



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

БГТУ.СМК-Ф-4.2-К5-01

Факультет

О

Естественнонаучный

шифр

наименование

Кафедра

О7

Информационные системы и программная инженерия

шифр

наименование

Дисциплина

Разработка трансляторов искусственных языков

КУРСОВАЯ РАБОТА

на тему

Создание транслятора с языка С на язык PYTHON
при помощи FLEX и BISON

Выполнил студент группы О726Б

Махов Н.М.

Фамилия И.О.

Устиновский Г.С.

Фамилия И.О.

Подпись

Оценка

«____»

2024г.

САНКТ-ПЕТЕРБУРГ

2024г.

Отчет 20с., 3 рис., 1 таб., 5 источн., 1 прил.

FLEX, BISON, ТРАНСЛЯТОР, C, PYTHON, СИНТАКСИЧЕСКИЙ АНАЛИЗ, ТОКЕН, ЛЕКСИЧЕСКИЙ АНАЛИЗ

Цели исследования: Данная курсовая работа направлена на создание транслятора, осуществляющего преобразование программного кода с языка C на Python с использованием таких инструментов, как FLEX и BISON. В ходе выполнения работы решались задачи, связанные с выполнением лексического анализа исходного кода, его синтаксического разбора и разработкой правил соответствия между синтаксисами C и Python.

Объект исследования: в качестве объекта исследования рассматривается процесс трансляции программного кода между языками программирования C и Python, основанный на применении технологий лексического и синтаксического анализа.

Методы исследования: Методы исследования включают применение FLEX для выполнения лексического анализа, а также использование BISON для синтаксического разбора, что позволяет преобразовывать код с языка C в эквивалентный код на Python. Кроме того, использовались методы проектирования грамматик и установления соответствий между синтаксисами двух языков.

В результате выполнения работы планируется создание транслятора, выполняющего преобразование кода с языка C на Python с помощью инструментов FLEX и BISON. Разработанный транслятор будет включать функционал лексического анализа исходного кода с использованием FLEX и синтаксического разбора посредством BISON.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....	4
ВВЕДЕНИЕ	5
1 Постановка задачи	7
2 Теоретические сведения.....	8
3 Проектирование транслятора	9
3.1 Устройство программы.....	9
4 Реализация транслятора	10
4.1 Файл Tokens.l	10
4.2 Файл Rules.y	11
5 Демонстрация работы программы	17
ЗАКЛЮЧЕНИЕ.....	18
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	19
ПРИЛОЖЕНИЕ А.....	20

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Flex (Fast Lexical Analyzer) — это инструмент для генерации лексических анализаторов. Он служит заменой Lex в системах, основанных на GNU, и обладает схожей функциональностью. Однако Flex не является частью проекта GNU[1].

GNU Bison — программа для автоматической генерации синтаксических анализаторов на основе заданной грамматики. Она распространяется как свободное программное обеспечение, созданное в рамках проекта GNU, и поддерживается на всех традиционных операционных системах[2]. Программа bison во многом совместима с yacc и часто используется совместно с лексическим анализатором flex.

Транслятор — это программа, преобразующая текст программы с одного языка программирования на другой. Чаще всего используется для преобразования программного кода с языка высокого уровня в машинные команды или ассемблер.

Токен — это отдельный элемент программного кода, например, число или лексема.

Парсинг — процесс обработки токенов, заключающийся в их упорядочении в соответствии с синтаксическими и другими правилами языка программирования.

ВВЕДЕНИЕ

В условиях развития технологий программирование требует применения надежных и эффективных средств для преобразования исходного высокоуровневого кода, ориентированного на специфические области, в низкоуровневый код, максимально приближенный к машинному языку. В этом аспекте разработка трансляторов становится важным элементом, обеспечивающим автоматизацию преобразования кода с одного языка программирования на другой.

Язык C[3] представляет собой высокоуровневый язык программирования, активно применяемый в системах промышленной автоматизации и контроллерах. Тем не менее, несмотря на широкое распространение в данной сфере, C может испытывать определенные трудности при интеграции с языками программирования, такими как Python[4], которые обычно используются при создании систем искусственного интеллекта. В таблице 1 показаны основные различия в синтаксисе между этими языками.

Различия языков показаны на рисунке 1.

