**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.В. Волков |

**Владимир 2021**

**Оглавление**

[**Аннотация** 3](#_Toc72369769)

[**1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА** 4](#_Toc72369770)

[1.1 Основные требования 4](#_Toc72369771)

[1.2 Лексический анализатор 5](#_Toc72369772)

[1.3 Синтаксический анализатор 6](#_Toc72369773)

[1.4 Построение генератора объектного кода 9](#_Toc72369774)

[2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ 11](#_Toc72369775)

# **Аннотация**

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Python с использованием библиотек rply и llvmlite. Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов виде llvm-кода.

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

* построение лексического анализатора;
* построение синтаксического анализатора;
* построение генератора llvm-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. Арифметические операторы;
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 rply.lexer. Грамматика языка включает в себя элементы из популярных языков, таких как Python, C и др.

Список зарезервированных слов:

* int
* float
* print
* while
* break
* continue
* if
* else
* and
* or

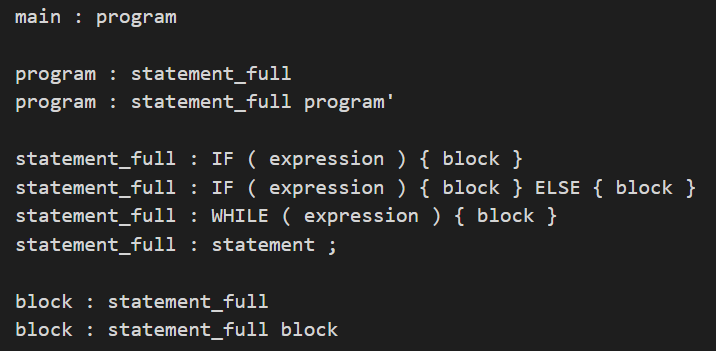
А также прочие символы на подобии *“{}”, “=”, “>”* и тд.

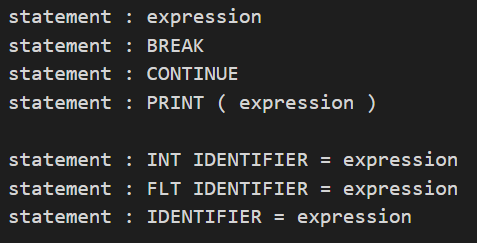
## Синтаксический анализатор

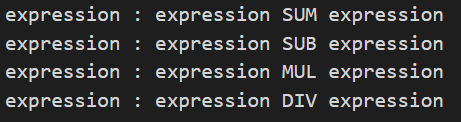
Второй стадией компилятора является синтаксический анализ. На вход синтаксическому анализатору подаётся набор токенов из лексического анализатора. На основе грамматики языка строится дерево разбора грамматики.

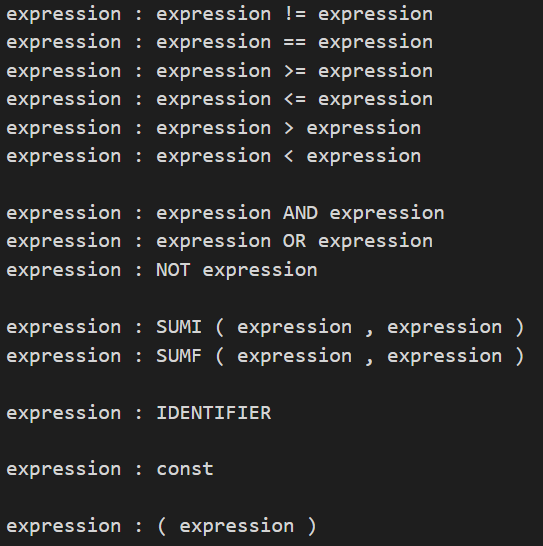
Все взаимодействия во время построения дерева разбора обрабатываются через класс Parser, который возвращает классы модуля AST при совпадении правила из КС-грамматики языка.

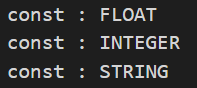
КС-грамматика языка представлена ниже.







