**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

**РАЗРАБОТКА КОМПИЛЯТОРА**

Пояснительная записка

На 18 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов |
| Исполнитель |  | студент гр. ИБ-118 Д.А Губарев |

**Владимир 2021**

**Оглавление**

[**Аннотация** 3](#_Toc72675939)

[**1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА** 4](#_Toc72675940)

[1.1 Основные требования 4](#_Toc72675941)

[1.2 Лексический анализатор 5](#_Toc72675942)

[1.3 Синтаксический анализатор 6](#_Toc72675943)

[1.4 Таблица символов 13](#_Toc72675944)

[1.5 Промежуточный код 14](#_Toc72675945)

[1.6 Генерация кода ассемблера 15](#_Toc72675946)

[**2** **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ** 16](#_Toc72675947)

# **Аннотация**

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Python с использованием библиотеки ply. Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов виде кода для MIPS-ассемблера.

Разработка компилятора подмножества процедурного языка в ассемблер состоит из следующих стадий:

* построение лексического анализатора;
* построение синтаксического анализатора;
* построение генератора MIPS-кода

# **1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

## Основные требования

Разработка будет производиться в соответствии со следующими требованиями:

* Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться индентация программы;
3. Должны поддерживаться комментарии любой длины;
4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.

* Требования к операторам:

1. Оператор присваивания;
2. Арифметика (\*, /, +, -, >, <, =);
3. Логические операторы (И, ИЛИ, НЕ);
4. Условный оператор (ЕСЛИ);
5. Оператор цикла (while);
6. Базовый вывод (строковой литерал, переменная);
7. Типы (целочисленный, вещественный).

## Лексический анализатор

Лексический анализатор является первой фазой компилятора. Он преобразует входной поток символов в поток токенов.

Грамматика языка реализована с использованием библиотеки ply.lex.

Сам язык являеться чем то смешенным между python и pascal.

Зарезервированные слова: Аналоги в других языках:

1. def def
2. return return
3. and and
4. or or
5. not not
6. print print
7. while while
8. if if
9. break break
10. continue continue
11. int integer
12. real float
13. vars var
14. str string
15. print print
16. { begin
17. } end

А также символы  *=, >, :, :=, (, ), ; , + , - , \* , /, <,*

## Синтаксический анализатор

Второй стадией компилятора является синтаксический анализ. На вход синтаксическому анализатору подаётся набор токенов из лексического анализатора. На основе грамматики языка строится абстрактное дерево разбора грамматики.

КС-грамматика языка:

S' -> program  
prog -> VARS dec\_list OF stmt\_list CF

prog -> VARS dec\_list def\_list OF stmt\_list CF  
def\_list -> def

def\_list -> def\_list SC def

def -> DEF ID OP dec\_list CP OF stmt\_list\_def CF

def -> DEF ID OP dec\_list CP OF VARS dec\_list stmt\_list\_def CF

defstmt -> ID OP args CP

args -> arg

args -> args SC arg

arg -> ID  
arg -> NUMBER\_INT  
arg -> NUMBER\_REAL  
arg -> OP exp CP

dec\_list -> dec

dec\_list -> dec\_list SC dec

dec -> type DP OP id\_list CP

type -> INT  
type -> REAL  
type -> STRING

id\_list -> ID  
id\_list -> id\_list CM ID

stmt\_list -> stmt  
stmt\_list -> stmt\_list SC stmt

stmt -> prisv  
stmt -> print  
stmt -> while  
stmt -> if

stmt\_list\_if -> stmt\_if  
stmt\_list\_if -> stmt\_list\_if SC stmt\_if

stmt\_if -> prisv  
stmt\_if -> print  
stmt\_if -> while  
stmt\_if -> if  
stmt\_if -> CONTINUE  
stmt\_if -> BREAK

stmt\_list\_def -> stmt\_def  
stmt\_list\_def -> stmt\_list\_def SC stmt\_def

stmt\_def -> prisv  
stmt\_def -> print  
stmt\_def -> while  
stmt\_def -> if  
stmt\_def -> return

return -> RETURN exp

prisv -> ID PRISV exp  
prisv -> ID PRISV STRING

exp -> term  
exp -> exp SUM term  
exp -> exp SUB term

term -> factor  
term -> term MUL factor  
term -> term DIV factor

factor -> defstmt  
factor -> ID  
factor -> NUMBER\_INT  
factor -> NUMBER\_REAL  
factor -> OP exp CP

print -> PRINT OP exp CP  
print -> PRINT OP STRING CP

while -> WHILE bool\_exp OF stmt\_list CF

if -> IF bool\_exp OF stmt\_list\_if CF

bool\_exp -> bool\_exp OR bool\_exp\_term  
bool\_exp -> bool\_exp\_term  
bool\_exp -> NOT bool\_exp  
bool\_exp -> bool

bool\_exp\_term -> bool\_exp\_term AND bool  
bool\_exp\_term -> bool

bool -> OP exp RAVNO exp CP  
bool -> OP exp MORE exp CP  
bool -> OP exp LESS exp CP

**Объявление переменных**

Блок объявления переменных представляется следующим правилом КС-грамматики:

prog -> VARS dec\_list OF stmt\_list CF

prog -> VARS dec\_list def\_list OF stmt\_list CF

dec\_list -> dec

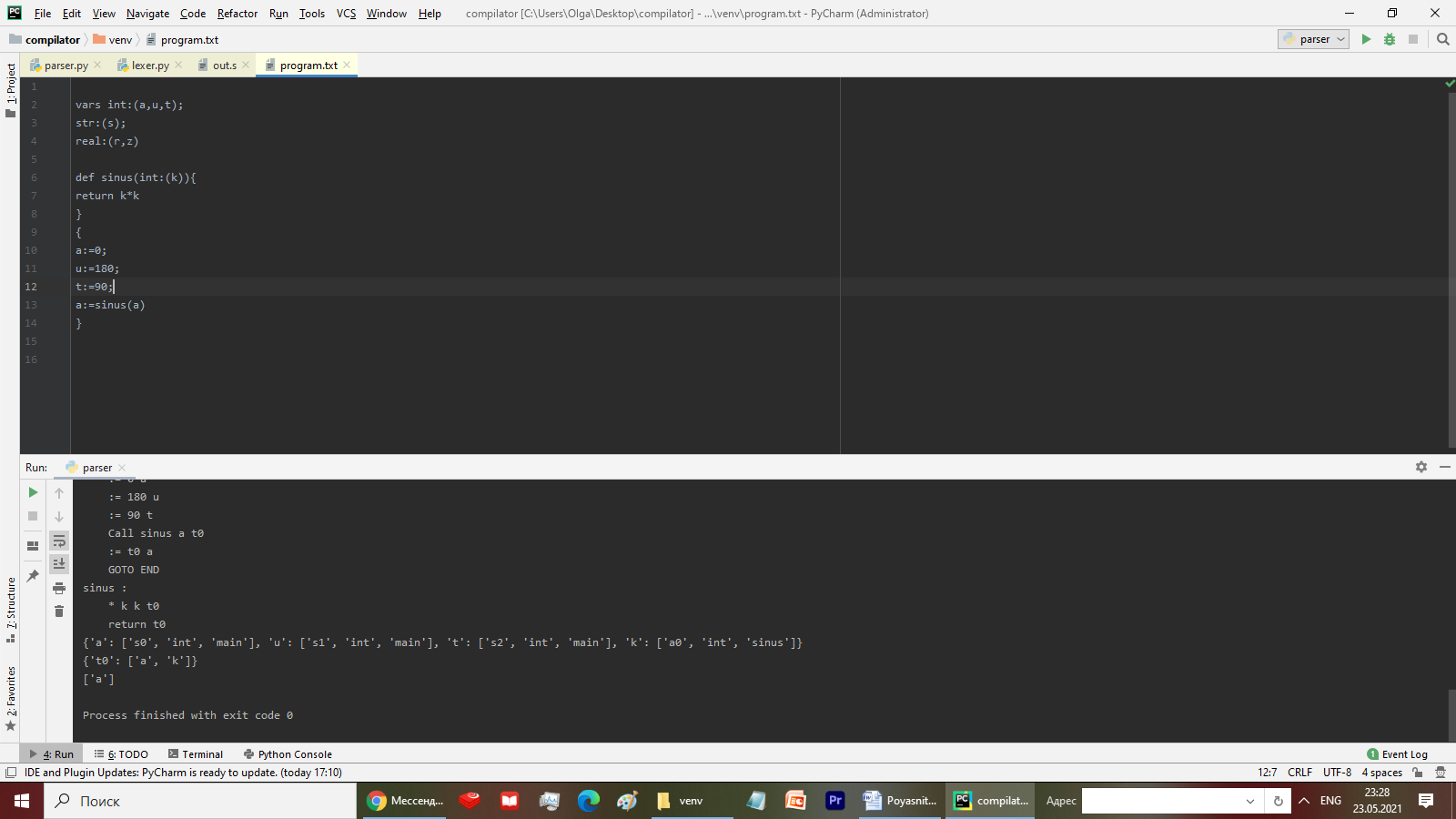
dec\_list -> dec\_list SC dec

dec -> type DP OP id\_list CP

type -> INT  
type -> REAL  
type -> STRING

id\_list -> ID  
id\_list -> id\_list CM ID

Пример в программе:



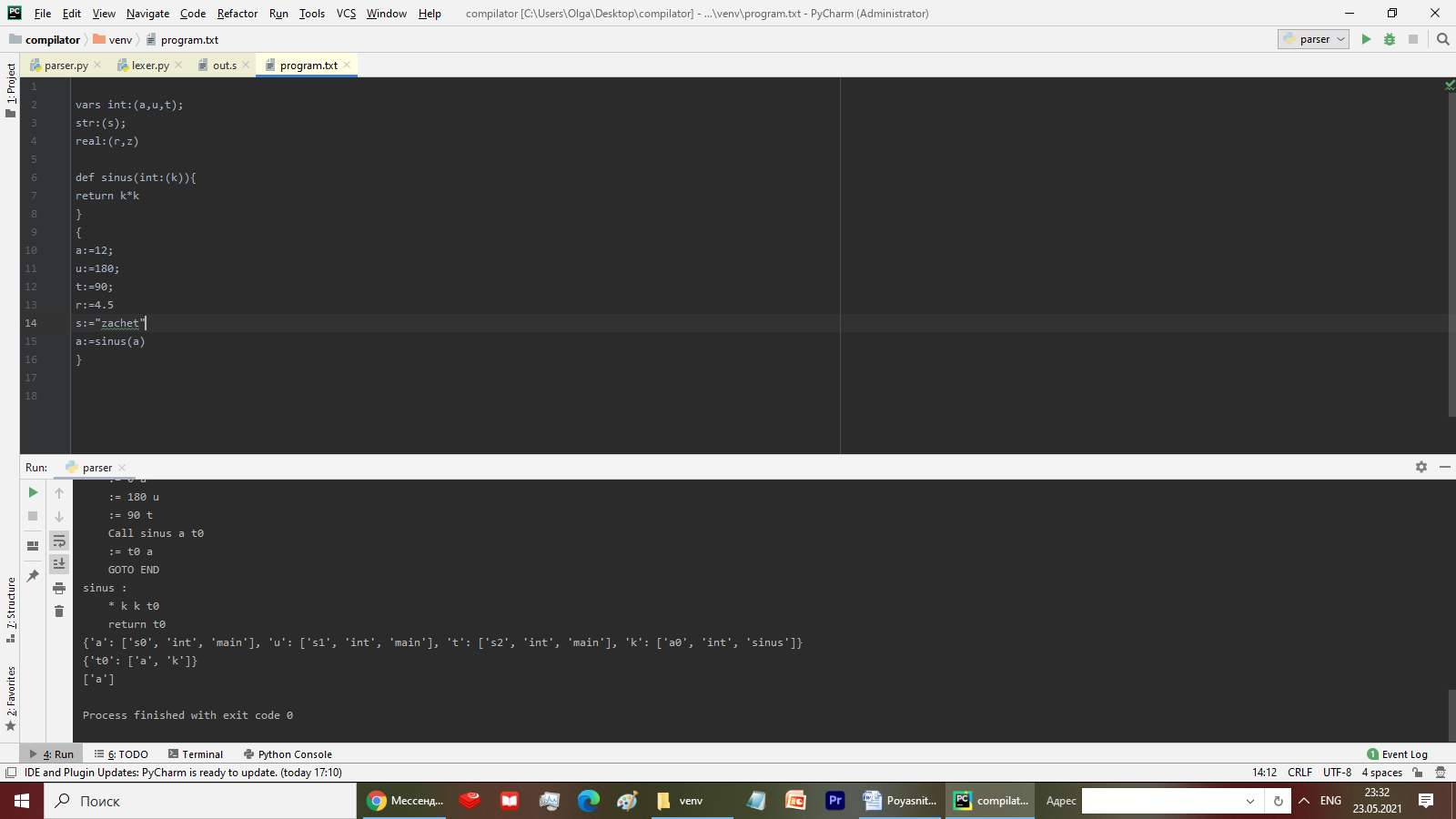
**Присваивание**

Блок присвоения представляется следующим правилом КС-грамматики:

prisv -> ID PRISV exp  
prisv -> ID PRISV STRING

[переменная]:=[выражение] или [переменная] или [строка];

Пример в программе:



**Условный оператор**

Условие определяется следующими правилами КС-грамматики:

if -> IF bool\_exp OF stmt\_list\_if CF

bool\_exp -> bool\_exp OR bool\_exp\_term  
bool\_exp -> bool\_exp\_term  
bool\_exp -> NOT bool\_exp  
bool\_exp -> bool

bool\_exp\_term -> bool\_exp\_term AND bool  
bool\_exp\_term -> bool

bool -> OP exp RAVNO exp CP  
bool -> OP exp MORE exp CP  
bool -> OP exp LESS exp CP

например

if (x>y) {}

if not(x>y){}

if (x>y) and (z<y){}

if (x=y) or (z>y){}

**Циклы**

Блок цикла определяется следующим правилом КС-грамматики:

while -> WHILE bool\_exp OF stmt\_list CF

Блок цикла действует по тому же правилу, что и while ,но будет повторяться пока наше условие не перестанет выполняться.

Кроме того, для данного блока имеются 2 специфичных оператора: *break* и *continue*. Оператор *break* позволяет досрочно выйти из тела цикла и продолжить выполнение команд, идущих за ним, а оператор *continue* позволяет прервать текущую итерацию и начать выполнение тела цикла заново.

**Объявление функций**

Пользовательские функции задаются после блока объявления переменных и перед основным телом программы. Объявление функции начинается с ключевого слова *def,* затем в скобках задаются аргументы с указанием типов, и далее в фигурных скобках тело функции.

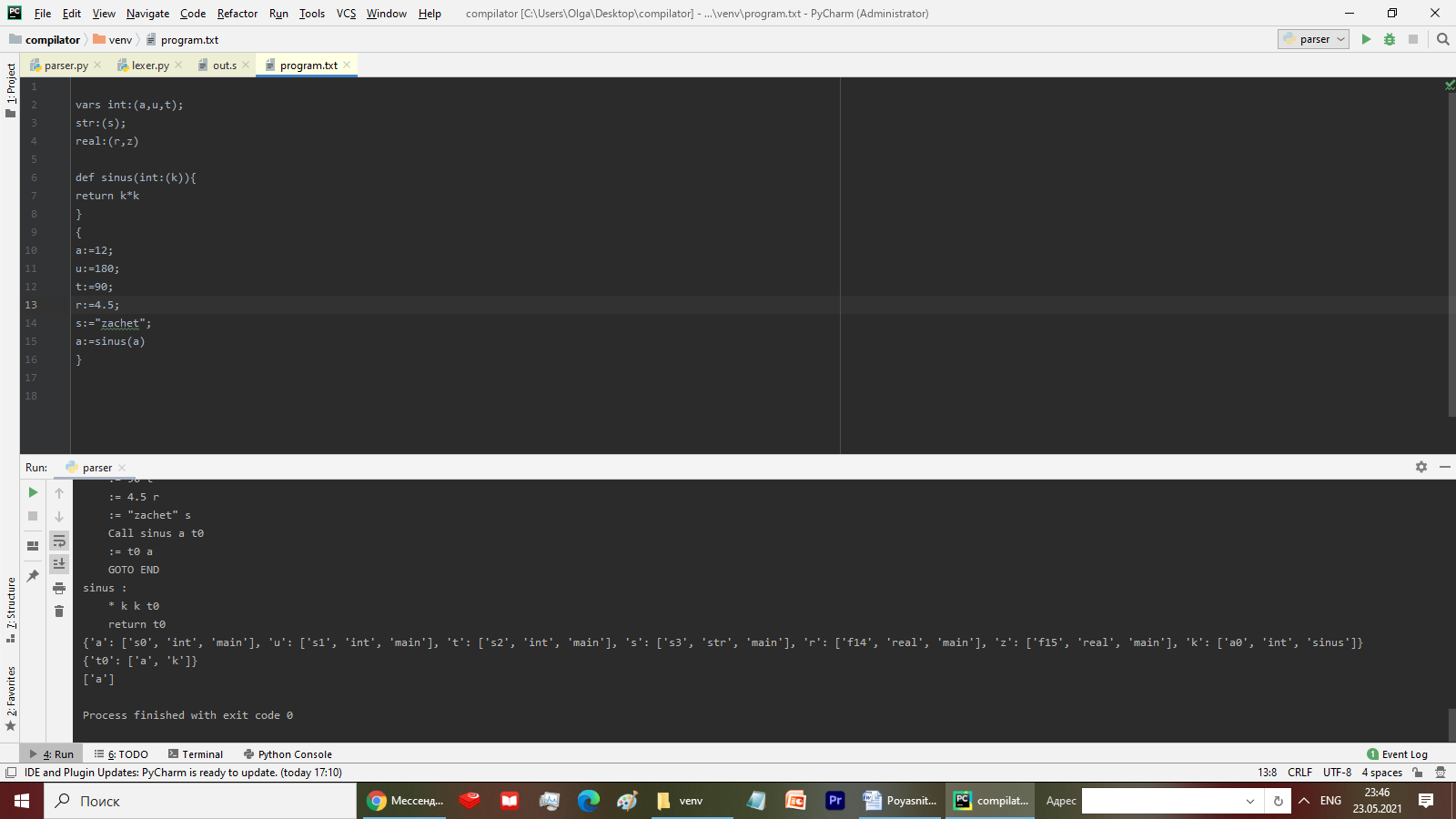
Объявление функций определяется следующим правилом КС-грамматики:

def -> DEF ID OP dec\_list CP OF stmt\_list\_def CF

def -> DEF ID OP dec\_list CP OF VARS dec\_list stmt\_list\_def CF

**Вызов функций**

В теле программы функции можно вызвать, указав название функции, и в скобках указав аргументы.