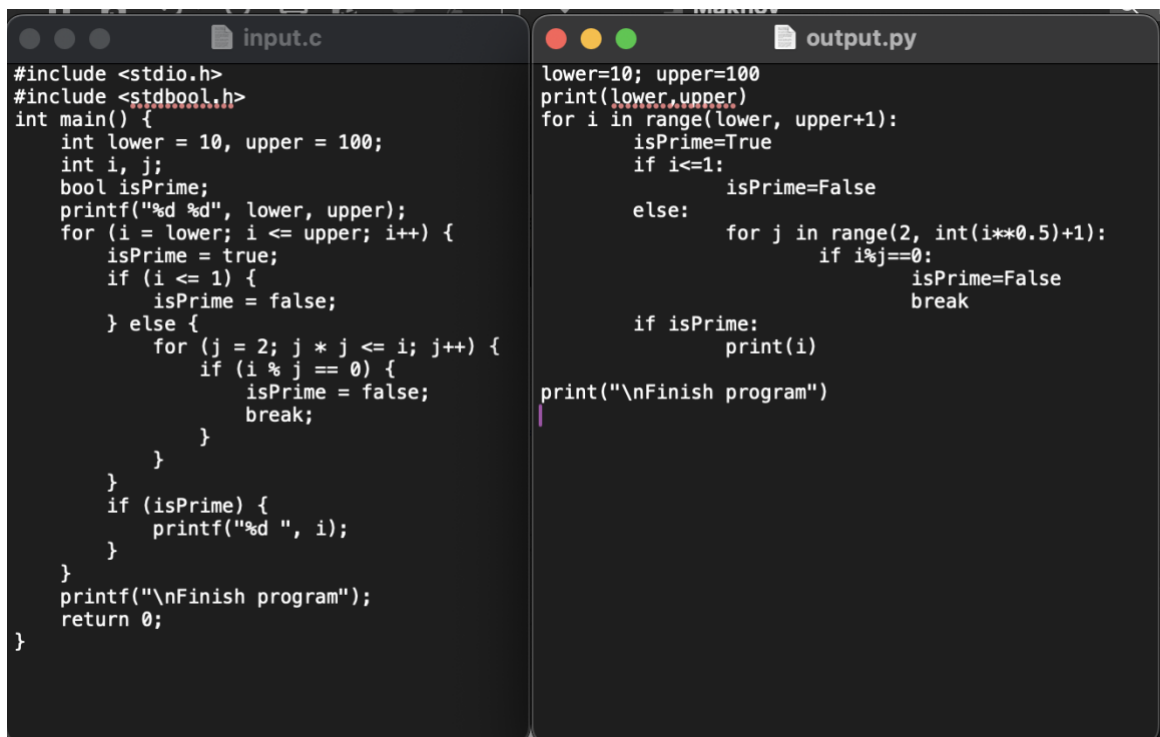


Рисунок 1 – Сравнение синтаксиса C и Python

Таблица 1 – основные различия синтаксиса языков С и Python

	С	Python
Синтаксис объявления переменной	<тип> <имя>;	<имя>
Операторные скобки	{ }	нет
Условие ЕСЛИ	if (<условие>) <действие>;	if <условие> : <действие>
Цикл со счетчиком	for(<переменная> = <значение>; <условие>; <действие после исполнения тела цикла>)	for <переменная> in range(<условие>) : <действие>

1 Постановка задачи

Ключевая цель курсовой работы – создание транслятора языка C на язык Python, используя LEX(FLEX) и BISON.

2 Теоретические сведения

Транслятор — это программа, предназначенная для преобразования исходного кода, написанного на одном языке программирования, в код на другом языке. Выделяют три основных вида трансляторов: компиляторы (переводят исходный код в машинные команды), интерпретаторы (выполняют код напрямую, без его предварительного преобразования) и трансляторы (переносят код между разными высокоуровневыми языками). В рамках данного проекта разрабатывается транслятор, выполняющий преобразование кода с языка C на Python.

Flex (Fast Lexical Analyzer) — инструмент, используемый для автоматического создания лексических анализаторов. Лексический анализатор (или сканер) разделяет исходный текст программы на токены — минимальные смысловые элементы, такие как ключевые слова, идентификаторы, операторы и прочие.

Bison — это утилита для создания синтаксических анализаторов. Синтаксический анализатор (парсер) принимает последовательность токенов, сформированную лексическим анализатором, проверяет их на соответствие грамматике языка и выполняет синтаксические преобразования.

3 Проектирование транслятора

3.1 Устройство программы

Два основных этапа работы программы -

- а) разбиение исходного текста на токены с помощью FLEX
- б) парсинг токенов с помощью BISON

Файл FLEX имеет расширение .l и содержит в себе определения токенов, поступающих на обработку в BISON. Файл BISON имеет расширение .y и содержит в себе правила парсинга токенов, описание грамматики языка и код обработки входного файла.