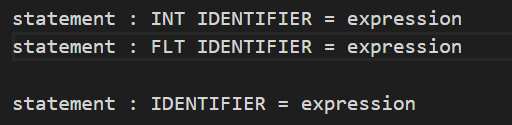




Привила использования грамматики представлены ниже.

**Присвоение**

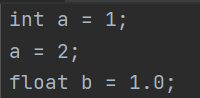
Блок присвоения представляется следующим правилом КС-грамматики:



Если переменная не была определена в программа (то есть ее нет в таблице символов), то указывается тип переменной (поддерживаются 2 типа: *int* – целочисленное и *float* – с плавающей точкой), имя и выражение, которое следует присвоить переменной.

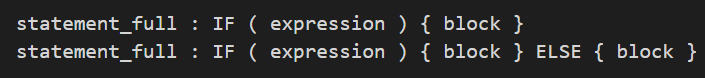
Если переменная уже определена в программе (есть в таблице символов), то тип повторно не указывается, необходимо указать имя переменной и выражение, которое ей присваивается.

Пример:



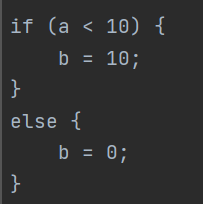
**Ветвление**

Блок ветвления определяется следующим правилом КС-грамматики:



Блок обязательно содержит ключевое слово *if*, условие ветвления в скобках и тело в фигурных скобках. Опционально блок может содержать ветку *else*, которая выполняется в случае, если условие ветвления ложно.

Пример:



**Циклы**

Блок цикла определяется следующим правилом КС-грамматики:

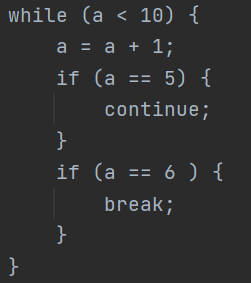


Аналогично блоку ветвления, блок цикла содержит ключевое слово (*while*), условие цикла в скобках и тело цикла в фигурных скобках.

Тело цикла будет выполняться до тех пор, пока условие цикла истинно.

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.

Пример:



**Вызов встроенных функций**

В языке есть встроенные функции, такие как *sum(a, b)* и *sumf(a, b)*.

Пример:

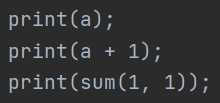


В результате выполнения данной команды в переменной *a* будет храниться значение 4 типа *int*.

**Стандартный вывод**

Встроенная функция *print(a)* позволяет выводить на стандартный вывод выражение *a.*

Пример:



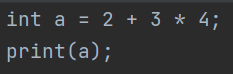
## Построение генератора объектного кода

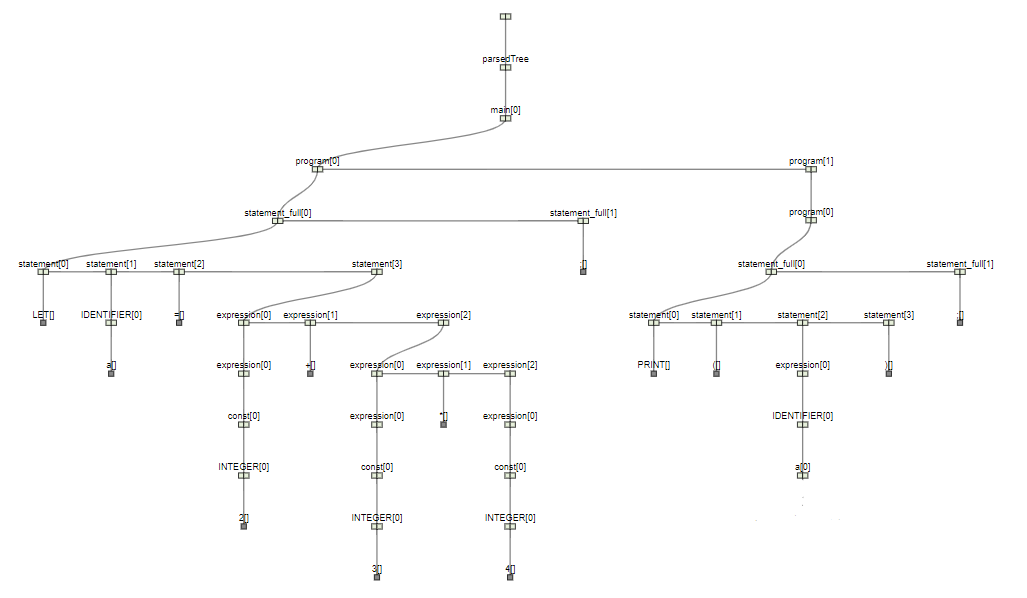
Генерация объектного кода выполняется объектами модуля AST, каждый из которых имеет метод генерации llvm-кода (*eval*).

