **False  
  
 for** s **in** input\_list:  
 **if** s **not in** tokens.keys() **and not** search(**"[A-Za-z0-9]"**, s):  
 print(**"Error: Unexpected '%s'"** % s)  
 has\_errors = **True  
  
 for** i **in** range(len(input\_list)):  
 **if** input\_list[i] == **":="**:  
 **if not** is\_var(input\_list[i - 1]):  
 print(**"Error: Incorrect usage of 'assign' operator"**)  
 has\_errors = **True  
  
 if** input\_list[i] **in** [**"+"**, **"-"**, **"/"**, **"\*"**, **">"**, **"<"**, **">="**, **"<="**]:  
 **if not** is\_var(input\_list[i - 1]) **and not** is\_num(input\_list[i - 1]):  
 print(**"Error: Incorrect usage of '%s' operator"** % input\_list[i])  
 has\_errors = **True  
 elif not** is\_num(input\_list[i + 1]) **and not** is\_var(input\_list[i + 1]):  
 print(**"Error: Incorrect usage of '%s' operator"** % input\_list[i])  
 has\_errors = **True  
  
 if** search(**"\["**, input\_list[i]):  
 **for** j **in** range(len(input\_list[i])):  
 **if** input\_list[i][j] == **"[" and** (**not** is\_var(input\_list[i][j - 1]) **or** (**not** is\_var(input\_list[i][j + 1])\  
 **and not** is\_num(input\_list[i][j + 1]))):  
 print(**"Error: Incorrect usage of array indexing"**)  
 has\_errors = **True  
  
 if not** brackets\_checker(input\_string):  
 print(**"Error: Unbalanced brackets"**)  
 has\_errors = **True  
  
 return** has\_errors  
  
**def** do\_math(op, op1, op2):  
 **if** op == **"\*"**:  
 **return** op1 \* op2  
 **elif** op == **"/"**:  
 **return** op1 / op2  
 **elif** op == **"+"**:  
 **return** op1 + op2  
 **else**:  
 **return** op1 - op2  
  
**def** to\_asm(inp: str):  
 **global** var\_dict  
 **if not** check\_syntax(inp):  
 **try**:  
 inp = inp[:-1]  
 var\_dict = dict()  
 types = [**"float"**, **"int"**, **"char"**]  
 inp\_arr = inp.split(**"; "**)file = open(**"assm.txt"**, **"w+"**)  
 counter = 0  
  
 **for** line **in** inp\_arr:  
 line\_tokens = line.split(**" "**)  
 line\_tokens1 = line.split(**" "**)  
 **if "+=" in** line\_tokens:  
 line\_tokens = [line\_tokens1[0], **":="**, line\_tokens1[0], **"+"**, line\_tokens1[2]]  
 line = line\_tokens1[0] + **" := "** + line\_tokens1[0] + **" + "** + line\_tokens1[2]  
 **if "-=" in** line\_tokens:  
 line\_tokens = [line\_tokens1[0], **":="**, line\_tokens1[0], **"-"**, line\_tokens1[2]]  
 line = line\_tokens1[0] + **" := "** + line\_tokens1[0] + **" - "** + line\_tokens1[2]  
 **if** line\_tokens[0] **in** types **and** line.find(**":="**) == -1:  
 **for** i **in** range(1, len(line\_tokens)):  
 **if** is\_valid\_var\_name(line\_tokens[i]):  
 **if** line\_tokens[i][len(line\_tokens[i]) - 1] == **","**:  
 name = line\_tokens[i][:-1]  
 **else**:  
 name = line\_tokens[i]  
 var\_dict[name] = Var(line\_tokens[0], name)  
 **elif** line\_tokens[i].find(**"["**) != -1:  
 **if** is\_valid\_var\_name(line\_tokens[i][:line\_tokens[i].index(**"["**)]):  
 **if** line\_tokens[i][len(line\_tokens[i]) - 1] == **","**:  
 name = line\_tokens[i][:line\_tokens[i].index(**"["**) + 1]  
 **else**:  
 name = line\_tokens[i][:line\_tokens[i].index(**"["**)]length = line\_tokens[i][line\_tokens[i].index(**"["**) + 1]  
 var\_dict[name] = Array(line\_tokens[0], name, int(length))  
 var\_dict[name].values = [**None for** i **in** range(int(length))]  
  