Запустив Bison с ключом -d, получим .h-файл с перечислением всех терминалов, поэтому в .y-файле Bison объявим все возможные терминалы, а в .l-файле включим полученный заголовок.

Так flex-парсер проходит через входной текст, передавая обработанную информацию Bison-парсеру.

4 Реализация транслятора

Транслятор состоит из двух основных файлов[5]:

- Файл с описанием токенов,
- Файл с правилами обработки этих токенов.

4.1 Файл Tokens.l

Определение всех лексем происходит в файле .l. Далее будут рассмотрены все лексемы, которые описываются в программе.

При чтении латиницы из входного файла программа будет считывать некоторые символы или их последовательности и передавать их в качестве токенов.

Название переменных обрабатываются с помощью регулярного выражения, которое подразумевает последовательность букв или цифр, но при этом цифра не может стоять на первом месте: `[a-zA-Z]*`

Далее нам необходимо обработать целочисленное значение. Оно может быть как отрицательным, так и положительным, поэтому регулярное выражение выглядит следующим образом: `[0-9]*`

Также с помощью регулярного выражения мы должны отследить ввод арифметических операций, что реализовано в файле.

Помимо вышеперечисленных правил программа будет считывать следующие лексемы и считать их ключевыми словами:

- `for`,
- `int`,
- `bool`,
- `"printf"`,
- `"#include <stdio.h >"`,
- `"#include <stdboo.h >"`,
- `"int main()"`,
- `return`,
- `if`,
- `else`.

Содержимое файла Tokens.l:

```
%{
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "Rules.tab.h"
}%
%option noyywrap
%%
"int"           { strcpy(yylval.var, yytext); return INT; }
"bool"          { strcpy(yylval.var, yytext); return BOOL; }
"#include <stdio.h>" { strcpy(yylval.var, yytext); return INCLUDE; }
"#include <stdbool.h>" { strcpy(yylval.var, yytext); return INCLUDEBOOL; }
"int main()"    { strcpy(yylval.var, yytext); return MAIN; }
"printf"        { strcpy(yylval.var, yytext); return PRINT; }
"if"            { strcpy(yylval.var, yytext); return IF; }
"else"          { strcpy(yylval.var, yytext); return ELSE; }
"break"         { strcpy(yylval.var, yytext); return BREAK; }
"for"           { strcpy(yylval.var, yytext); return FOR; }
"return"        { strcpy(yylval.var, yytext); return RETURN; }
"true"          { strcpy(yylval.var, yytext); return TRUE; }
"false"         { strcpy(yylval.var, yytext); return FALSE; }
\"([^\\"\\]|\\.)*\" { strcpy(yylval.var, yytext); return STRING; }
[a-zA-Z]*       { strcpy(yylval.var, yytext); return LETTER; }
[0-9]*          { strcpy(yylval.var, yytext); return NUMBER; }
"="             { strcpy(yylval.var, yytext); return ASSIGN; }
":="            { strcpy(yylval.var, yytext); return DPOINT; }
"{"             { strcpy(yylval.var, yytext); return LBRACE; }
"}"             { strcpy(yylval.var, yytext); return RBRACE; }
"("             { strcpy(yylval.var, yytext); return LBRACKET; }
")"             { strcpy(yylval.var, yytext); return RBRACKET; }
","             { strcpy(yylval.var, yytext); return COMMA; }
"%"             { strcpy(yylval.var, yytext); return MOD; }
"++"            { strcpy(yylval.var, yytext); return PP; }
"+"             { strcpy(yylval.var, yytext); return P; }
"*"             { strcpy(yylval.var, yytext); return MULTY; }
"<"             { strcpy(yylval.var, yytext); return LESS; }
"<="            { strcpy(yylval.var, yytext); return LESSEQ; }
"=="            { strcpy(yylval.var, yytext); return EQ; }
";"
[ \t\n]+
"/*"([^\*]|\\*+[^*/])*\*+/"
"//".*
%%
```

4.2 Файл Rules.y

Описание грамматики происходит в файле .y. Далее будут рассмотрены правила созданного подмножества языка C.

В программе реализованы правила для начала программы; блока объявлений; работы с переменными, куда включена обработка цикла FOR и так далее.