Независимо от исходного кода в целевой код на языке llvm добавляются встроенные функции языка, в том числе определяется функция стандартного вывода.

В модуле AST определены классы вершин синтаксического дерева. При синтаксическом анализе эти вершины добавляются к корневой вершине *main*.

Пример дерева разбора для программы:



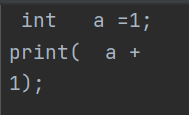


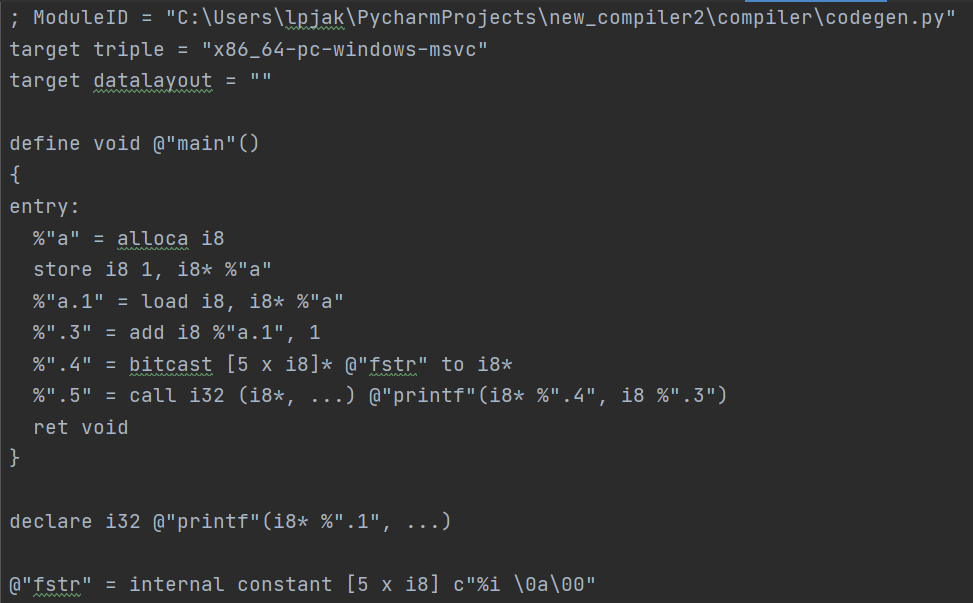
# **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

При проектировании компилятора к основному языку были установлены следующие минимальные требования: наличие операторных скобок, игнорирование пробелов и идентации программы, поддержка многострочных комментариев и вызова функций. Наличие операторов присваивания, условных, цикла, арифметических, логических. Должны присутствовать два типа данных – целочисленный и вещественный.

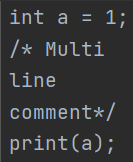
Далее приведено тестирование компилятора.