 **elif** line\_tokens[0] **in** types **and** line.find(**":="**) != -1:  
 var\_dict[line\_tokens[1]] = Var(line\_tokens[0], line\_tokens[1], count\_value(line[line.index(**":="**) + 3:], line\_tokens[0]))  
 **elif** line\_tokens[0] **not in** types **and** line.find(**":="**) != -1**and** line\_tokens[0] **in** var\_dict.keys():  
 var\_dict[line\_tokens[0]].value = count\_value(line[line.index(**":="**) + 3:], var\_dict[line\_tokens[0]].var\_type)  
 **if '+' in** line:  
 file.writelines(**'; {} \n'**.format(line))  
 file.writelines(**'mov eax, {}\n'**.format(line\_tokens[2]))  
 **if** line\_tokens[-1][line\_tokens[-1].index(**'['**)+1].isdigit():  
 file.writelines(**'add eax, dword ptr[{}+{}]\n'**.format(line\_tokens[-1][0], line\_tokens[-1][  
 line\_tokens[-1].index(**'['**)+1]))  
 file.writelines(**'mov {}, eax\n'**.format(line\_tokens[0]))  
 **else**:  
 arg2 = str(var\_dict[line\_tokens[-1].replace(**';'**, **''**)[-2]]).split(**' '**)[-1][-2]  
 file.writelines(**'add eax, dword ptr[{}+{}]\n'**.format(line\_tokens[-1][0], str(int(arg2)\*4)))  
 file.writelines(**'mov {}, eax\n'**.format(line\_tokens[0]))  
 **elif '-' in** line:  
 file.writelines(**'; {} \n'**.format(line))  
 file.writelines(**'mov eax, {}\n'**.format(line\_tokens[2]))  
 **if** line\_tokens[-1][line\_tokens[-1].index(**'['**)+1].isdigit():  
 file.writelines(**'add eax, dword ptr[{}+{}]\n'**.format(line\_tokens[-1][0], line\_tokens[-1][  
 line\_tokens[-1].index(**'['**)+1]))  
 file.writelines(**'mov {}, eax\n'**.format(line\_tokens[0]))  
 **else**:  
 arg2 = str(var\_dict[line\_tokens[-1].replace(**';'**, **''**)[-2]]).split(**' '**)[-1][-2]  
 file.writelines(**'sub eax, dword ptr[{}+{}]\n'**.format(line\_tokens[-1][0], str(int(arg2)\*4)))  
 file.writelines(**'mov {}, eax\n'**.format(line\_tokens[0]))  
 **else**:  
 file.writelines(**'; {} \n'**.format(line))  
 file.writelines(**'mov eax, {}\n'**.format(line\_tokens[-1]))  
 file.writelines(**'mov {}, eax\n'**.format(line\_tokens[0]))  
 **elif** line\_tokens[0].find(**"["**):  
 arr\_name = line\_tokens[0][:line\_tokens[0].index(**"["**)]  
 ind = line\_tokens[0][line\_tokens[0].index(**"["**) + 1]  
 **if** search(**'[0-9]+[.][0-9]+'**, ind):  
 print(**'Error: Index must be int'**)  
 **raise** IndexError  
 **if** ind **in** var\_dict.keys():  
 ind = var\_dict[ind].value  
 ind = int(ind)  
 **if** ind >= var\_dict[arr\_name].length:  
 print(**"Error: Index out of bounds"**)  
 **raise** IndexError  
 var\_dict[arr\_name].values[ind] = count\_value(line[line.index(**":="**) + 3:], var\_dict[arr\_name].var\_type)  
 file.writelines(**'; {} \n'**.format(line))  
 file.writelines(**'mov eax, {}\n'**.format(line\_tokens[-1]))  
 file.writelines(**'mov dword ptr[{}+{}], eax\n'**.format(line\_tokens[0][0], str(4\*counter)))  
 counter += 1  
 file.close()  
 **return** var\_dict  
 **except** (TypeError, IndexError, ValueError):  
 print(**"Syntax error"**)  
 **else**:  
 print(**"Syntax error"**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
  
 inp = **"float b, a[4];"** \  
 **" int n;"** \  
 **" a[0] := 1;"** \  
 **" a[1] := 2;"** \  
 **" a[2] := 3;"**\  
 **" a[3] := 4;"**\  
 **" b := 5;"** \  
 **" n := 2;"** \  
 **" b += a[n];"** \  
  
 res = to\_asm(inp)  
 print(res)

**Utils.py**

**from** re **import** search  
**from** Stack **import** Stack  
  
tokens = {**":="**: **"Assign"**, **"-"**: **"Minus"**, **"+"**: **"Plus"**,  
 **"["**: **"Left bracket"**, **"]"**: **"Right Bracket"**,  
 **";"**: **"Semicolon"**, **"="**: **"Equals"**,  
 **">"**: **"Greater"**, **"<"**: **"Lower"**,  
 **">="**: **"Greater or equals"**, **"<="**: **"Lower or equals"**,  
 **"\*"**: **"Multiply"**, **"/"**: **"Divide"**, **"%"**: **"Modular"**}

**def** is\_var(ss):  
 res = **False  
 if** len(ss) == 0:  
 res = **False  
 elif** search(**"[A-Za-z]"**, ss[0]):  
 res = **True  
 elif not** search(**"[A-Za-z0-9\_]\*"**, ss[1:]):  
 res = **False  
 return** res  
  
  
**def** is\_num(ss):  
 res = **False  
 if** len(ss) == 0:  
 res = **False  
 elif** search(**"[0-9]"**, ss[0]) **and not** search(**"[A-Za-z]"**, ss):  
 res = **True  
 return** res  
  
  
**def** brackets\_checker(sss):  
 st = Stack()  
 balanced = **True** index = 0  
 **while** index < len(sss) **and** balanced:  
 symbol = sss[index]  
 **if** symbol == **"["**:  
 st.push(symbol)  
 **elif** symbol == **"]"**:  
 **if** st.isEmpty():  
 balanced = **False  
 else**:  
 st.pop()  
 index += 1  
 **return** balanced **and** st.isEmpty()  
  