Главным не терминалом в программе является START, правило которого подразумевает наличие остальных не терминалов:

- ident_list,
- ident,
- print,
- cycle,
- statement,
- values,
- typeOf,
- comp,
- expression,
- comparison,
- body,

Нетерминал start при встрече терминала START выводит записывает переведенный код в выходной файл.

Содержимое правил из файла Rules.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern int yylex();
void yyerror(char *s) { printf("%s\n", s); }
%}

%union {
    int number;
    char var[1000];
}
```

```
%token <var> INT BOOL INCLUDE INCLUDEBOOL MAIN PRINT IF ELSE BREAK RETURN FOR
%token <var> ASSIGN LBRACE RBRACE LBRACKET RBRACKET COMMA EQ P PP MOD LESS LESSEQ
DPOINT MULTY
%token <var> LETTER NUMBER STRING TRUE FALSE
```

```
%type <var> START ident_list ident print cycle body body2 body3 body4 statement values
typeOf comp expression comparison
```

```
%start START
```

```
%%
```

```
START: INCLUDE INCLUDEBOOL MAIN LBRACE ident_list print cycle print RETURN values
RBRACE {
```

```
    printf("%s\n", $5);
    printf("%s\n", $6);
    printf("%s\n", $7);
    printf("%s\n", $8);
};
```

```
ident_list: ident {
    strcpy($$, $1);
}
| ident_list ident {
    strcpy($$, $1);
    strcat($$, $2);
};
```

```
ident: typeOf values comp values {
    strcpy($$, $2);
    strcat($$, $3);
    strcat($$, $4);
}
| typeOf values COMMA values {
    strcpy($$, "");
}
| typeOf values comp values COMMA values comp values {
    strcpy($$, $2);
    strcat($$, $3);
    strcat($$, $4);
    strcat($$, "; ");
    strcat($$, $6);
    strcat($$, $7);
    strcat($$, $8);
}
| typeOf values {
    strcpy($$, "");
};
```

```
values: NUMBER {
    strcpy($$, $1);
}
| TRUE {
    strcpy($$, "True");
}
| FALSE {
    strcpy($$, "False");
}
| LETTER {
    strcpy($$, $1);
};
```

```

typeOf: INT {
    strcpy($$, $1);
}
| BOOL {
    strcpy($$, $1);
};

print: PRINT LBRACKET STRING COMMA values COMMA values RBRACKET {
    strcpy($$, "print(");
    strcat($$, $5);
    strcat($$, ",");
    strcat($$, $7);
    strcat($$, ")");
}
| PRINT LBRACKET STRING RBRACKET {
    strcpy($$, "print(");
    strcat($$, $3);
    strcat($$, ")\n");
};

cycle: FOR LBRACKET values comp values values comp values values PP RBRACKET LBRACE
body RBRACE {
    strcpy($$, $1);
    strcat($$, " ");
    strcat($$, $3);
    strcat($$, " in range(");
    strcat($$, $5);
    strcat($$, ", ");
    strcat($$, $8);
    strcat($$, "+1):\n");
    strcat($$, $13);
};

comp: LESS {
    strcpy($$, $1);
}
| LESSEQ {
    strcpy($$, $1);
}
| EQ {
    strcpy($$, $1);
}
| ASSIGN {
    strcpy($$, $1);
};

body: statement {
    strcpy($$, "\t");
    strcat($$, $1);
    strcat($$, "\n");
}
| body statement {
    strcpy($$, $1);
    strcat($$, "\t");
    strcat($$, $2);
    strcat($$, "\n");
};

body2: statement {
    strcpy($$, "\n\t\t");
    strcat($$, $1);

```

```

    }
    | body2 statement {
        strcpy($$, $1);
        strcat($$, "\n\t\t");
        strcat($$, $2);
    };

body3: statement {
    strcpy($$, "\n\t\t\t");
    strcat($$, $1);
}
| body3 statement {
    strcpy($$, $1);
    strcat($$, "\n\t\t\t");
    strcat($$, $2);
};

body4: statement {
    strcpy($$, "\n\t\t\t\t");
    strcat($$, $1);
}
| body4 statement {
    strcpy($$, $1);
    strcat($$, "\n\t\t\t\t");
    strcat($$, $2);
};