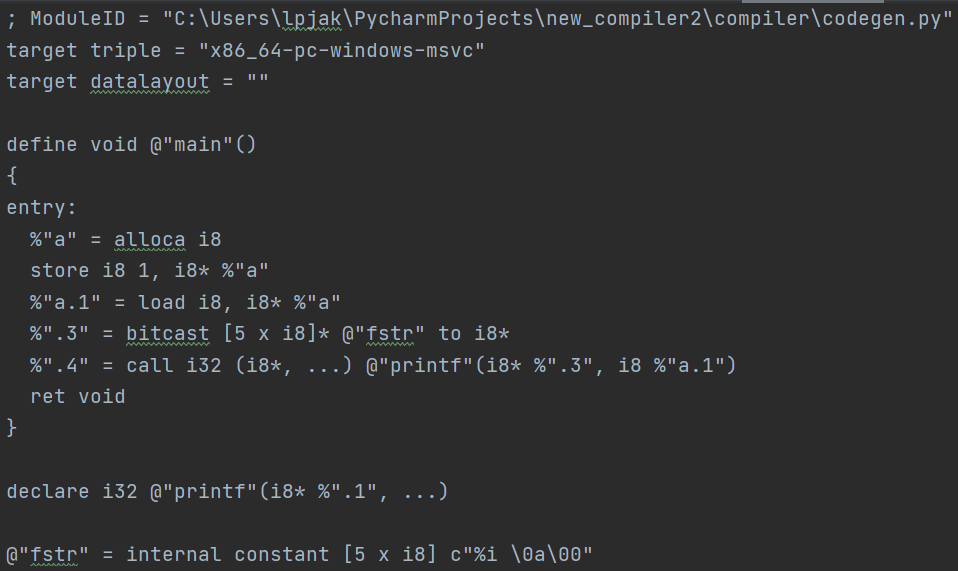
Проверка на игнорирование пробелов и индентацию:



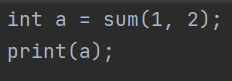


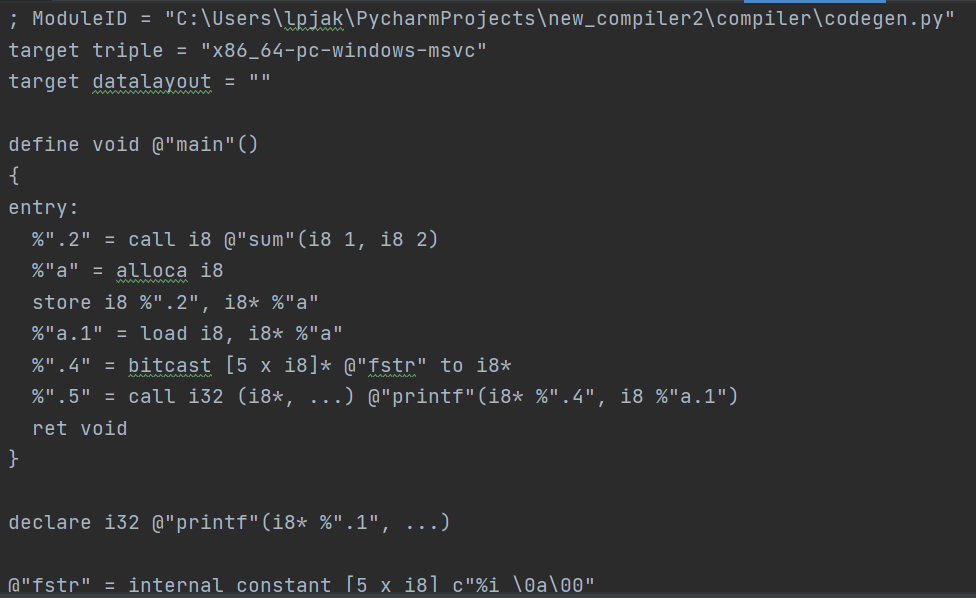
Проверка многострочных комментариев

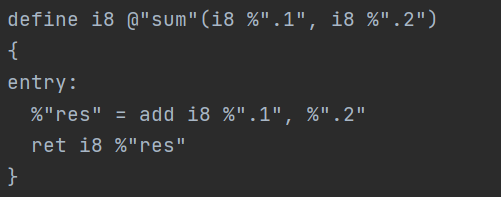




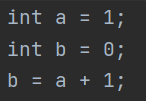
Проверка вызова функций

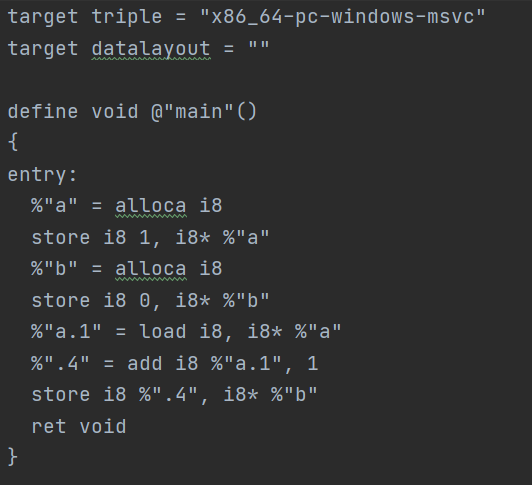




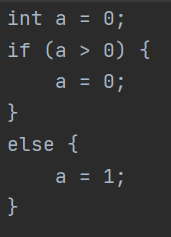


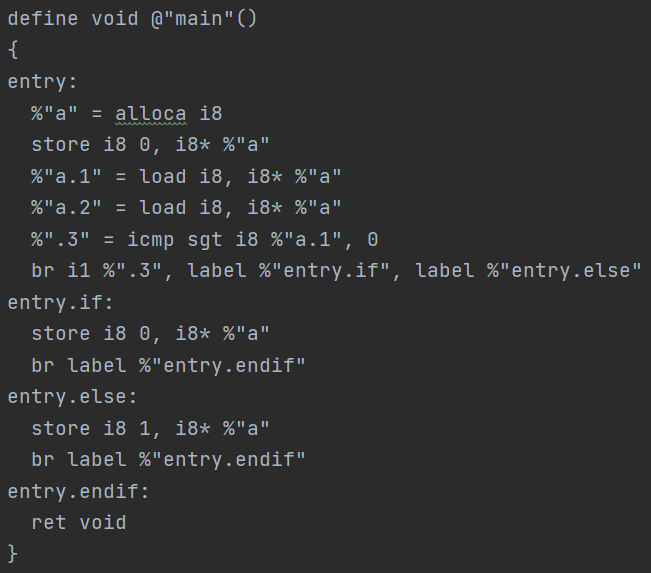
Проверка оператора присваивания:



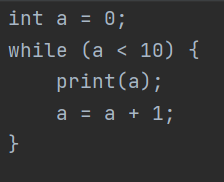


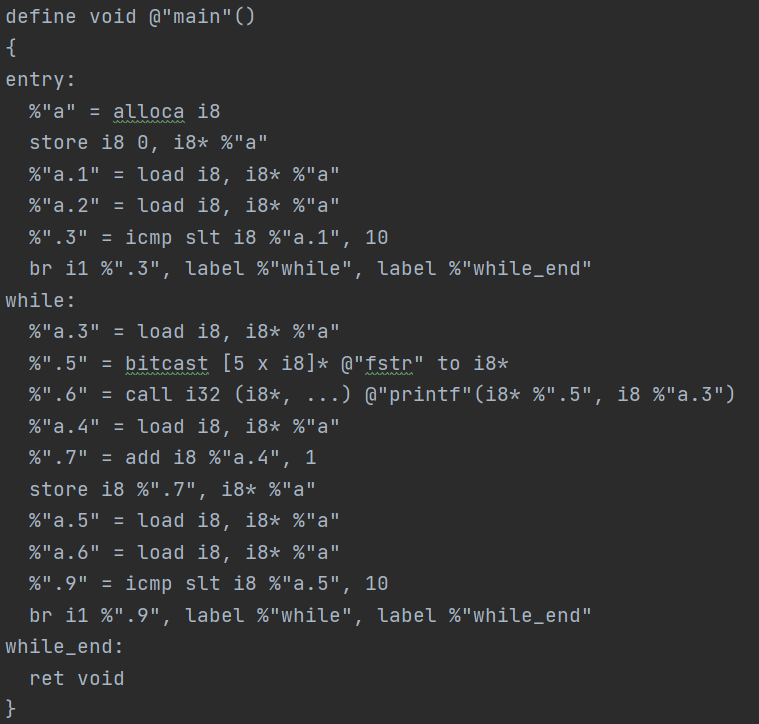
Проверка условного оператора:

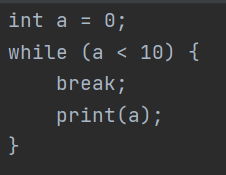


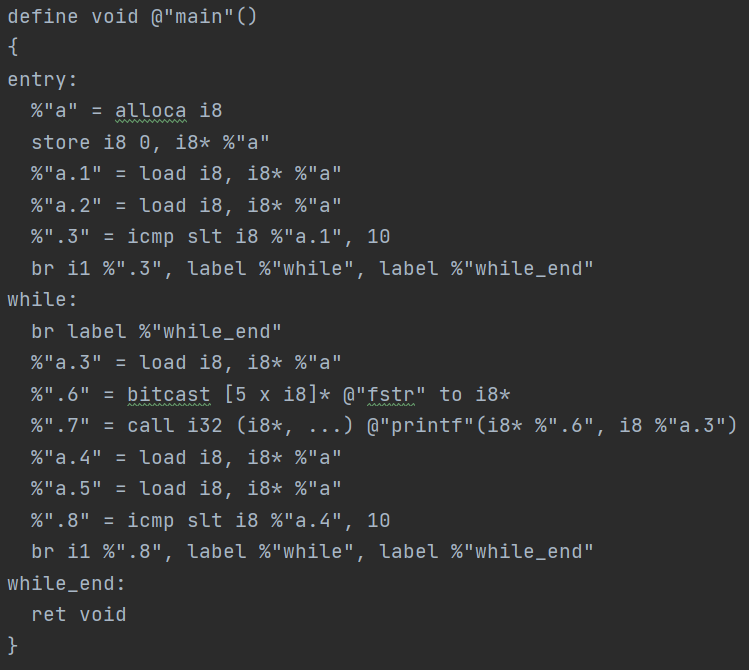


Проверка оператора цикла:



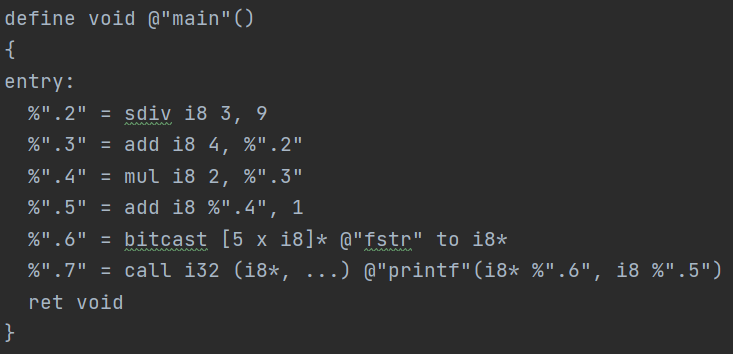




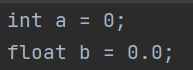


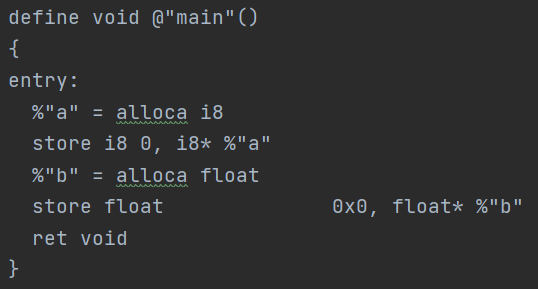
Проверка арифметики:





Проверка типов данных:





Реквизиты к курсовой работе:

<https://github.com/LPJWS/Compiler/tree/master>