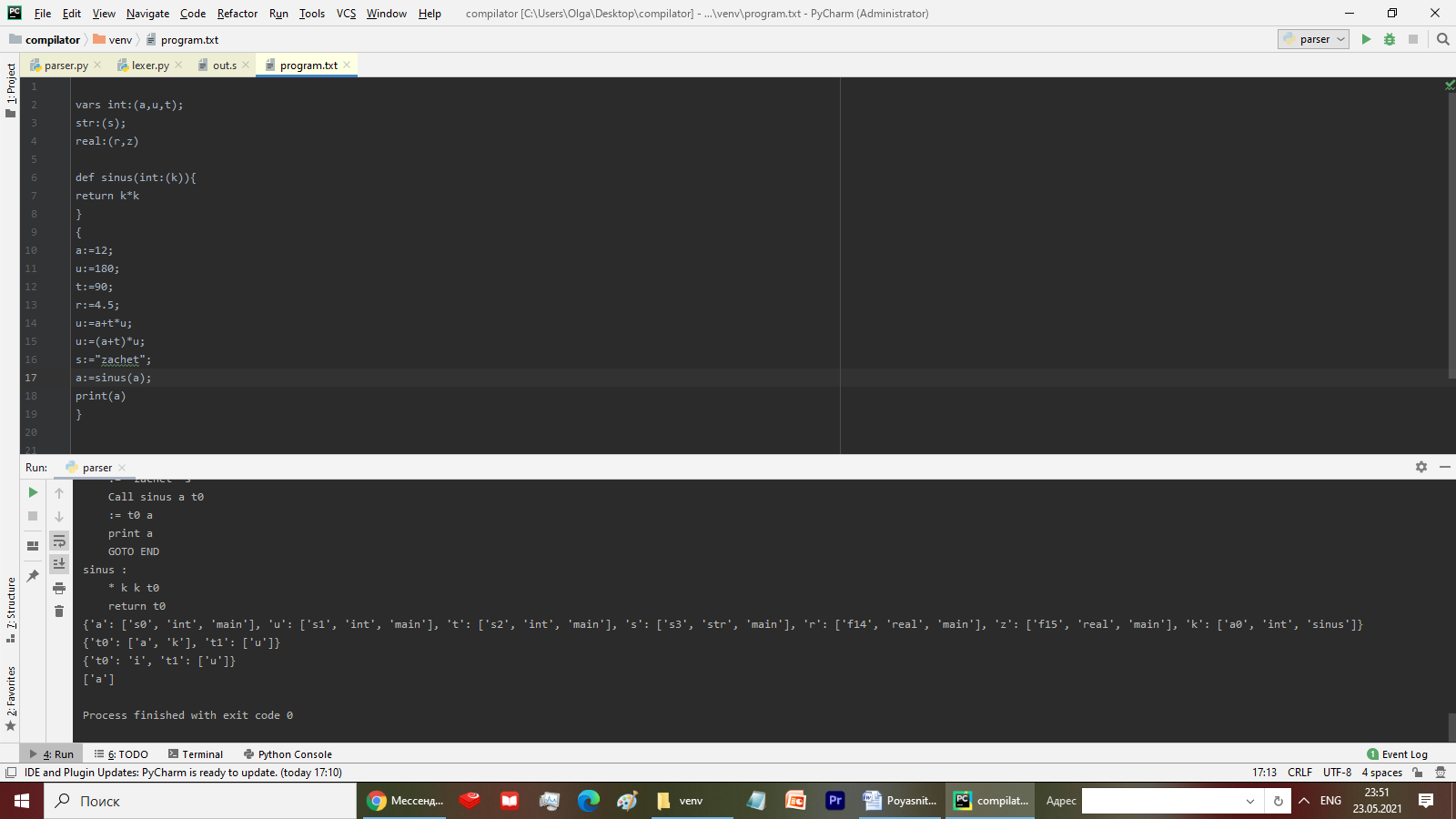
**Стандартный вывод**

Встроенная функция *print()* позволяет выводить на стандартный вывод выражение в скобках , выводилось с новой строкинужно написать print(“\n”)*.*

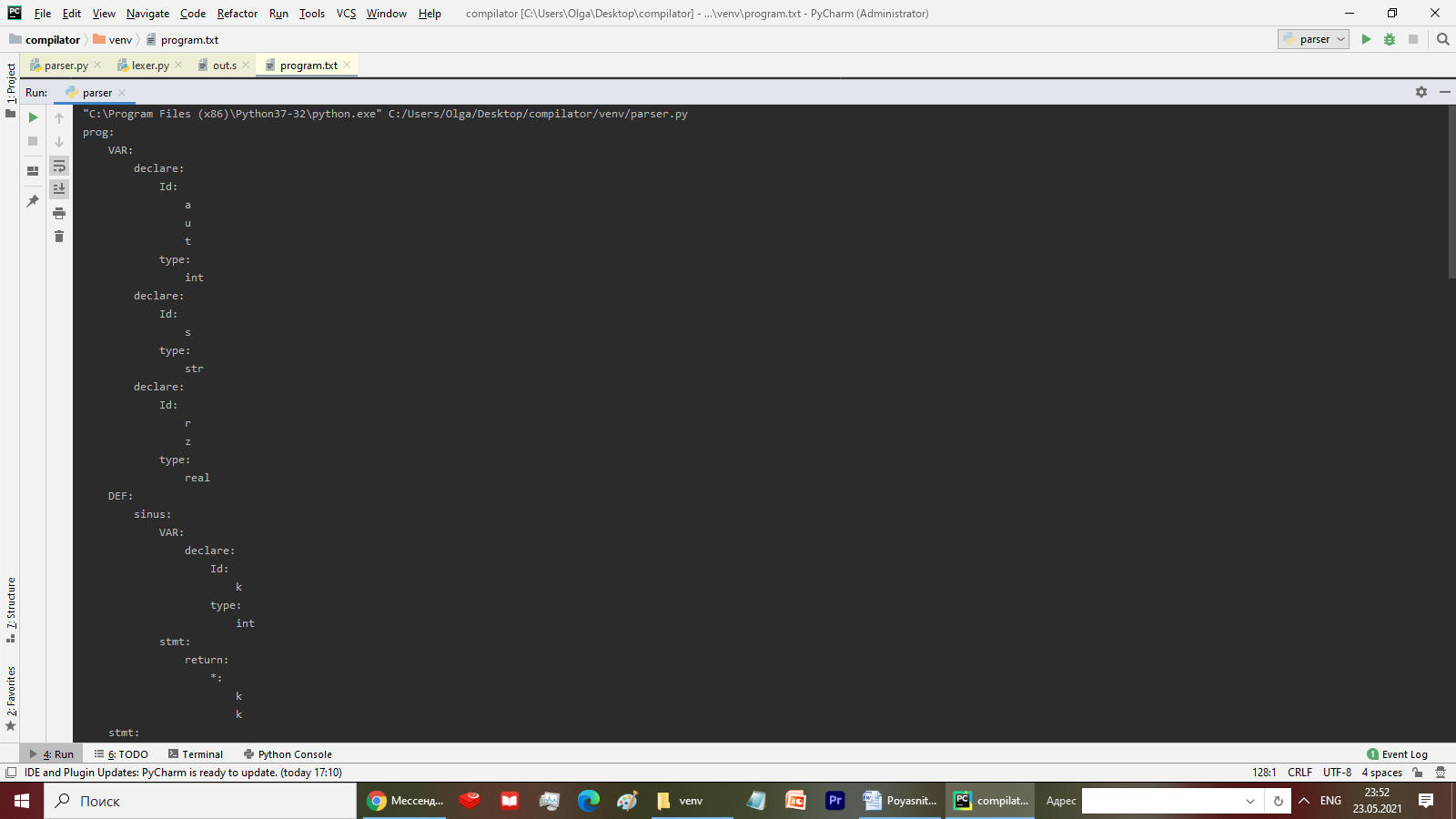
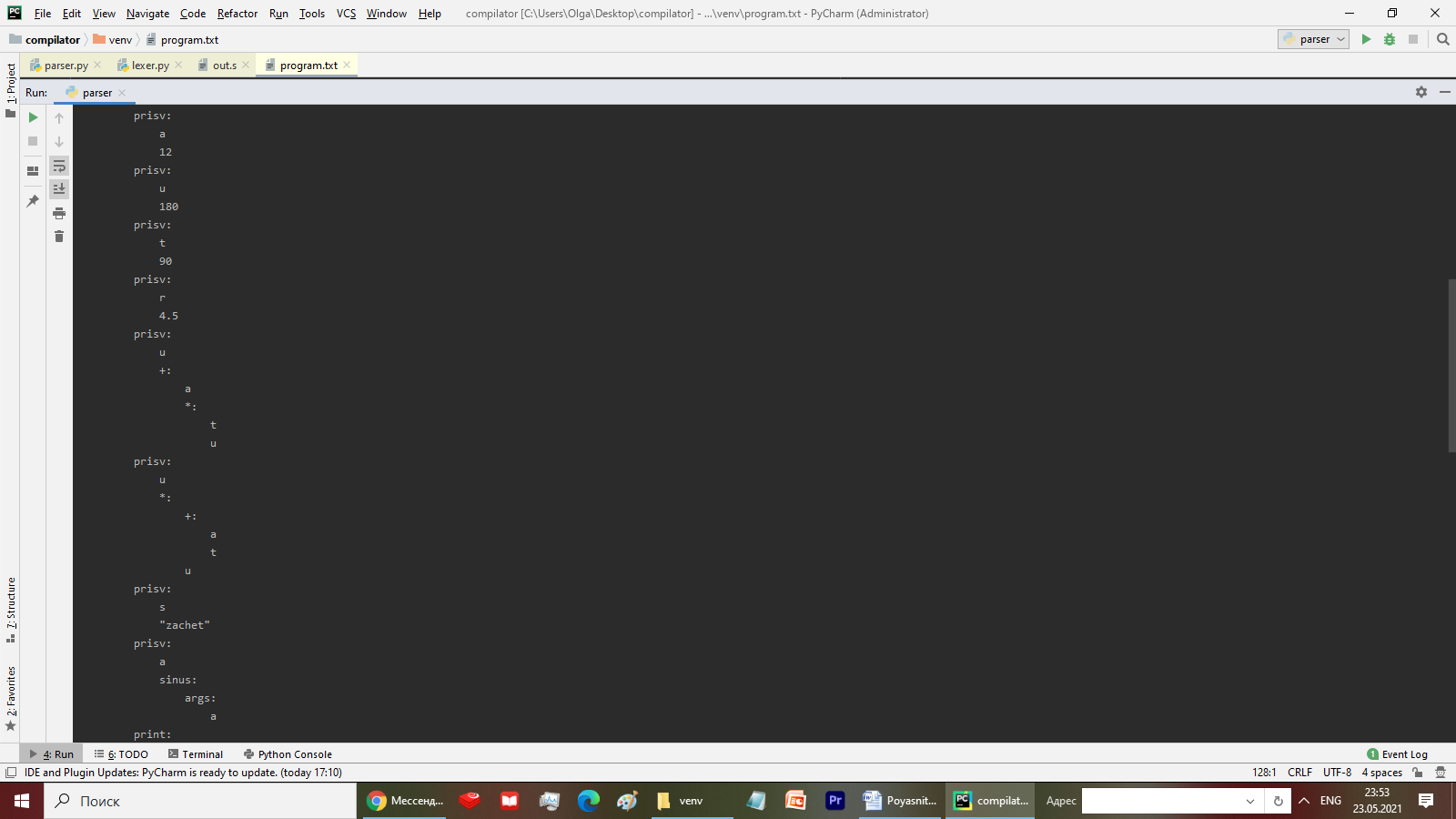
На выходе синтаксический анализатор возвращает дерево синтаксического разбора.

Пример:

Программа:



Дерево синтаксического разбора:

1)2) 

## Таблица символов

Следующим этапом является генерация таблицы символов. Таблица символов содержит в себе все переменные, их типы, область видимости, а также используемые регистры в MIPS-коде.

Для генерации таблицы символов используется рекурсивный обход синтаксического дерева обхода.

Пример:

Программа:

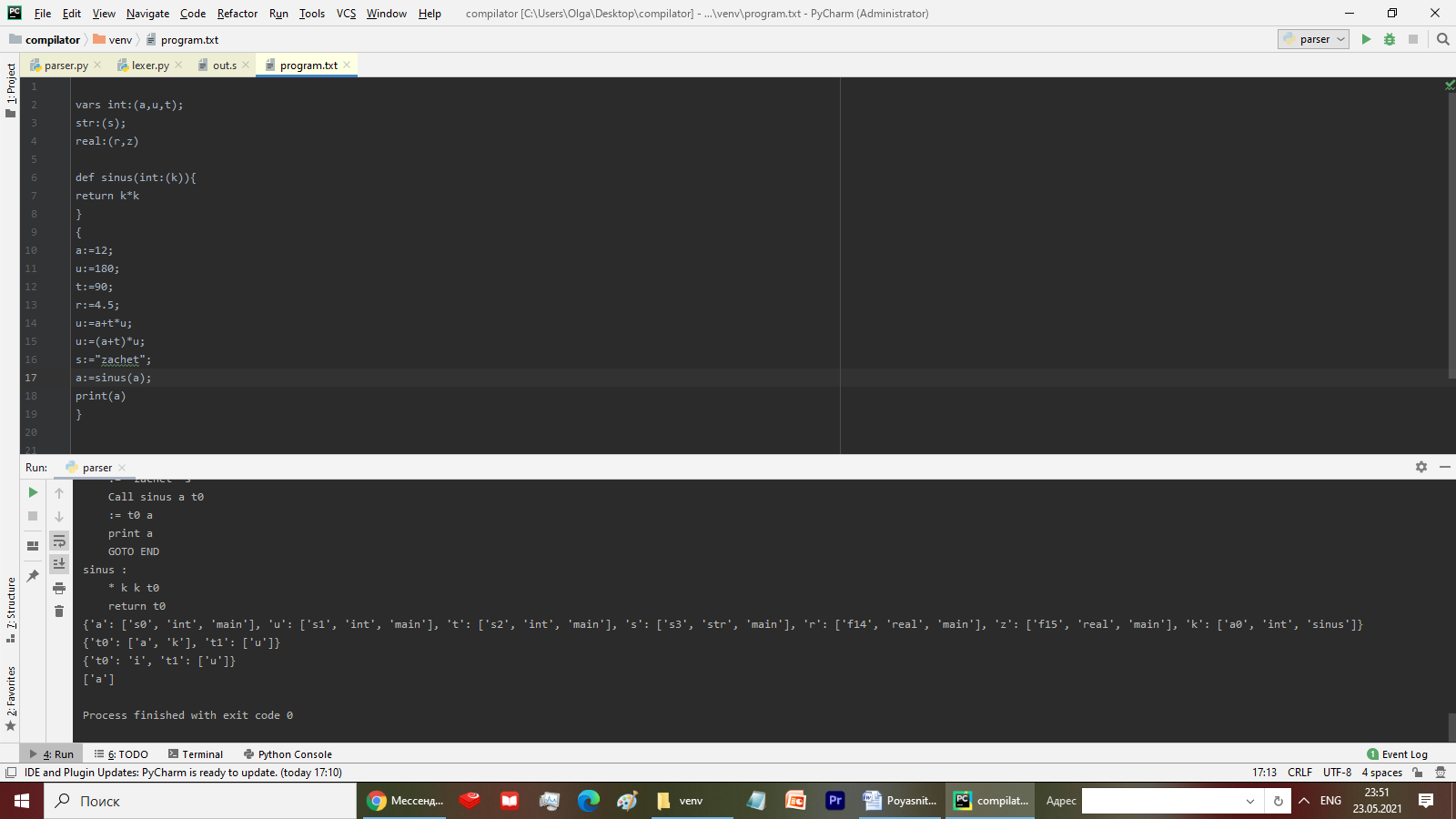


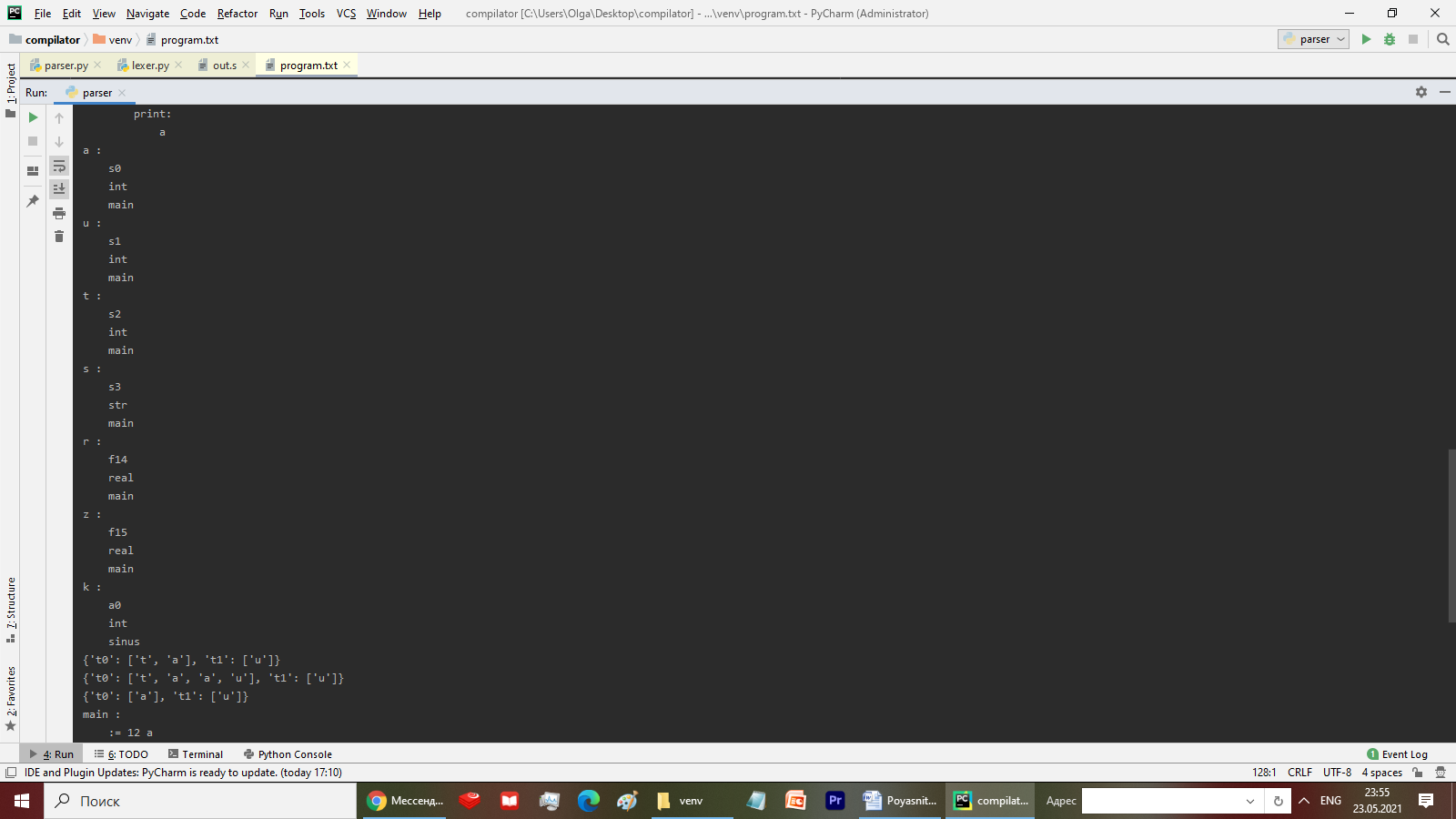
Таблица символов:

Первое значение – представление переменной для ассемблера

(sn- для int и str в main , an- для функций (n сбрасываеться для каждой функции), fn –для чисел с плавающей точкой

Второе значение – тип переменной

Третье значение – область видимости



## Промежуточный код

Следующим этапом является генерация промежуточного кода.

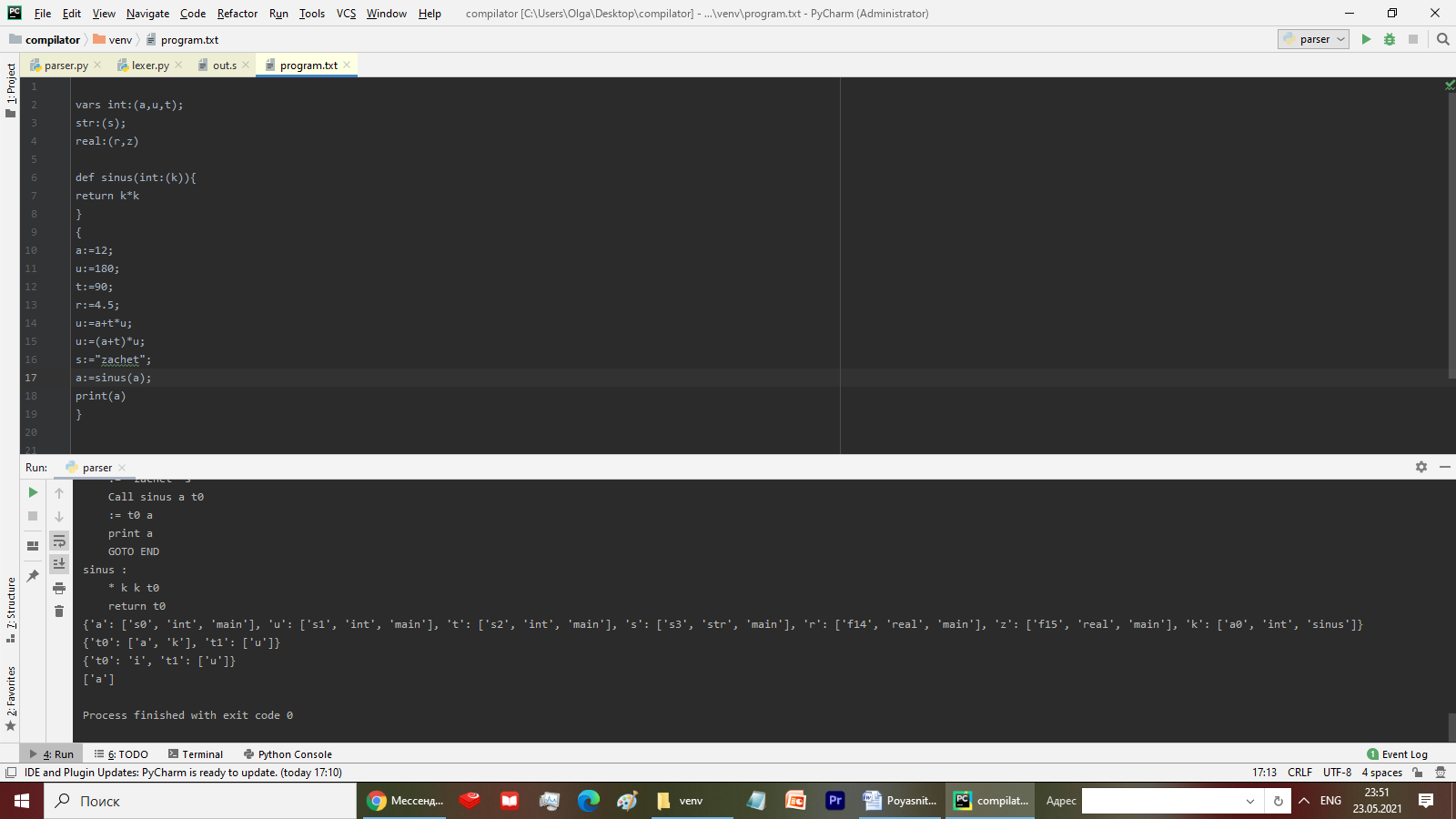
Был выбран трехадресный код , как промежуточный.

Трех адресный код строится следующим образом:

[операция] [первый аргумент] [второй аргумент] [результат]

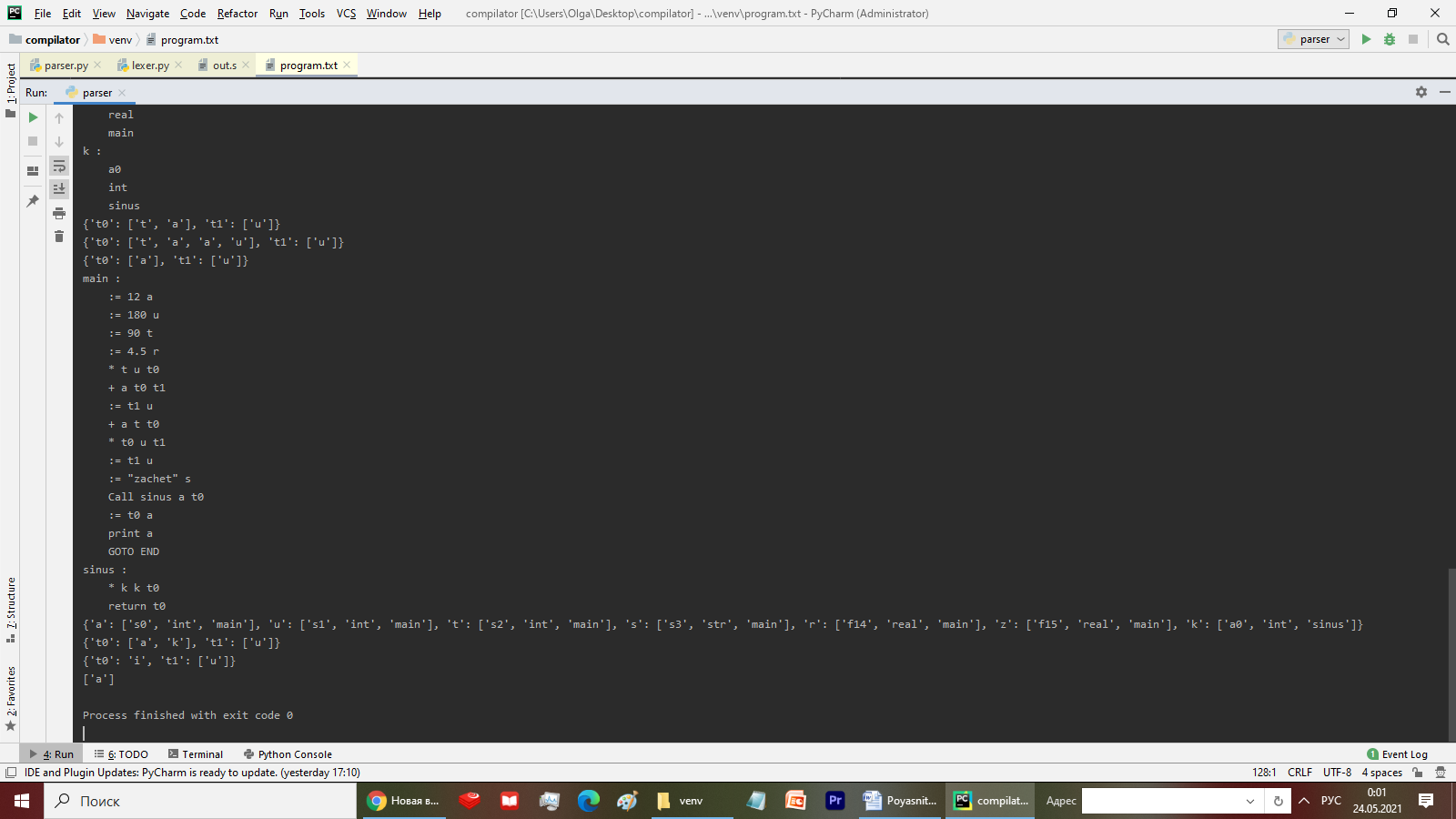
Промежуточный код генерируется с помощью рекурсивного обхода дерева синтаксического обхода.

Пример:

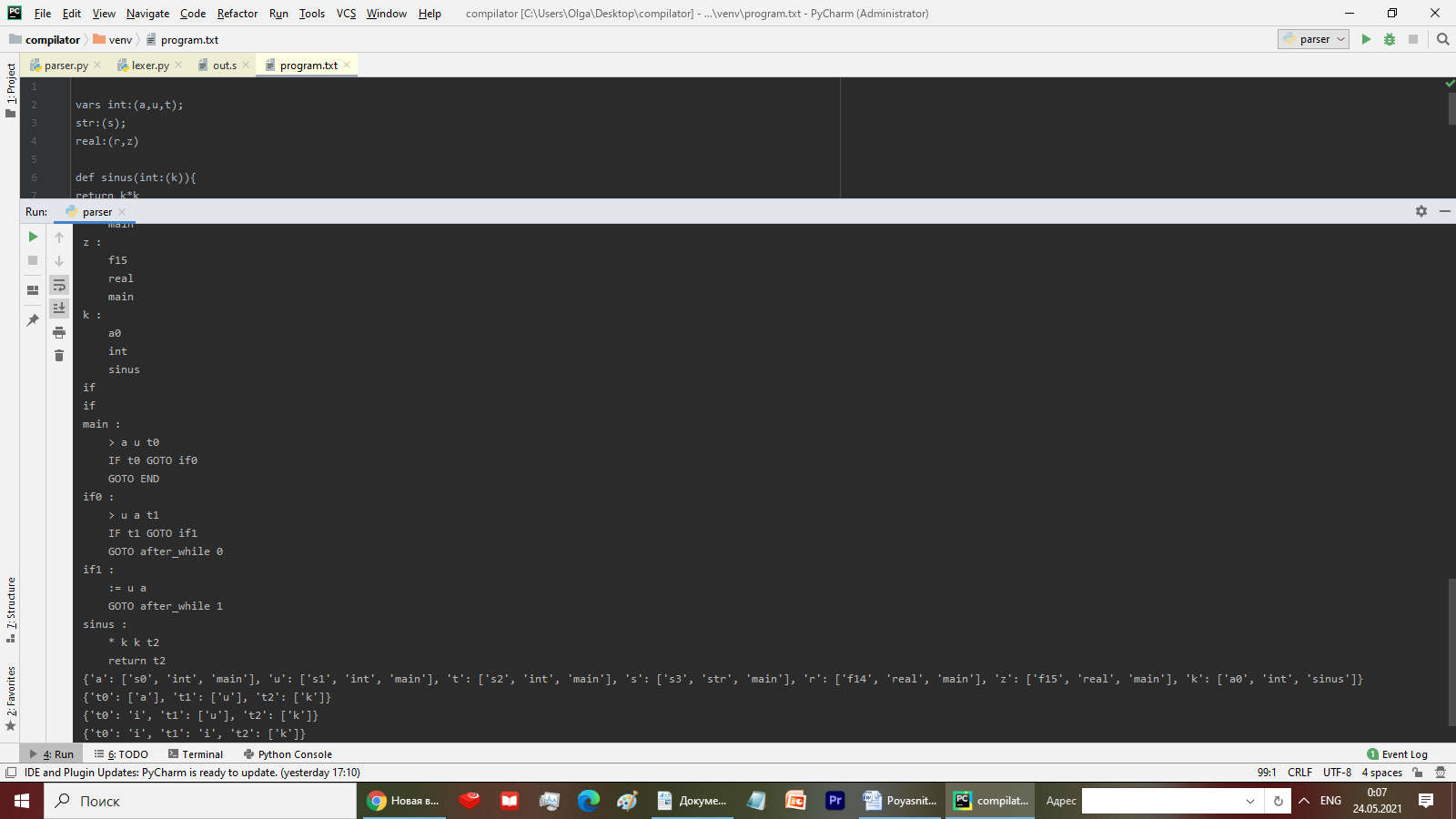


Трех адресный код:

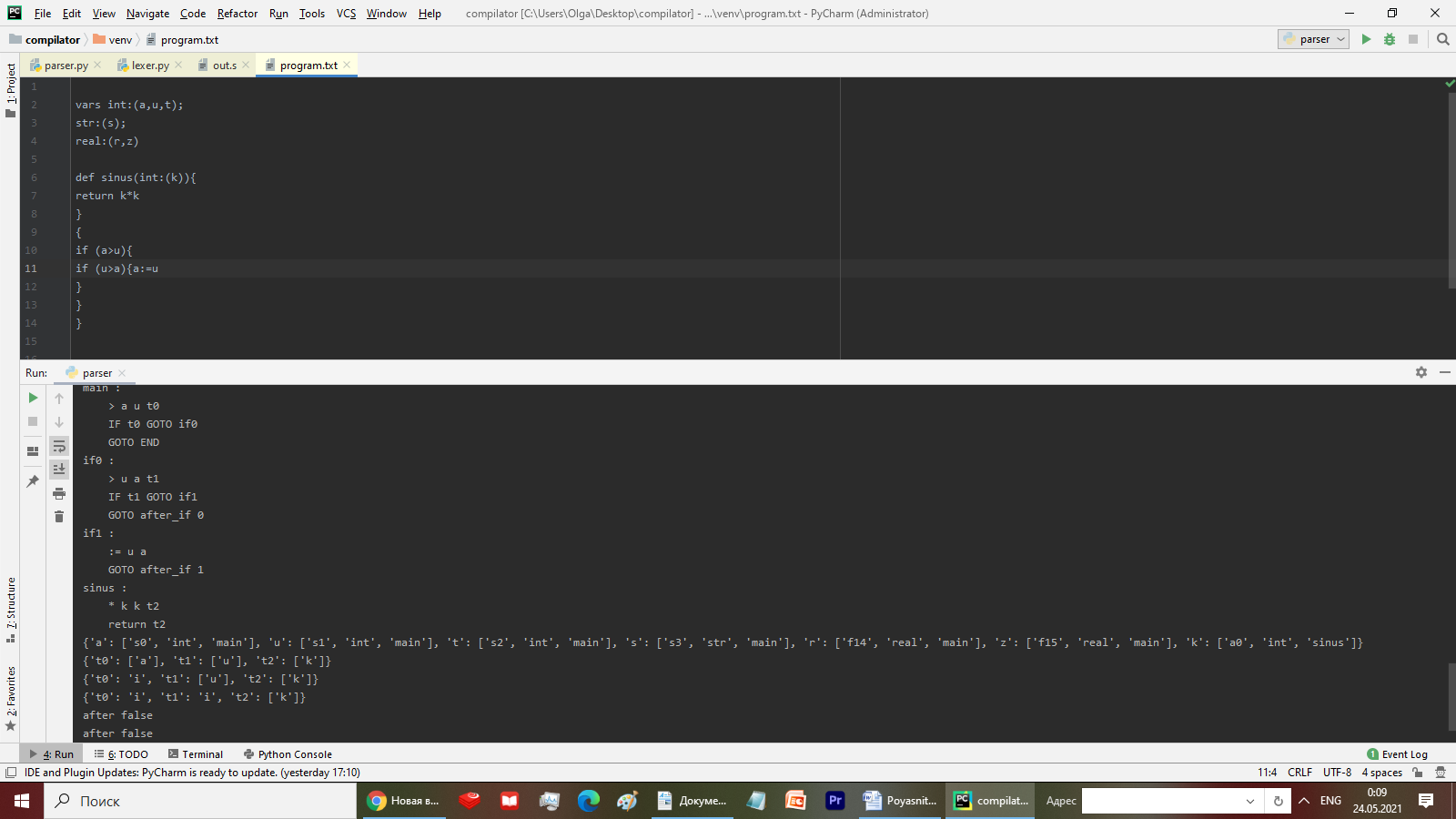
Сall – операция для вызова функции



Пример с while



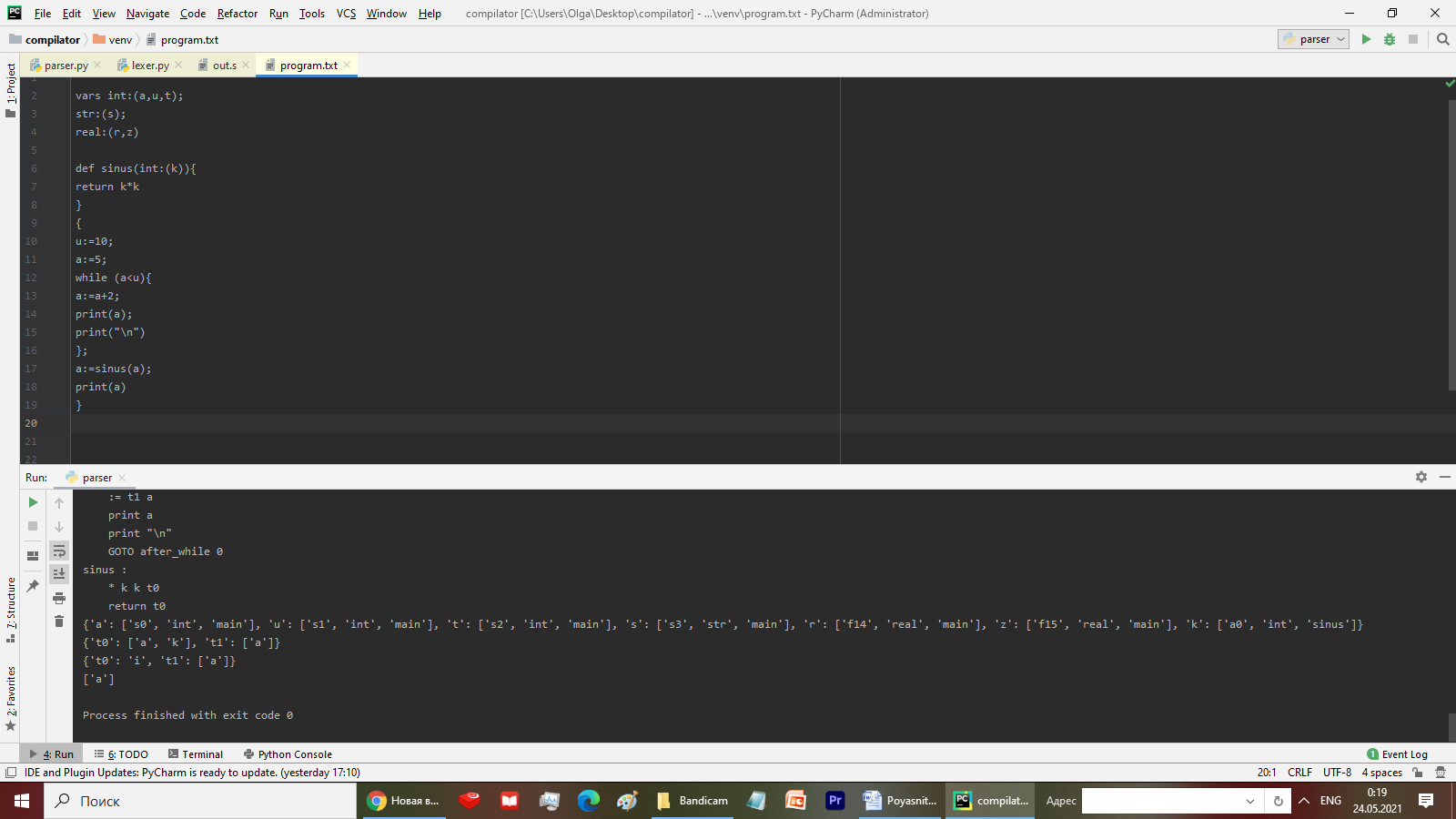
Пример c if



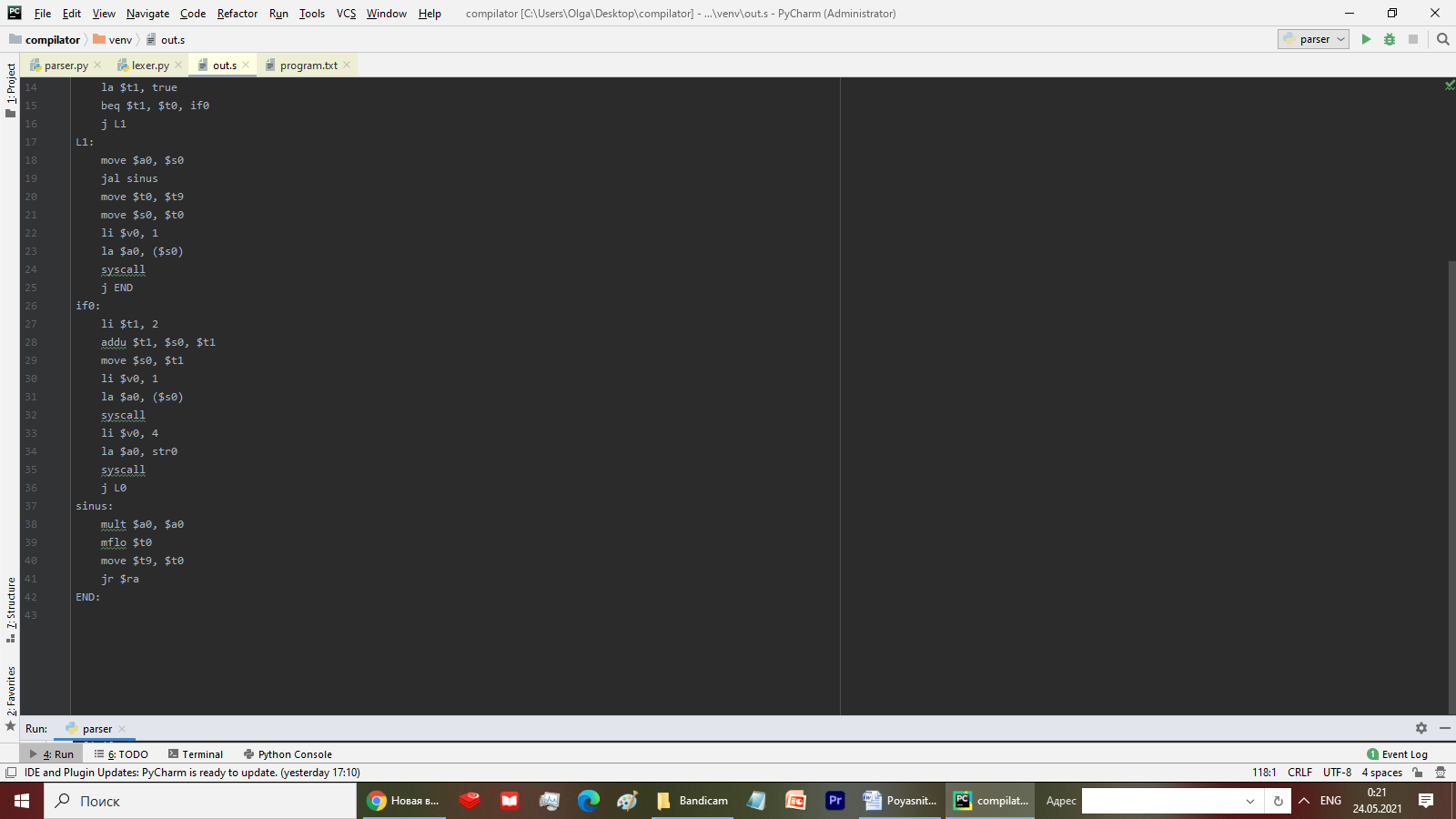
## Генерация кода ассемблера

Целевым языком является язык ассемблера MIPS. Промежуточный код построчно транслируется в код ассемблера MIPS.

Пример для программы:

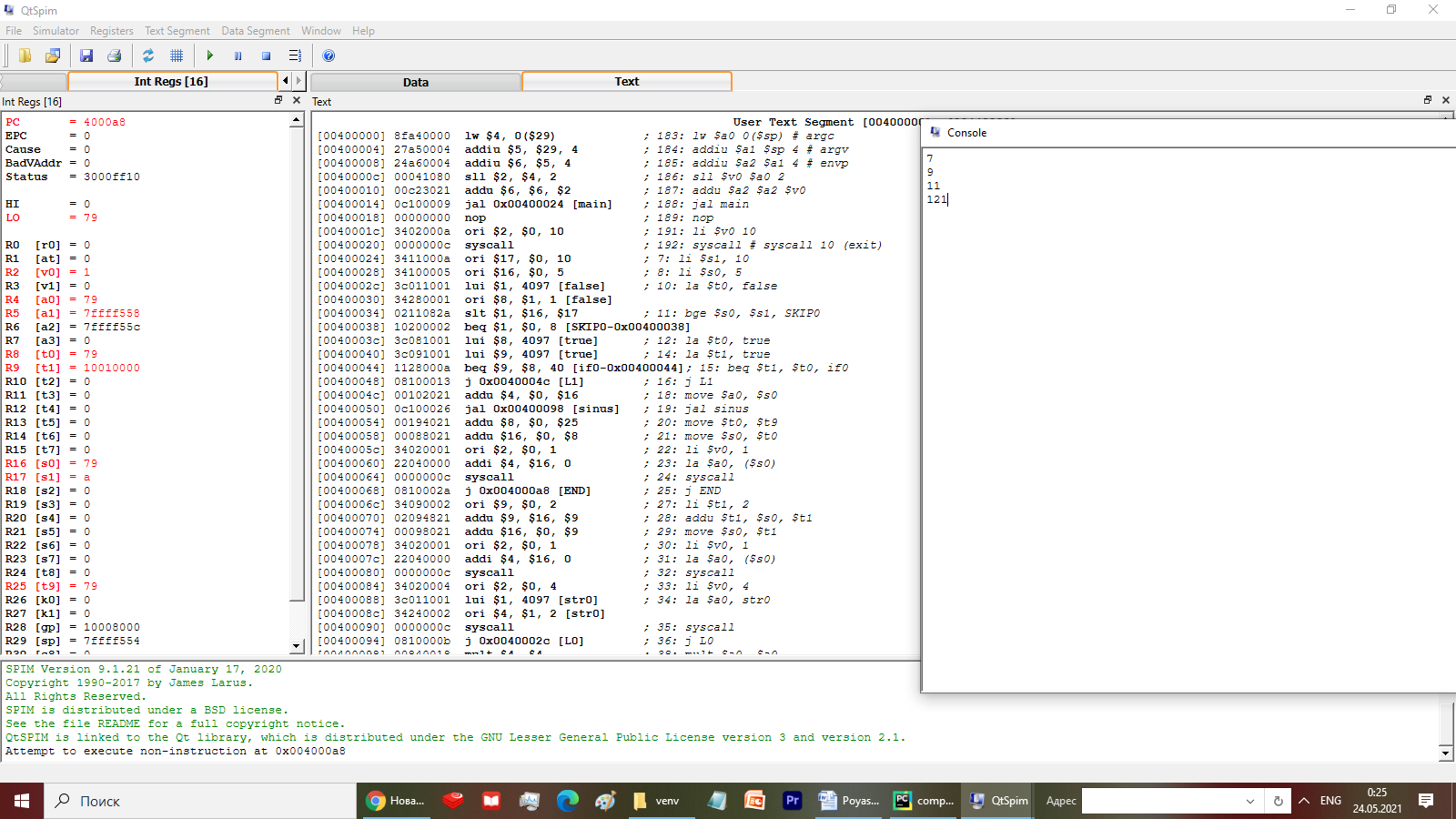


Код ассемблера MIPS



Для запуска сгенерированного кода используется эмулятор MIPS QtSpim.