statement: IF LBRACKET comparison RBRACKET LBRACE body2 RBRACE {
    strcpy($$, $1);
    strcat($$, " ");
    strcat($$, $3);
    strcat($$, ":");
    strcat($$, $6);
}
| IF LBRACKET values MOD values comp values RBRACKET LBRACE body4 RBRACE {
    strcpy($$, $1);
    strcat($$, " ");
    strcat($$, $3);
    strcat($$, $4);
    strcat($$, $5);
    strcat($$, $6);
    strcat($$, $7);
    strcat($$, ":");
    strcat($$, $10);
}
| values comp values {
    strcpy($$, $1);
    strcat($$, $2);
    strcat($$, $3);
}
| ELSE LBRACE body2 RBRACE {
    strcpy($$, $1);
    strcat($$, ":");
    strcat($$, $3);
}
| FOR LBRACKET values comp values expression comp values values PP RBRACKET LBRACE
body3 RBRACE {
    strcpy($$, $1);
    strcat($$, " ");
    strcat($$, $3);
    strcat($$, " in range(");

```

```

        strcat($$, $5);
        strcat($$, ", ");
        strcat($$, "int(i**0.5)+1:");
        strcat($$, $13);
    }
| BREAK {
    strcpy($$, $1);
}
| PRINT LBRACKET STRING COMMA values RBRACKET {
    strcpy($$, "print(");
    strcat($$, $5);
    strcat($$, ",");
};

expression: values {
    strcpy($$, $1);
}
| values MULTY values {
    strcpy($$, $1);
    strcat($$, $2);
    strcat($$, $3);
};

comparison: values {
    strcpy($$, $1);
}
| values comp values {
    strcpy($$, $1);
    strcat($$, $2);
    strcat($$, $3);
};

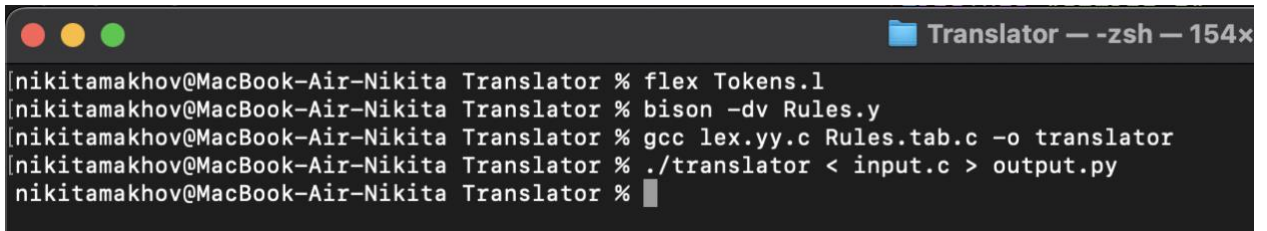
%%

int main() {
    yyparse();
}

```


5 Демонстрация работы программы

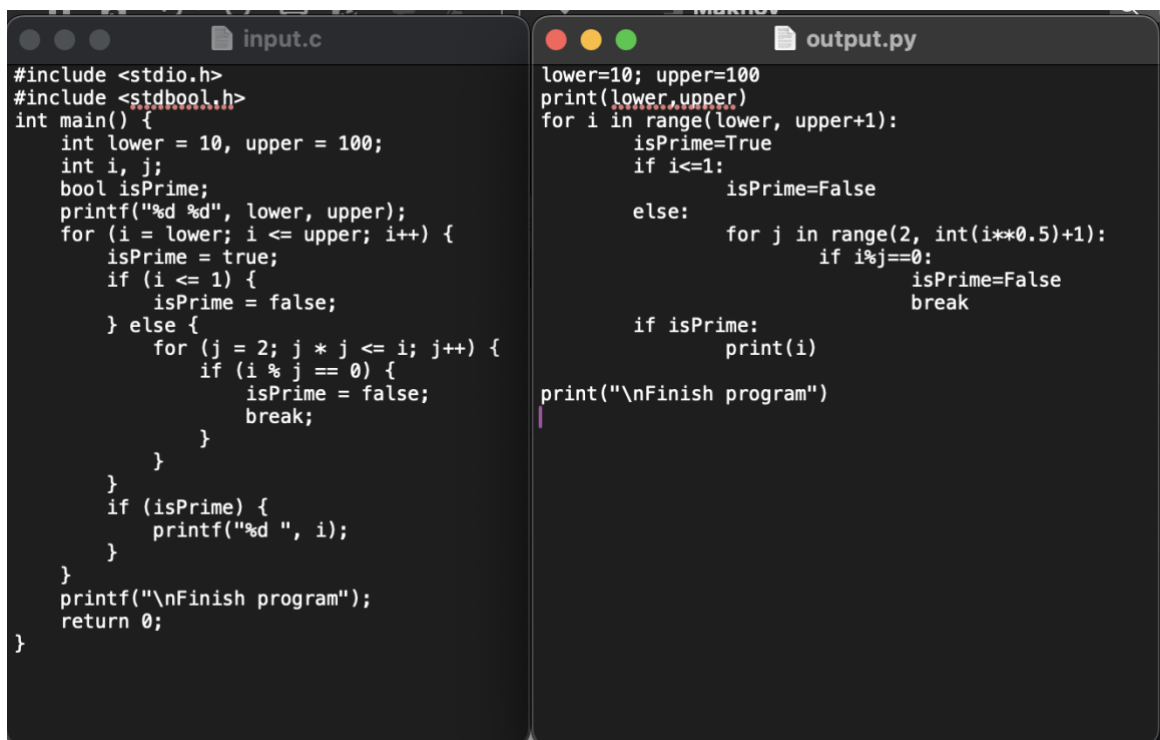
На рисунке 2 команды для компиляции программы.

A terminal window titled "Translator — -zsh — 154x" showing a series of commands and their outputs. The user is at a prompt on a MacBook-Air-Nikita. The commands are: flex Tokens.l, bison -dv Rules.y, gcc lex.yy.c Rules.tab.c -o translator, and ./translator < input.c > output.py.