  
**def** print\_table(inp\_list):  
 **for** string **in** inp\_list:  
 **if** string **in** tokens:  
 print(string + **" is a token '%s'"** % tokens[string])  
 **else**:  
 has\_tokens = **False** has\_numbers = **False  
 for** c **in** string:  
 **if** c **in** tokens:  
 has\_tokens = **True  
 if** search(**r"[0-9]"**, c):  
 has\_numbers = **True  
 if** has\_tokens:  
 **for** c **in** string:  
 **if** c **in** tokens:  
 print(c + **" is a token '%s'"** % tokens[c])  
 **elif** search(**r"[A-Za-z]"**, c):  
 print(c + **" is a var '%s'"** % c)  
 **elif** search(**r"[0-9]"**, c):  
 print(c + **" is a const '%s'"** % c)  
 **elif** has\_numbers:  
 print(string + **" is a const '%s'"** % string)  
 **else**:  
 print(string + **" is a var '%s'"** % string)

**Stack.py**

**class** Stack:  
 **def** \_\_init\_\_(self):  
 self.items = []  
  
 **def** isEmpty(self):  
 **return** self.items == []  
  
 **def** push(self, item):  
 self.items.append(item)  
  
 **def** pop(self):  
 **return** self.items.pop()  
  
 **def** peek(self):  
 **return** self.items[len(self.items) - 1]  
  
 **def** size(self):  
 **return** len(self.items)

**Лістинг програми 2:**

**Compile.cpp**

#include <omp.h>

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n;

int b, a[4];

\_\_asm {

; a[0] : = 1

mov eax, 1

mov dword ptr[a + 0], eax

; a[1] : = 2

mov eax, 2

mov dword ptr[a + 4], eax

; a[2] : = 3

mov eax, 3

mov dword ptr[a + 8], eax

; a[3] : = 4

mov eax, 4

mov dword ptr[a + 12], eax

; b: = 5

mov eax, 5

mov b, eax

; n: = 3

mov eax, 3

mov n, eax

; b: = b + a[n]

mov eax, b

add eax, dword ptr[a + 12]

mov b, eax

}

cout << b << endl;

cout << "Press Enter...";

string t;

getline(cin, t);

}

**Контрольні приклади**

**Приклад 1:**

Input: **"float b, a[4]; int n; a[0] := 1; a[1] := 2; a[2] := 3; a[3] := 4; b := 5; n := 2; b += a[n];"**

Assembler:

; a[0] := 1   
mov eax, 1  
mov dword ptr[a+0], eax  
; a[1] := 2   
mov eax, 2  
mov dword ptr[a+4], eax  
; a[2] := 3   
mov eax, 3  
mov dword ptr[a+8], eax  
; a[3] := 4   
mov eax, 4  
mov dword ptr[a+12], eax  
; b := 5   
mov eax, 5  
mov b, eax  
; n := 3   
mov eax, 3  
mov n, eax  
; b := b + a[n]   
mov eax, b  
add eax, dword ptr[a+12]  
mov b, eax

Output:

9

Press Enter...

**Приклад 2:**

Input: **"float b, a[4]; int n; a[0] := 1; a[1] := 2; a[2] := 3; a[3] := 4; b := 5; n := 2; b -= a[n];"**

Assembler:

; a[0] := 1   
mov eax, 1  
mov dword ptr[a+0], eax  
; a[1] := 2   
mov eax, 2  
mov dword ptr[a+4], eax  
; a[2] := 3   
mov eax, 3  
mov dword ptr[a+8], eax  
; a[3] := 4   
mov eax, 4  
mov dword ptr[a+12], eax  
; b := 5   
mov eax, 5  
mov b, eax  
; n := 2   
mov eax, 2  
mov n, eax  
; b := b - a[n]   
mov eax, b  
sub eax, dword ptr[a+8]  
mov b, eax

Output:

2

Press Enter...

**Приклад 3:**

Input: **"float b, a[4]; int n; a[0] := 23; a[1] := 34; a[2] := 34; a[3] := 23; b := 5; n := 1; b -= a[n];"**

Assembler:

; a[0] := 23   
mov eax, 23  
mov dword ptr[a+0], eax  
; a[1] := 34   
mov eax, 34  
mov dword ptr[a+4], eax  
; a[2] := 34   
mov eax, 34  
mov dword ptr[a+8], eax  
; a[3] := 23   
mov eax, 23  
mov dword ptr[a+12], eax  
; b := 5   
mov eax, 5  
mov b, eax  
; n := 1   
mov eax, 1

mov n, eax  
; b := b - a[n]   
mov eax, b  
sub eax, dword ptr[a+4]  
mov b, eax

Output:

-29

Press Enter...

**Висновок**

Під час виконання розрахунково-графічної роботи було розроблено дві програми. Перша генерує асемблерну вставку. Асемблерний код генерується при кожному запуску програми. Друга – виконує кінцеві розрахунки програми.