Результат работы программы:

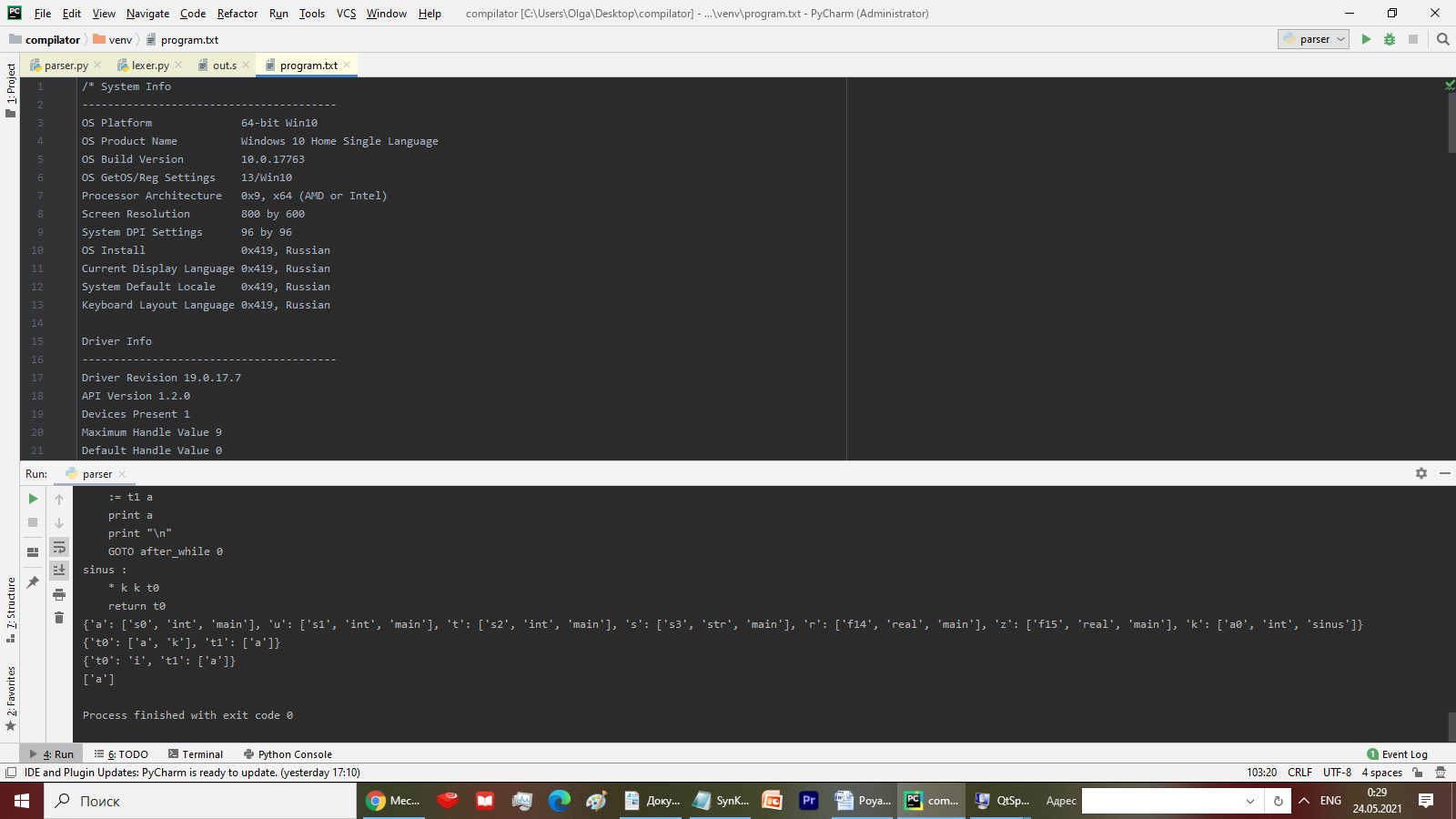


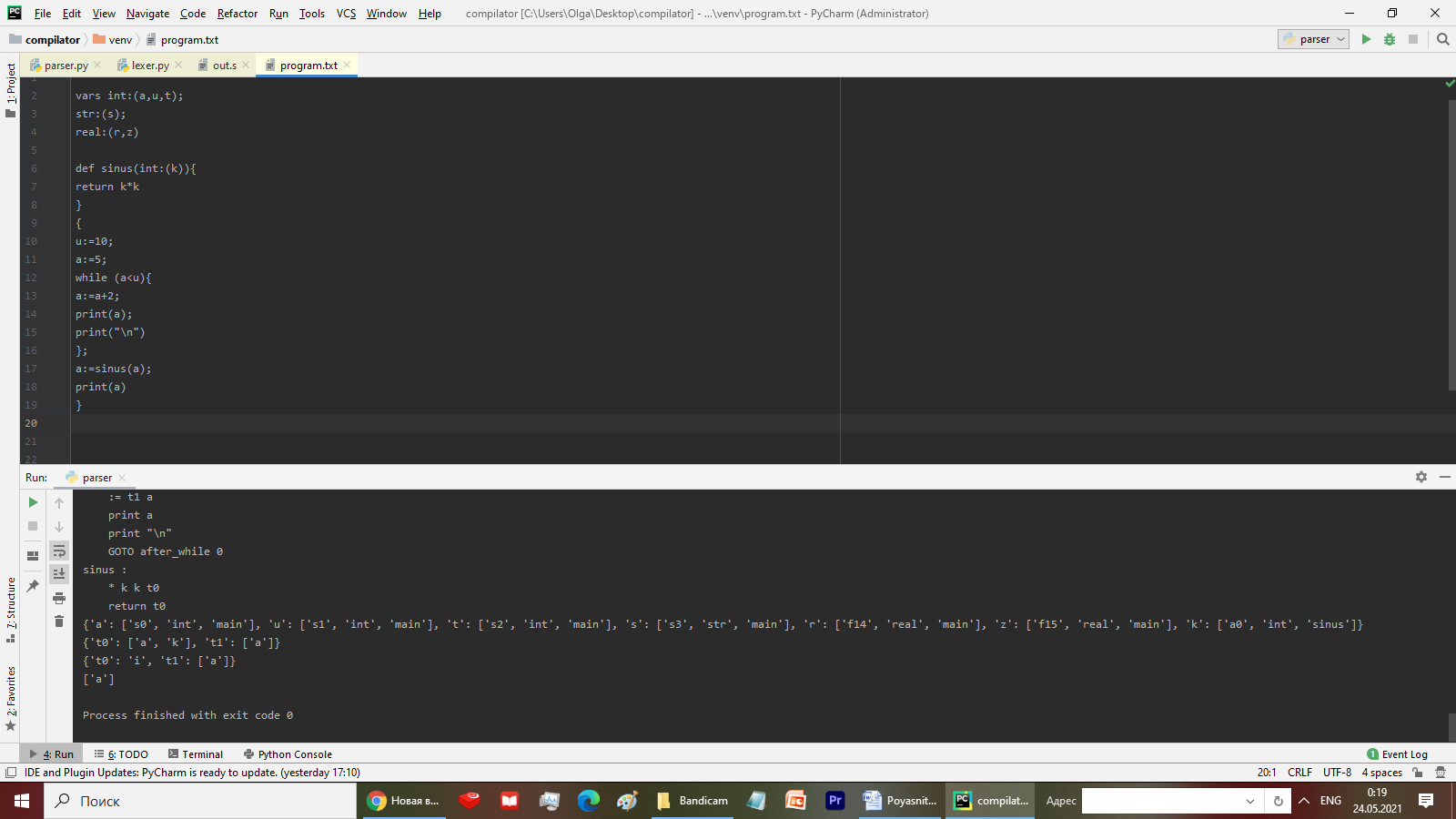
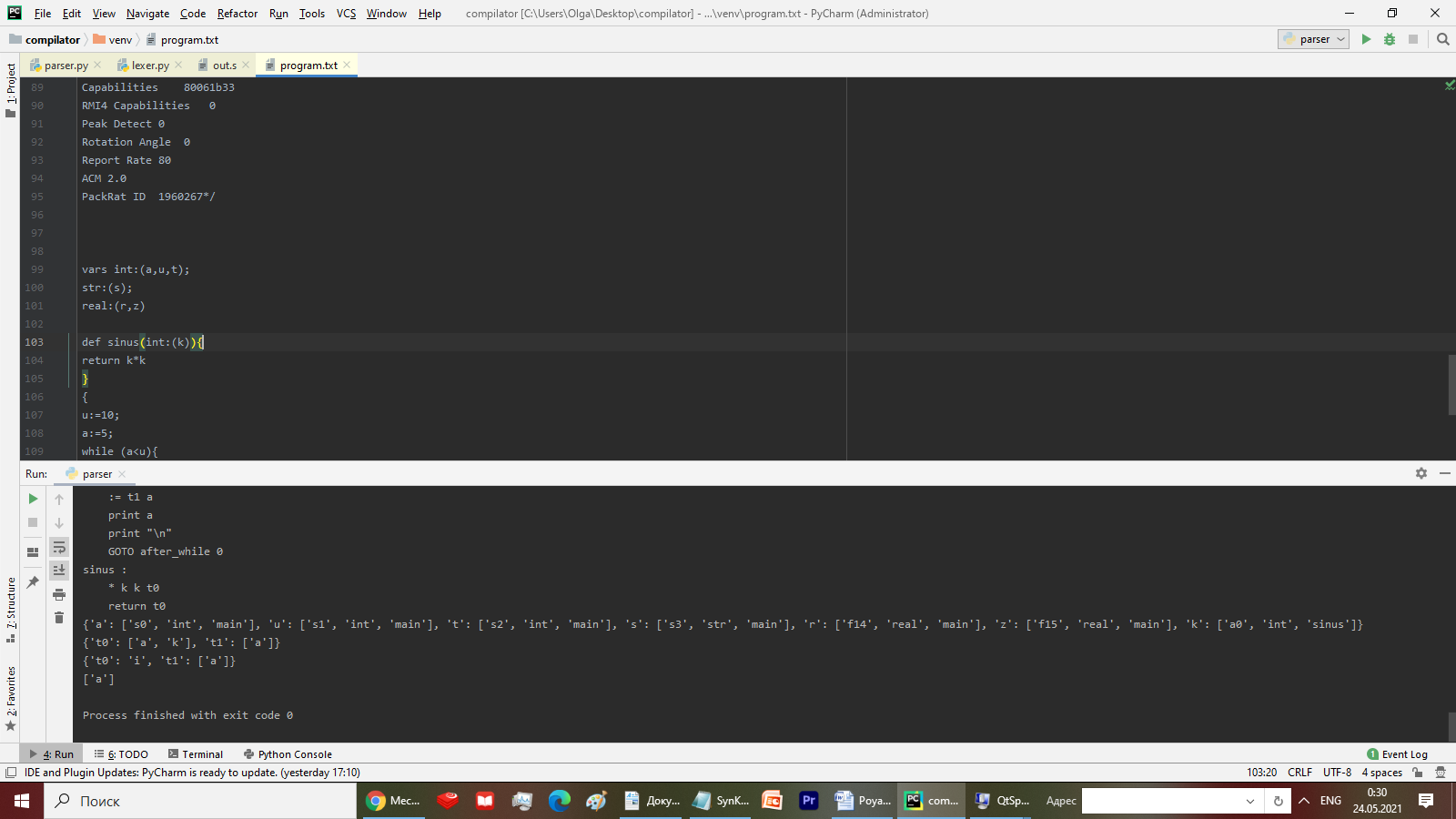
# **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

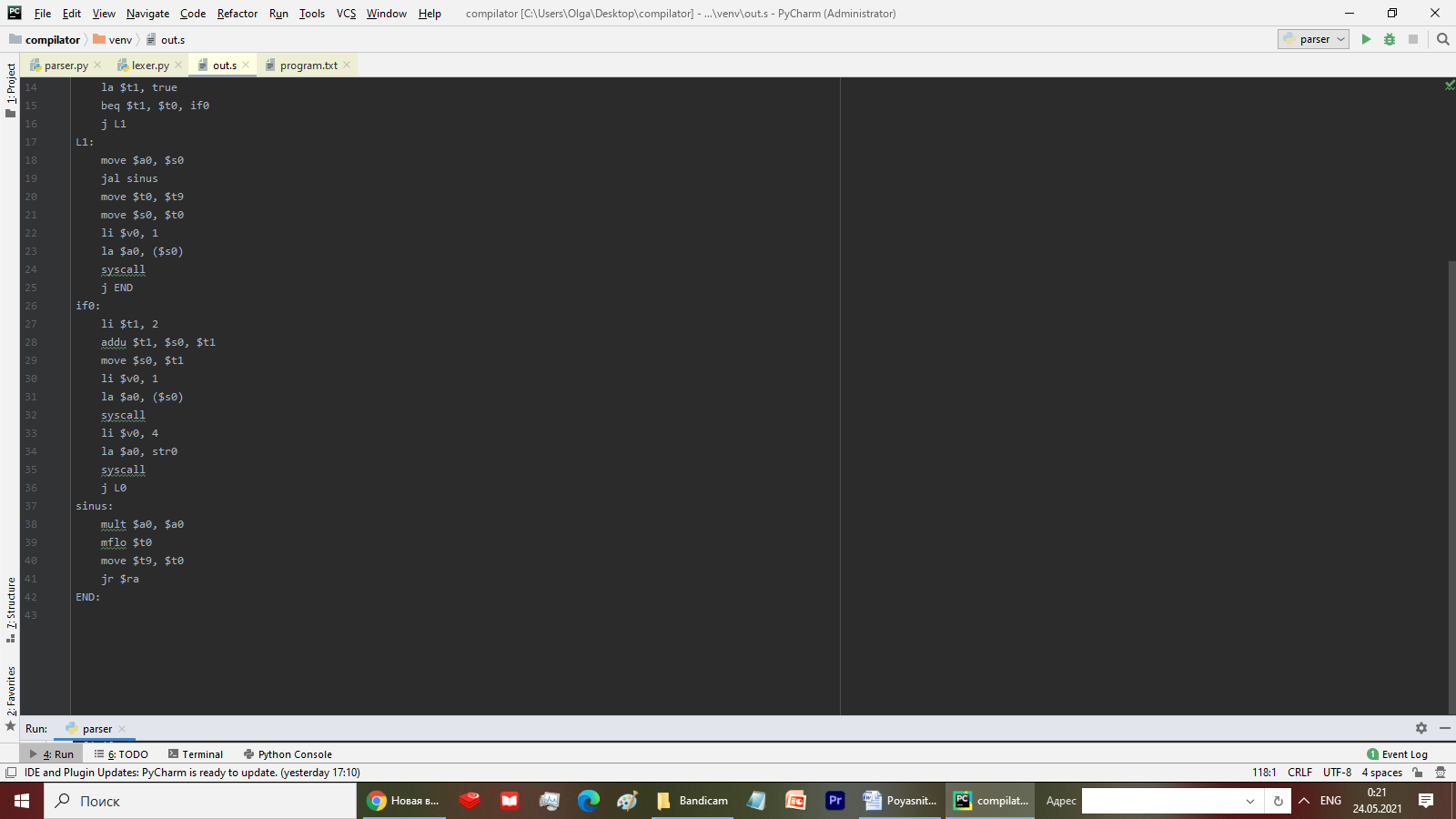
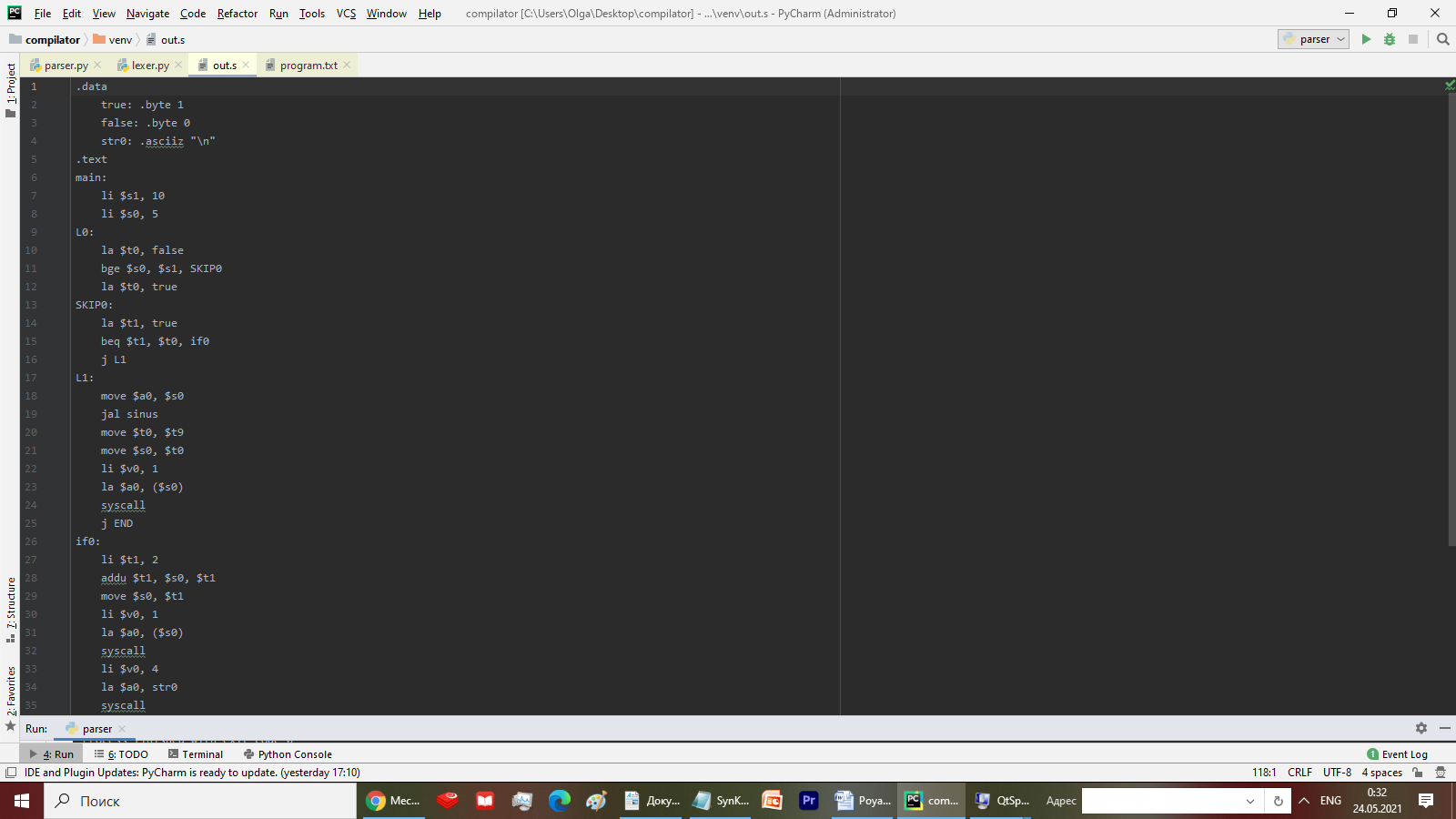
При проектировании компилятора к основному языку были установлены следующие минимальные требования: наличие операторных скобок, игнорирование индентации программы, поддержка комментариев любой и вызова функций. Наличие операторов присваивания, условных, цикла, арифметических, логических, базового вывода. Должны присутствовать два типа данных – целочисленный и с плавающей запятой.