```
[nikitamakhov@MacBook-Air-Nikita Translator % flex Tokens.l
[nikitamakhov@MacBook-Air-Nikita Translator % bison -dv Rules.y
[nikitamakhov@MacBook-Air-Nikita Translator % gcc lex.yy.c Rules.tab.c -o translator
[nikitamakhov@MacBook-Air-Nikita Translator % ./translator < input.c > output.py
[nikitamakhov@MacBook-Air-Nikita Translator % ]
```

Рисунок 2 – Компиляции программы

На рисунке 3 продемонстрирован результат работы транслятора.

Two side-by-side code editors. The left editor, titled "input.c", shows C code for finding primes between 10 and 100. The right editor, titled "output.py", shows the equivalent Python code. Both programs print the range and the primes found, and then print a finish message.

```
input.c
#include <stdio.h>
#include <stdbool.h>
int main() {
    int lower = 10, upper = 100;
    int i, j;
    bool isPrime;
    printf("%d %d", lower, upper);
    for (i = lower; i <= upper; i++) {
        isPrime = true;
        if (i <= 1) {
            isPrime = false;
        } else {
            for (j = 2; j * j <= i; j++) {
                if (i % j == 0) {
                    isPrime = false;
                    break;
                }
            }
        }
        if (isPrime) {
            printf("%d ", i);
        }
    }
    printf("\nFinish program");
    return 0;
}

output.py
lower=10; upper=100
print(lower,upper)
for i in range(lower, upper+1):
    isPrime=True
    if i<=1:
        isPrime=False
    else:
        for j in range(2, int(i**0.5)+1):
            if i%j==0:
                isPrime=False
                break
    if isPrime:
        print(i)
print("\nFinish program")
```

Рисунок 3 – Результат работы транслятора

ЗАКЛЮЧЕНИЕ

В ходе данной работы был разработан транслятор, предназначенный для перевода программного кода с языка C в эквивалентный код на языке программирования Python. Процесс разработки включал в себя использование инструментов лексического анализа и синтаксического разбора, таких как FLEX и BISON, для построения эффективного и надежного транслятора.

Одной из ключевых задач работы было создание грамматики, описывающей синтаксис языка C, а также определение правил сопоставления для перевода этой грамматики в язык Python. Реализованный транслятор успешно выполняет процесс анализа и преобразования кода, обеспечивая тем самым переносимость программ с языка C на язык Python.

В процессе работы получены знания в области программирования на языке C, изучен его синтаксис. Так же были улучшены навыки создания трансляторов искусственных языков высокого уровня на примере языка программирования C.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Официальная страница проекта FLEX. – URL:
<https://github.com/westes/flex>
- 2) Официальная страница GNU Bison. – URL:
<http://www.gnu.org/software/bison/>
- 3) Документация C. – URL: <https://learn.microsoft.com/ru-ru/?view=msvc-170>
- 4) Документация Python. – URL: <https://docs.python.org/3/>
- 5) Видеоуроки по созданию транслятора. – URL:
<https://www.youtube.com/watch?v=POjnw0xEVas&list=PLIrl0f9NJZy4oOOAVPU6MyRdFjJFGtceu>

ПРИЛОЖЕНИЕ А

Исходные файлы проекта

Исходные файлы проекта предоставлены на электронном носителе.