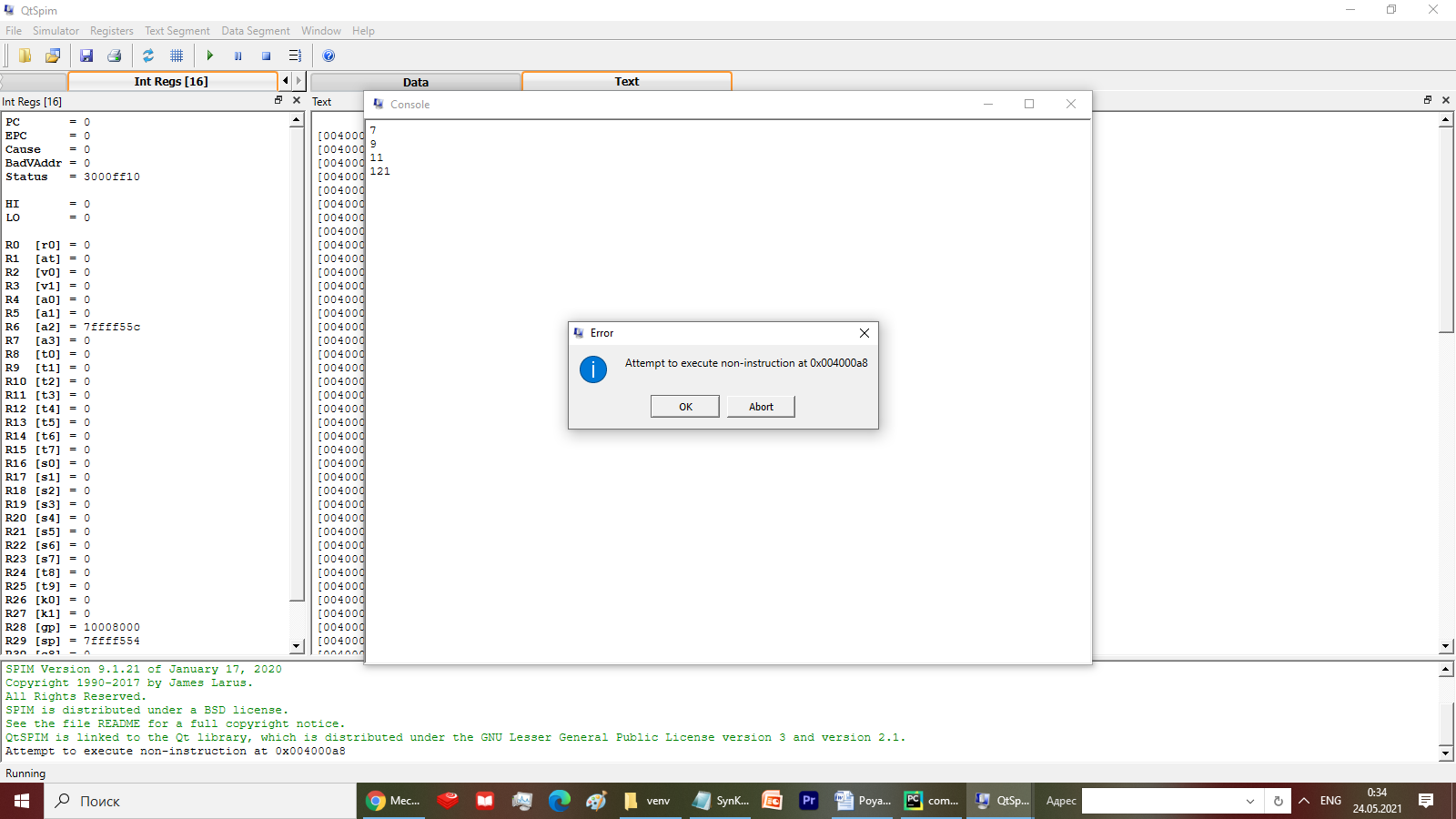
Далее приведено тестирование компилятора.

Проверка на игнорирование индентации, комментариев, проверка оператора присваивания, арифметики, оператора цикла, базового вывода:



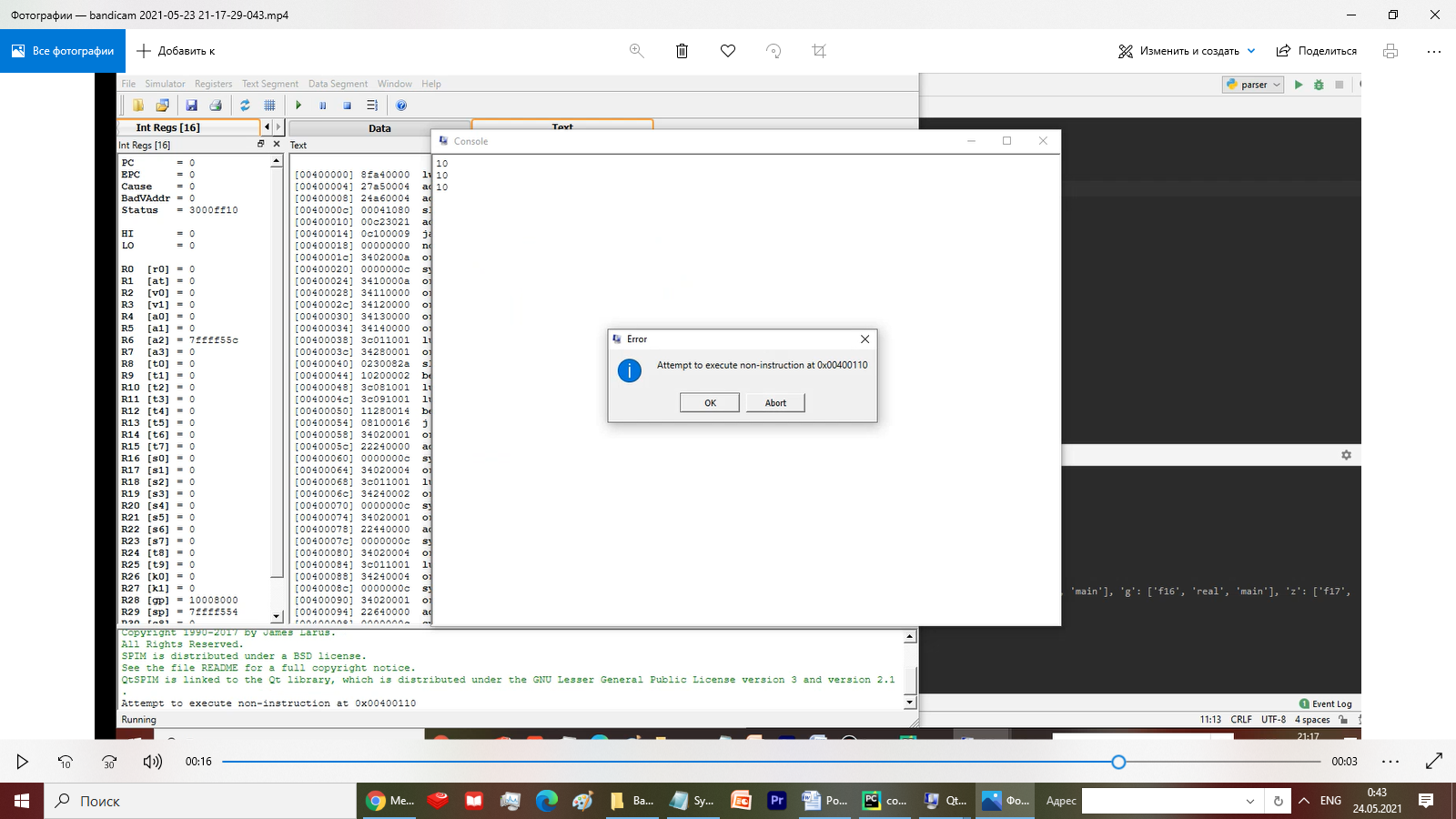




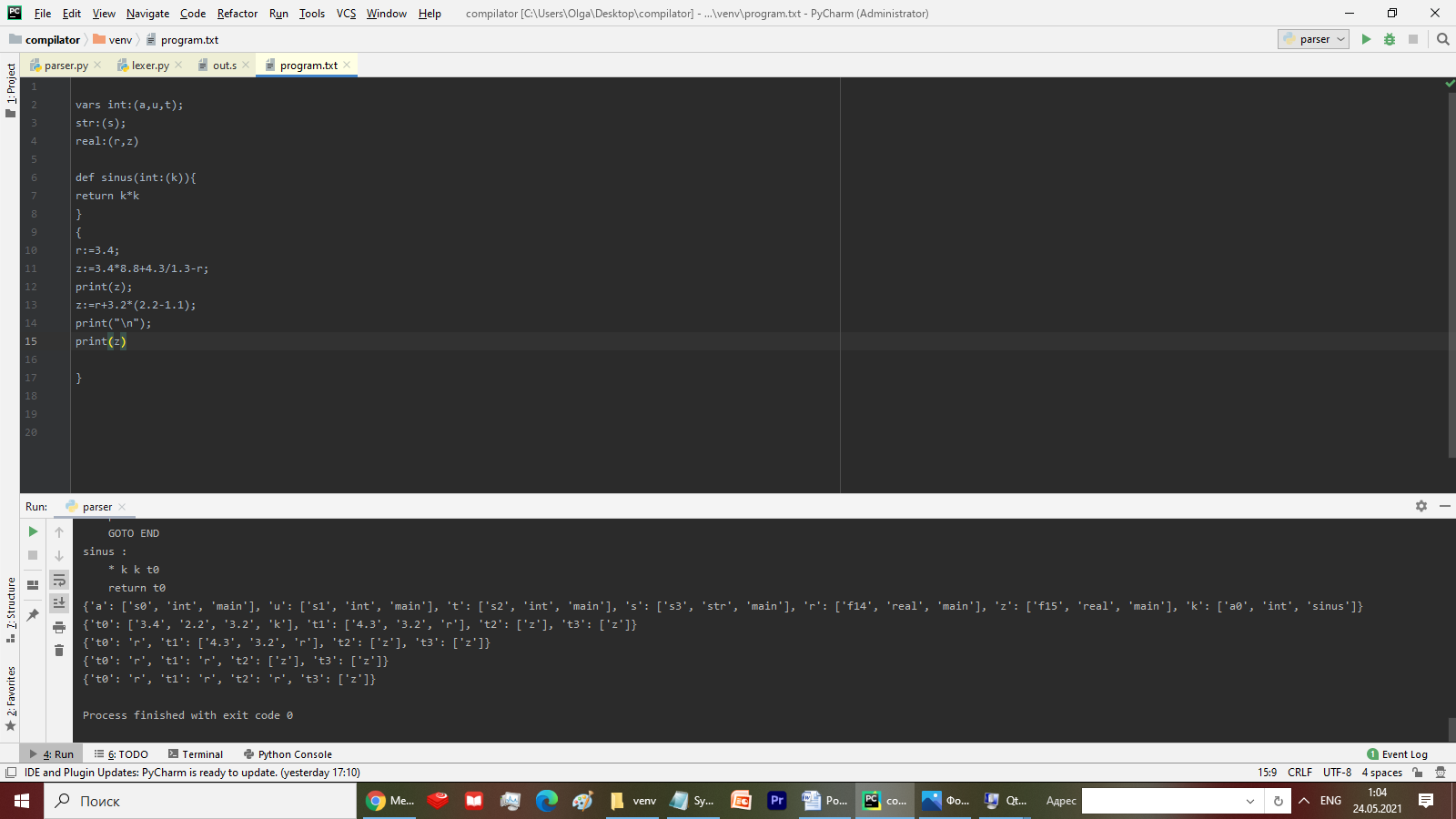


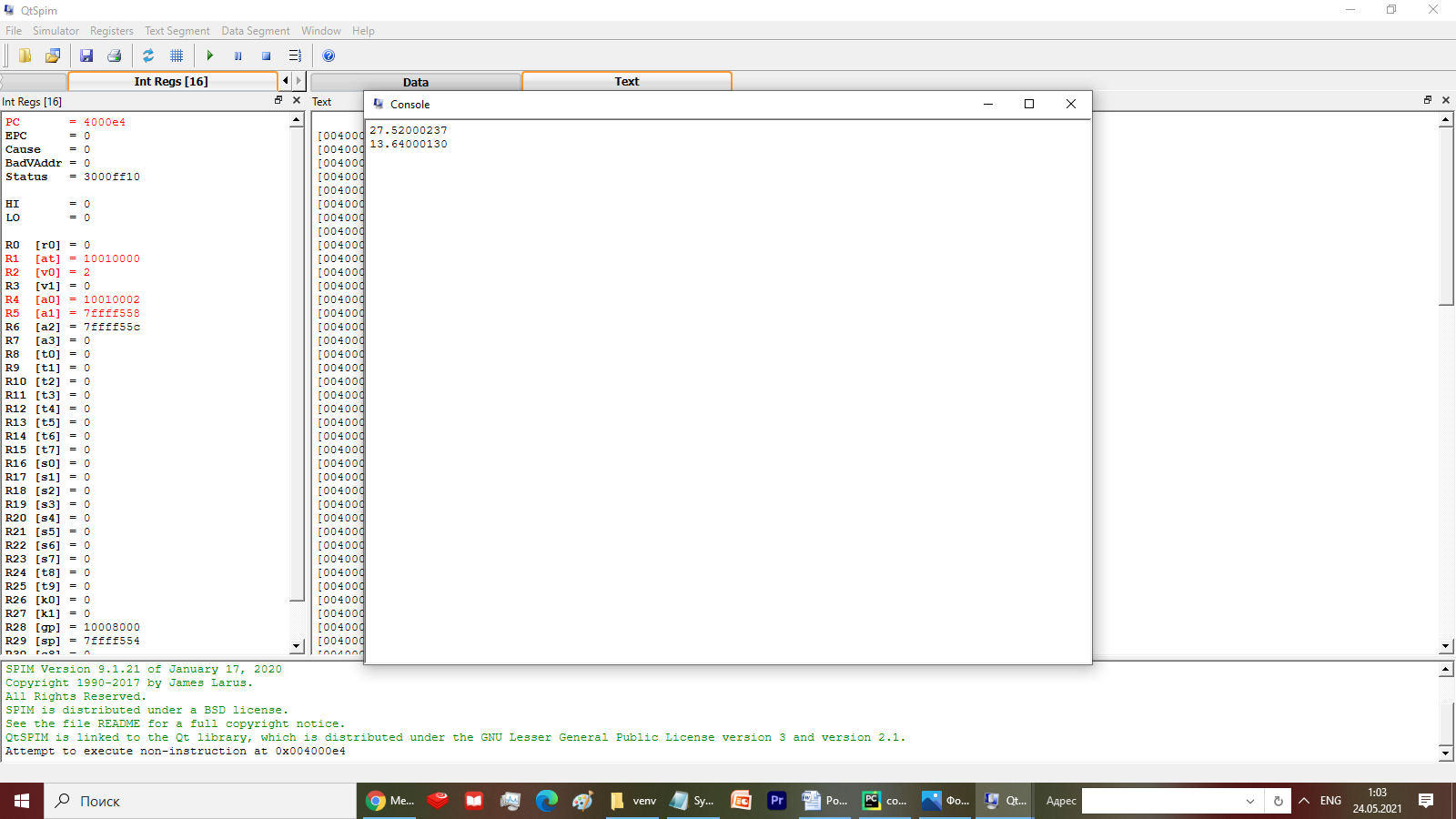
Проверка while



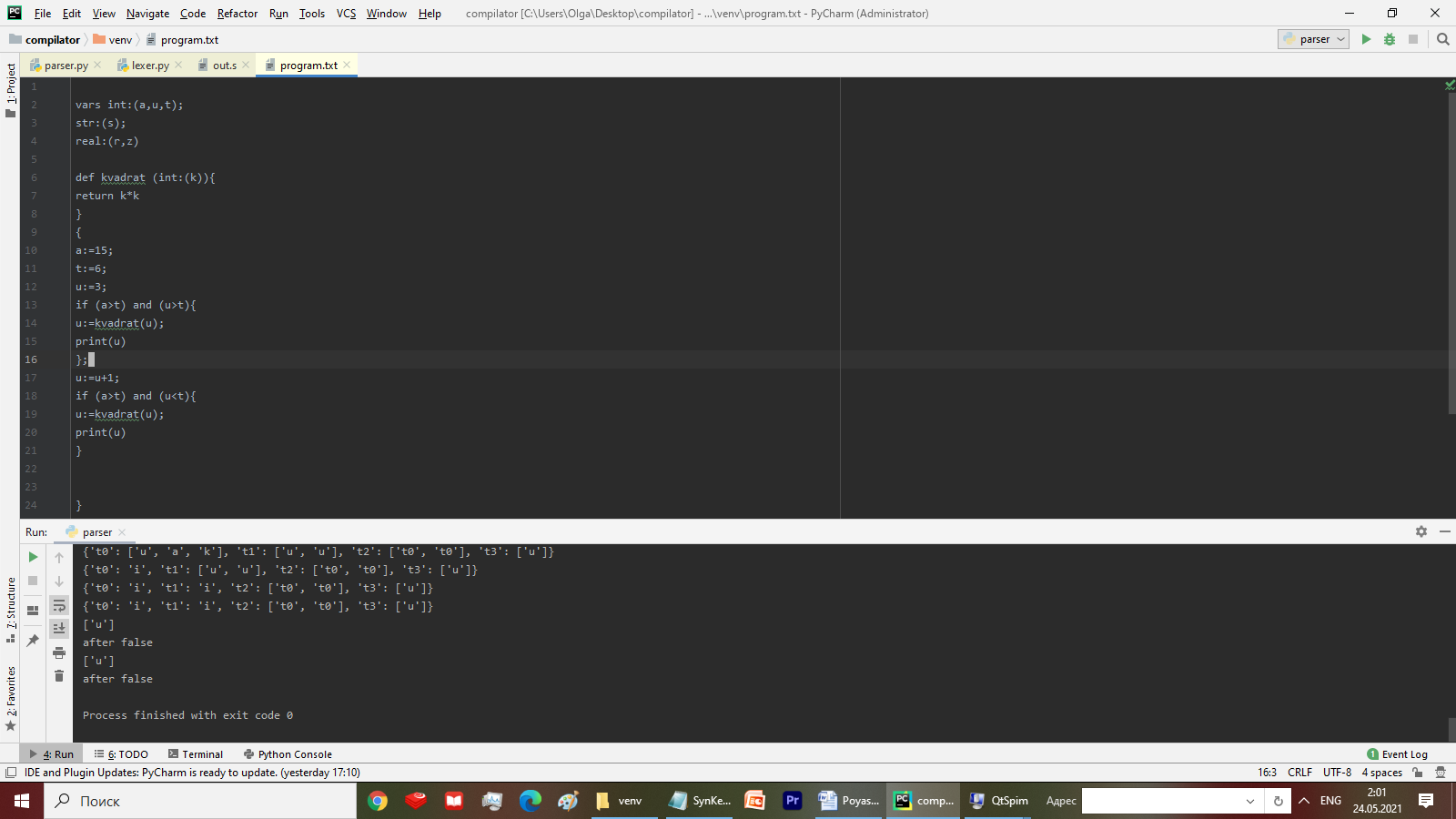


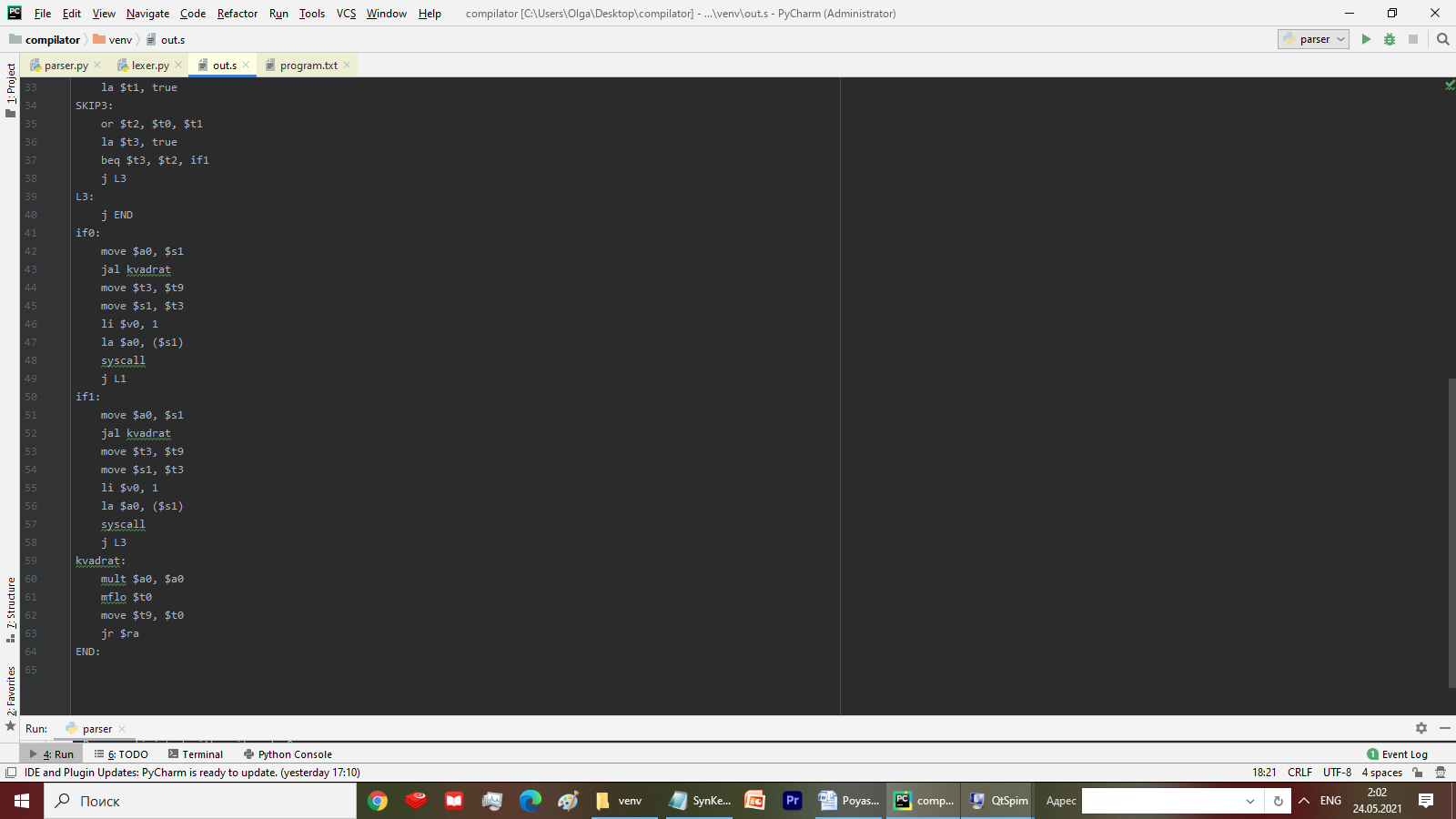
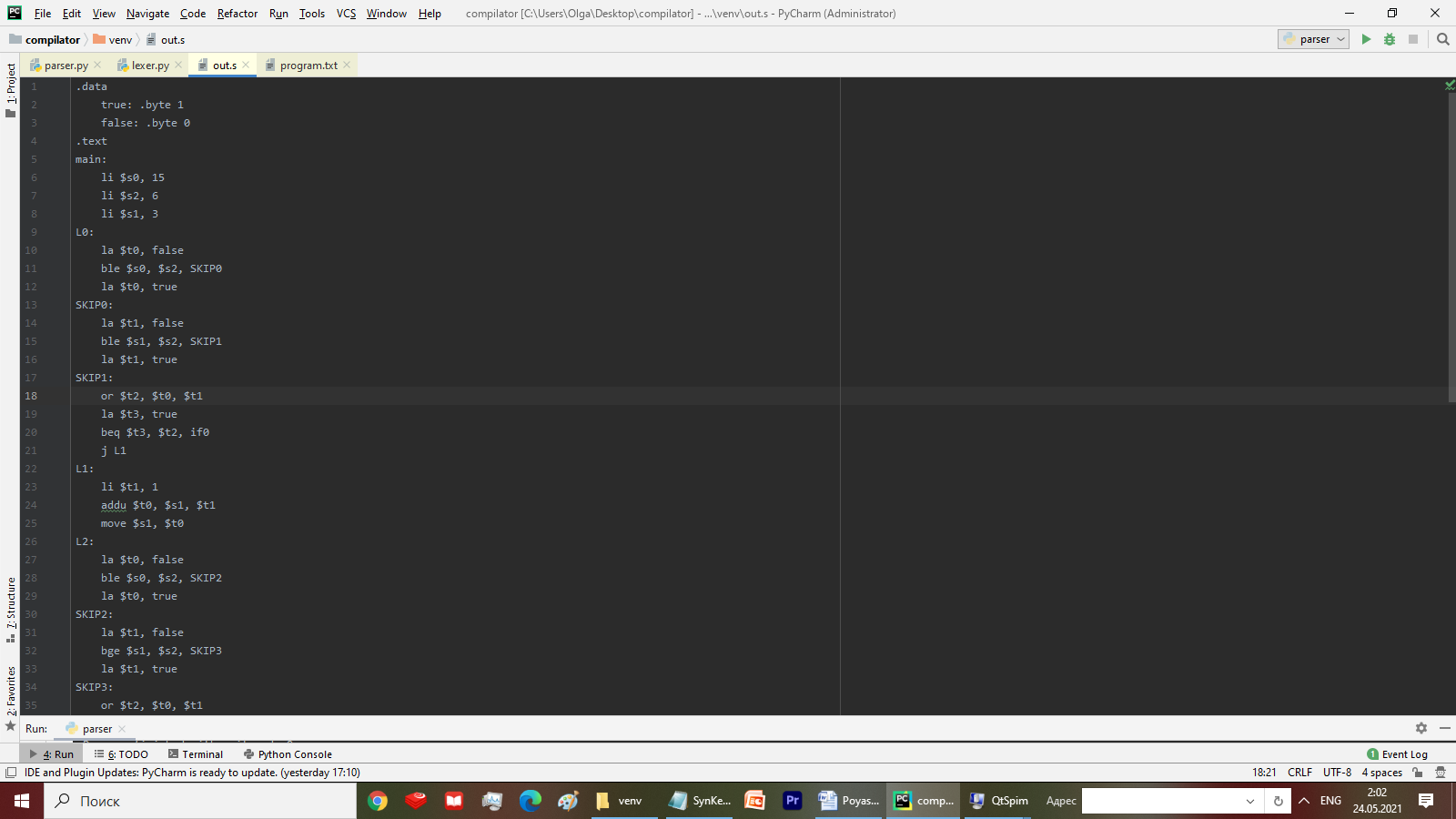
Арифметические операции



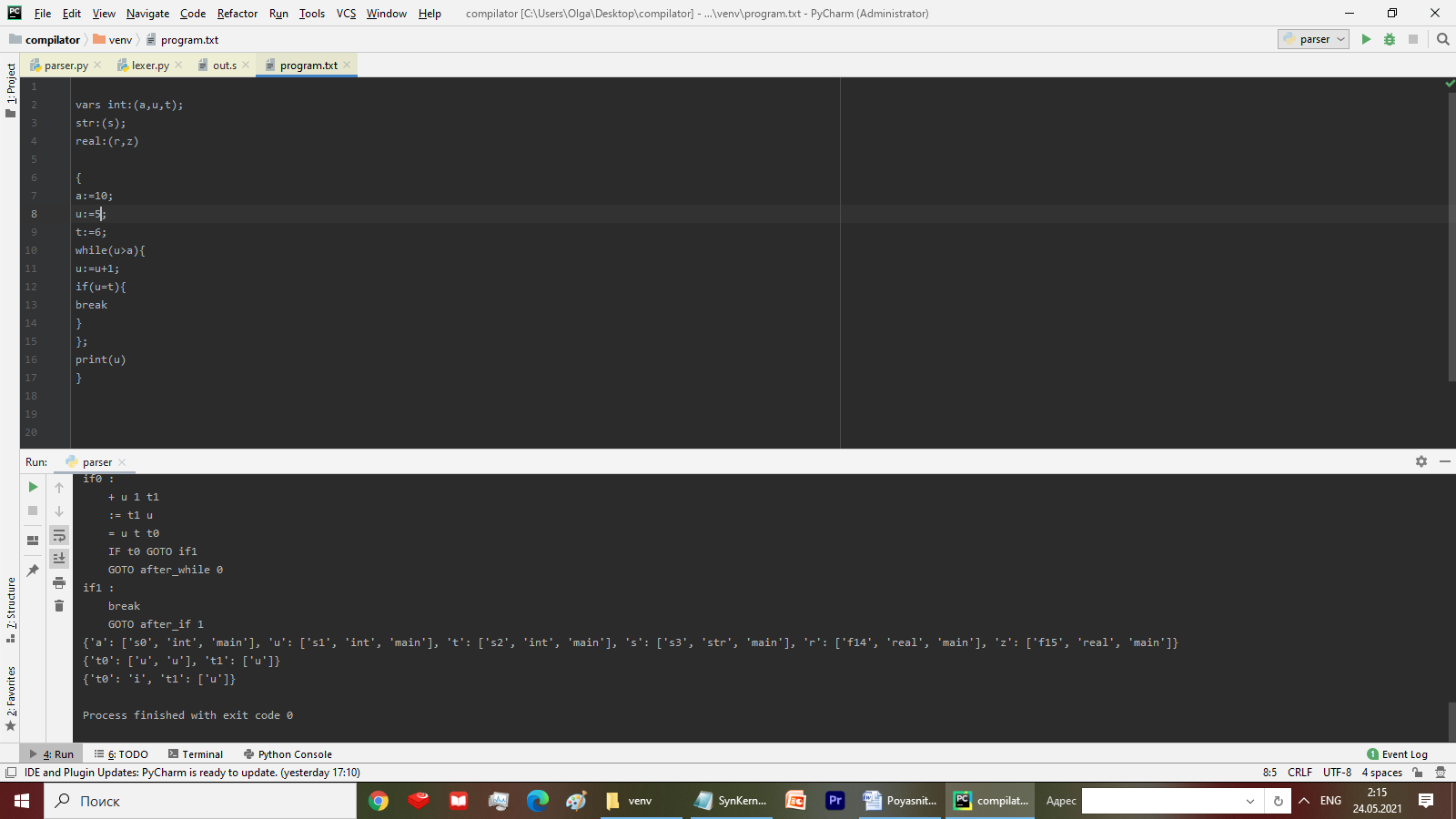


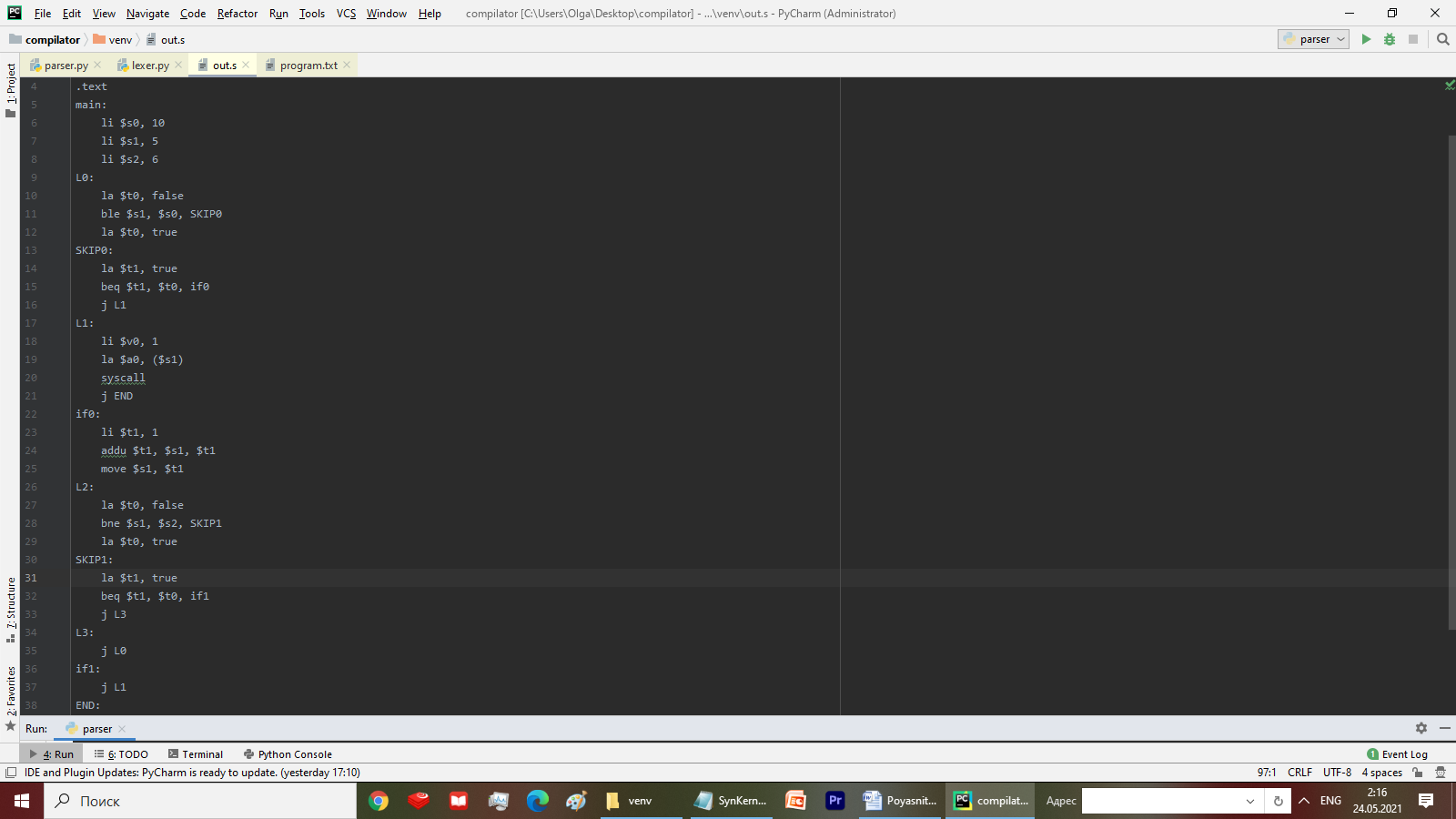
проверка вызова функций и условных операторов, типов данных инт флоат и стринг:

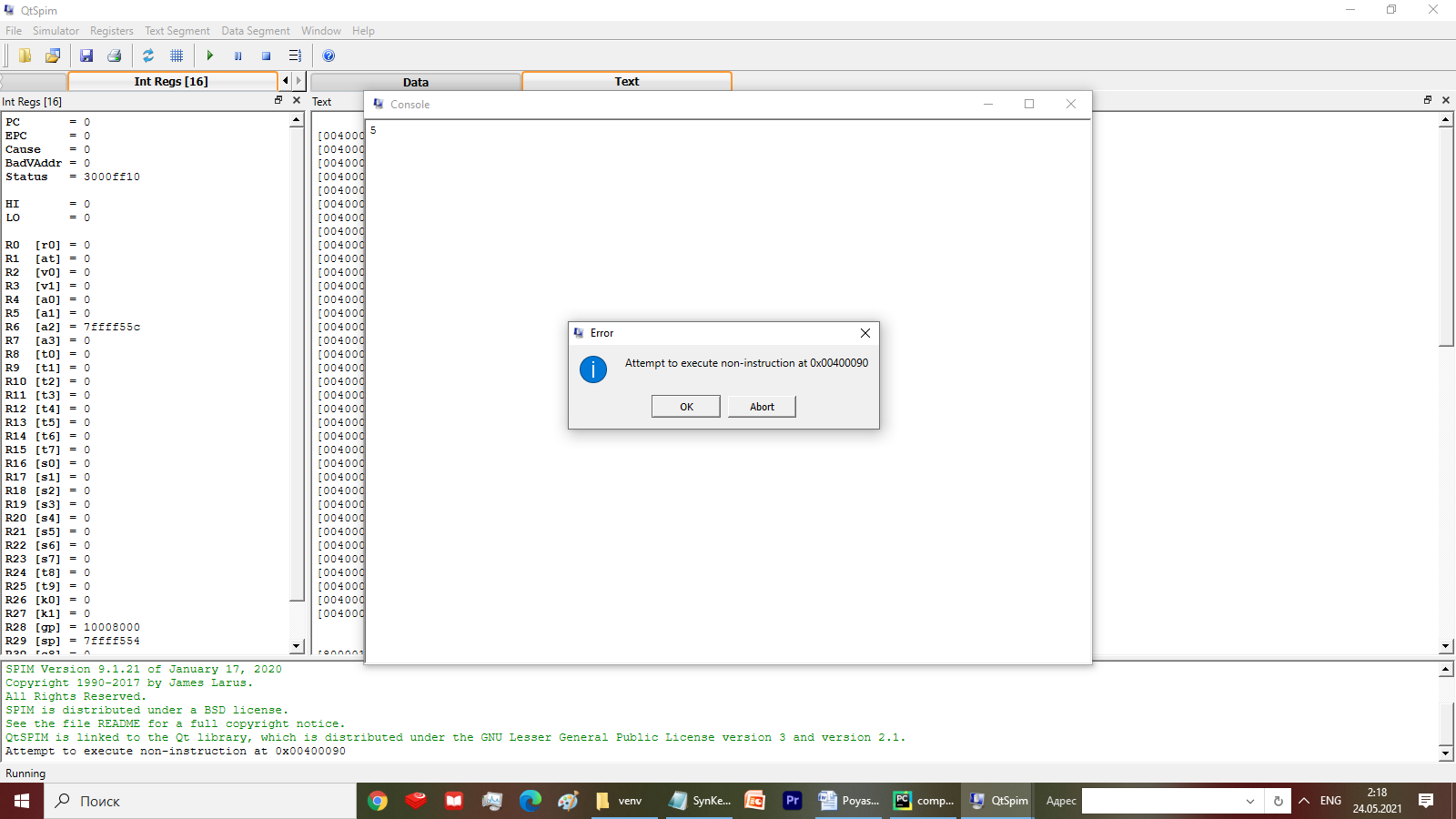


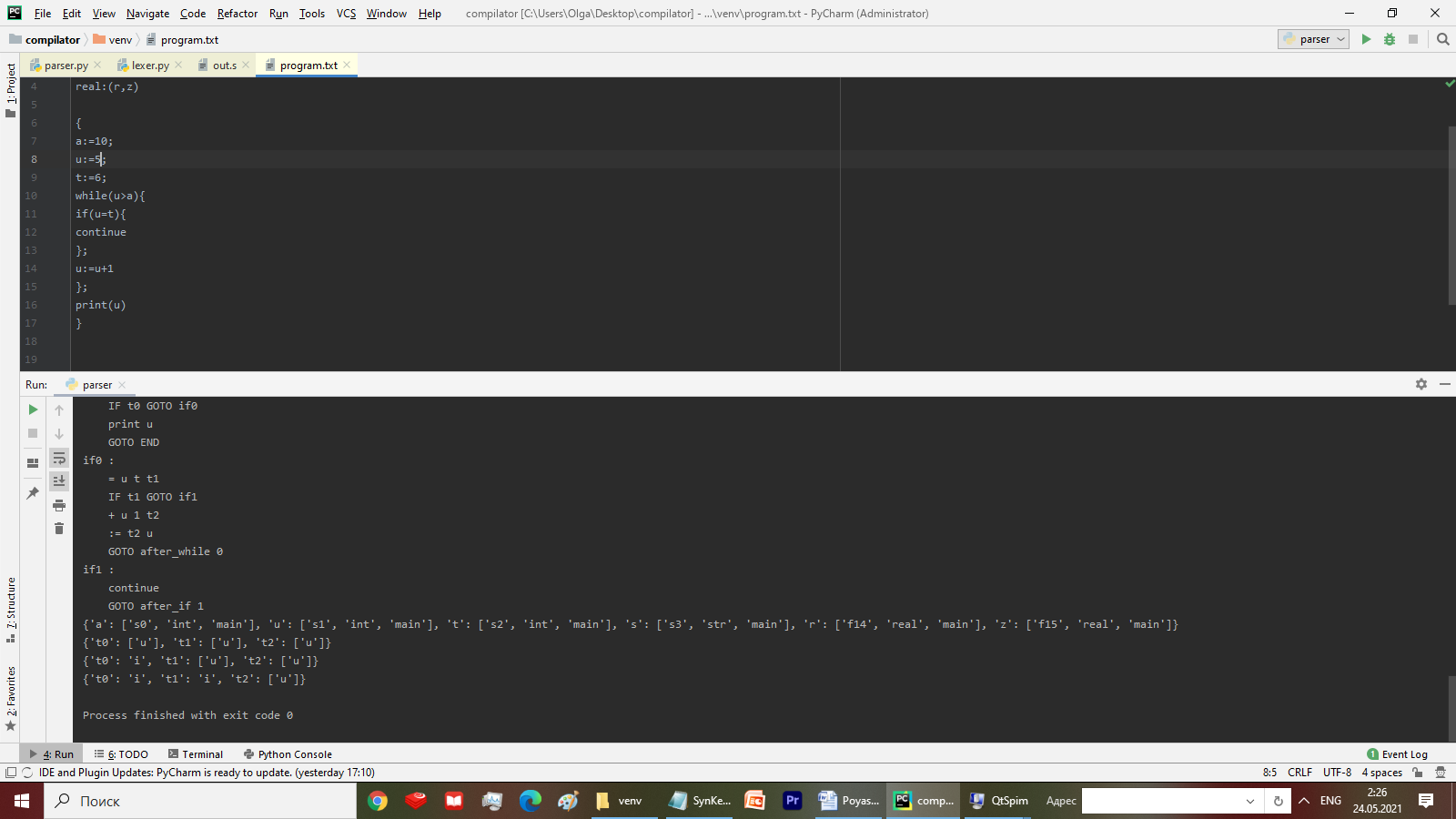


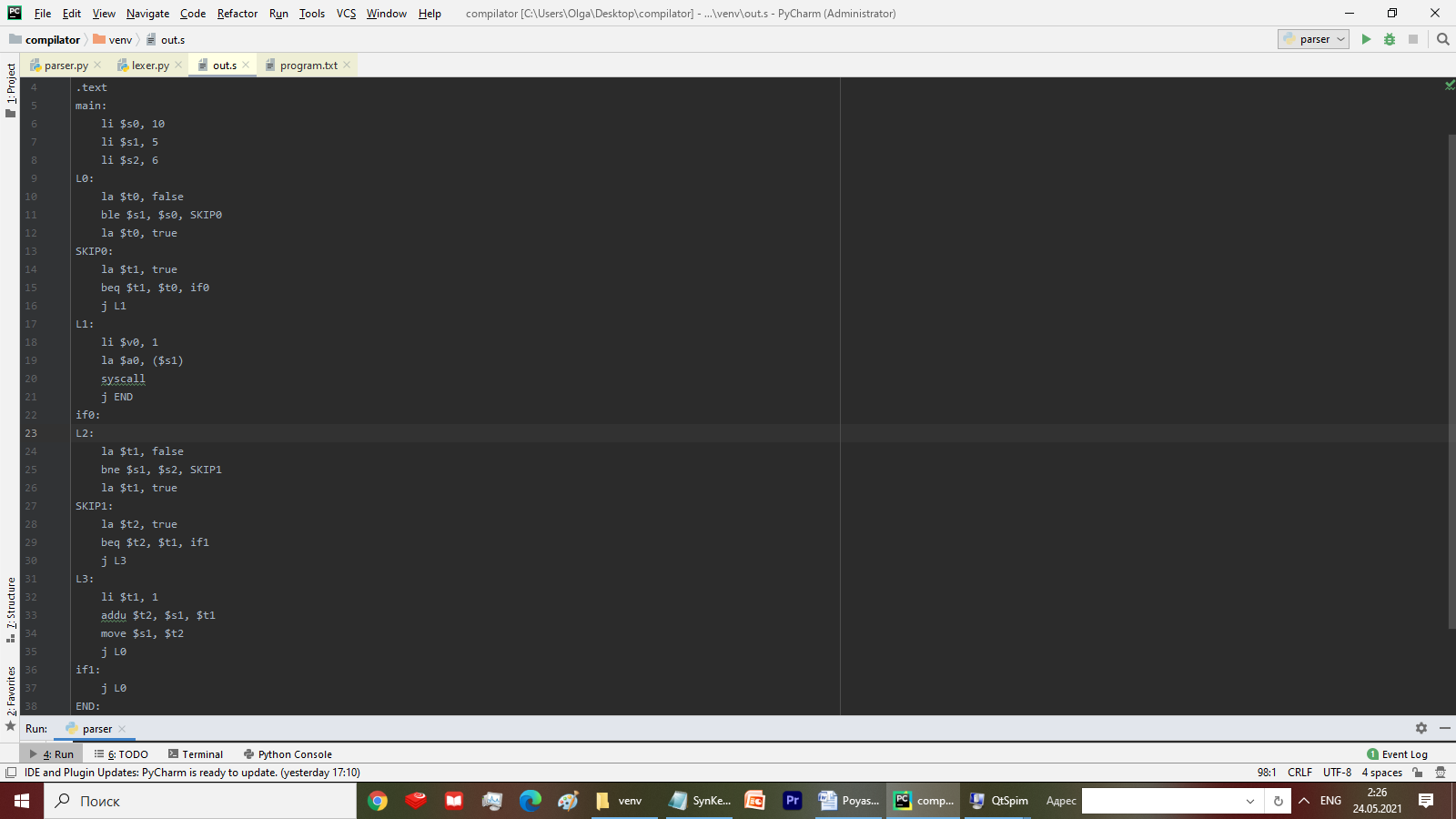
Проверка brake и continue

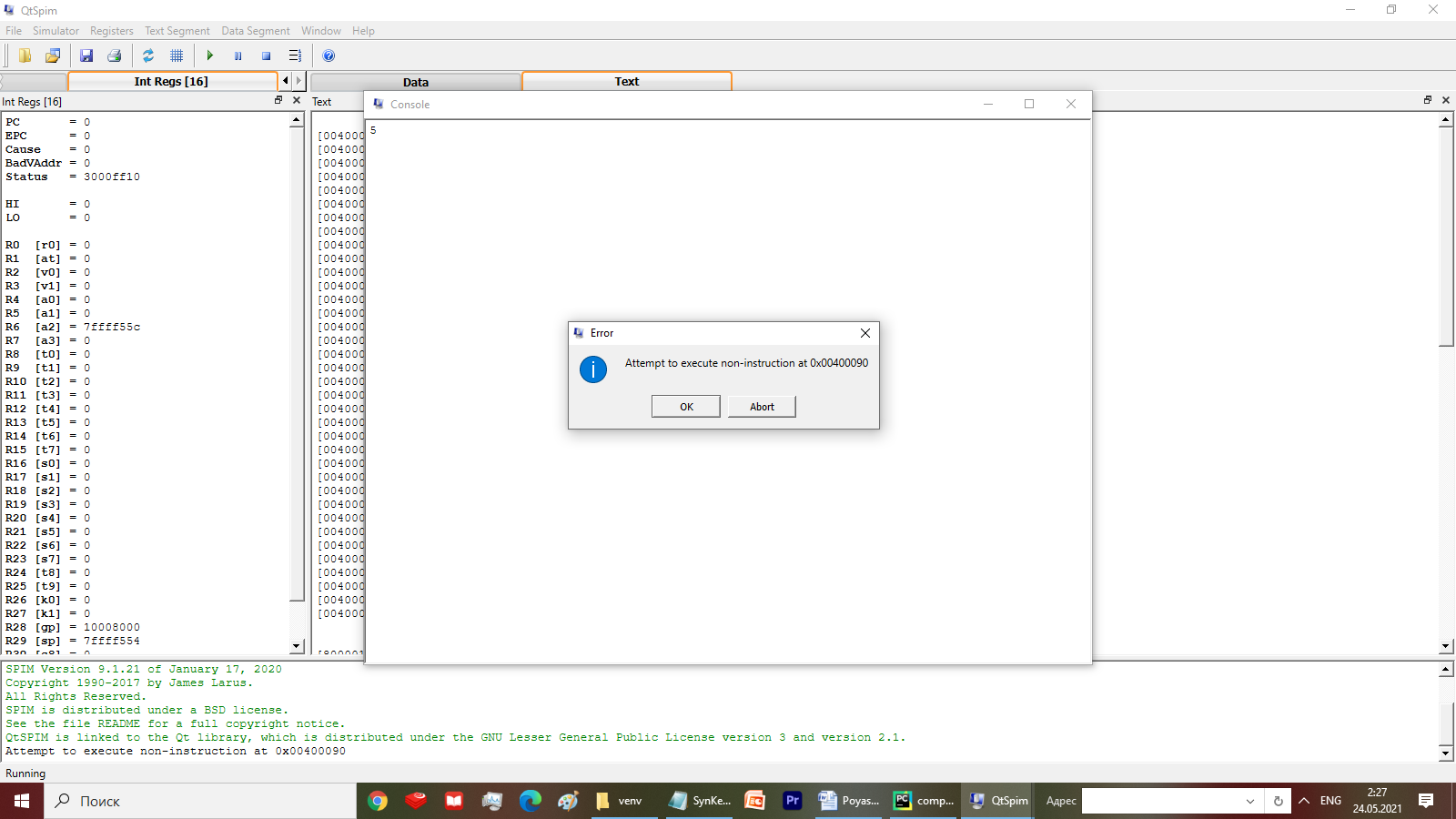












Реквизиты к курсовой работе:

https://github.com/alexonlg/kursach