

Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого

—
Институт прикладной математики и механики
Кафедра «Информационная безопасность компьютерных систем»

ЛАБОРАТОРНАЯ РАБОТА № 2

«Калькулятор полиномов»

по дисциплине «Формальные грамматики и теория компиляторов»

Выполнил
студент гр. 33609/3

<подпись>

Дарсигов Р.Б.

студент гр. 33609/3

<подпись>

Елисеев Н.Н.

Преподаватель

<подпись>

Семьянов П.В.

Санкт-Петербург
2018

1 ЦЕЛЬ РАБОТЫ

Изучить принципы составления грамматик и создания лексического анализатора.

Используя синтаксический анализатор YACC (Bison), разработать языковую грамматику, позволяющую обрабатывать полиномы.

Реализованный калькулятор полиномов должен поддерживать работу с операциями + (сложение), – (вычитание), * (умножение), () (скобки), ^ (возведение в степень).

Запись полиномов должна быть приближена к записи в классической математике.

Разработать синтаксис языка, поддерживающий базовые языковые конструкции:

- переменные типа полином и работа с ними;
- оператор вывода на печать;
- поддержка комментариев;
- сообщения об ошибках (лексические, синтаксические, семантические).

2 РЕЗУЛЬТАТЫ РАБОТЫ

В результате работы создан калькулятор полиномов, поддерживающий работу с заданными операциями.

Для хранения полиномов реализованы следующие структуры:

```
struct _monomial
{
    int    coefficient;    // коэффициент
    char   * variable;    // имя переменной
    int    power;         // степень
};

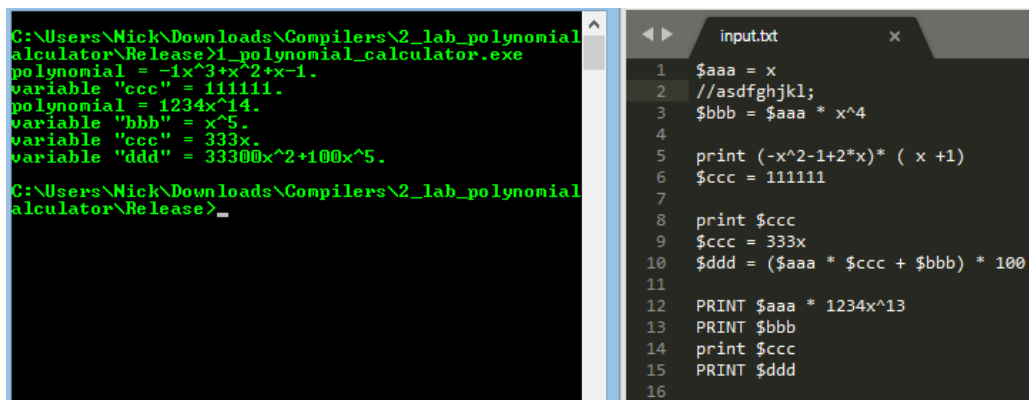
struct _polynomial
{
    struct _node * begin;    // указатель на первый одночлен в полиноме
    int    count;          // число элементов в полиноме
};

struct _node
{
    struct _monomial item;    // одночлен
    struct _node * next;    // указатель на след. узел
    struct _node * prev;    // указатель на пред. узел
};

struct _variable
{
    char   * variable;    // имя переменной
    struct _polynomial polynomial; // копия полинома
    struct _variable * next; // указатель на следующую переменную
    struct _variable * prev; // указатель на следующую переменную
};
```

Поскольку полином (многочлен) – это множество одночленов, создан список одночленов, структура полинома хранит указатель на первый одночлен.

Структура `_variable` используется для поддержки переменных типа полином.



```
C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>1_polynomial_calculator.exe
polynomial = -1x^3+x^2+x-1.
variable "ccc" = 11111.
polynomial = 1234x^14.
variable "bbb" = x^5.
variable "ccc" = 333x.
variable "ddd" = 33300x^2+100x^5.
C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>_

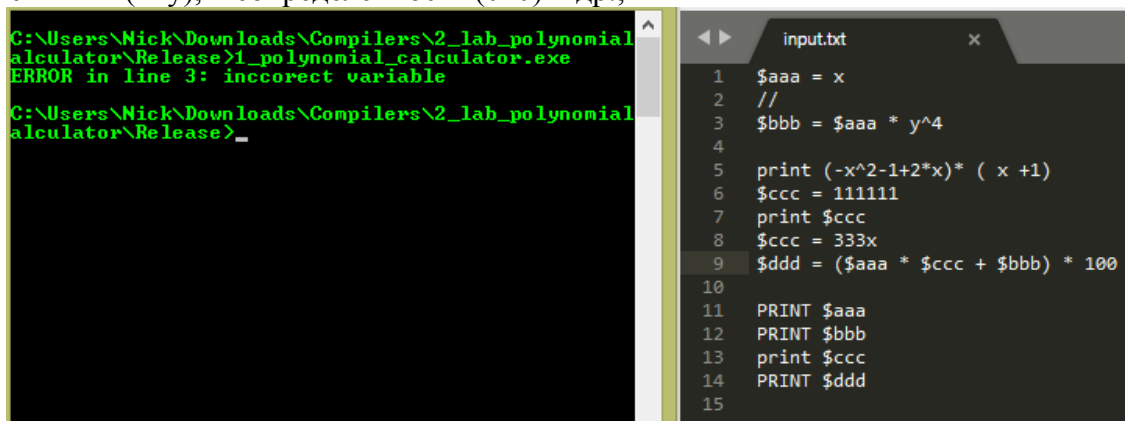
input.txt
1 $aaa = x
2 //asdfghjkl;
3 $bbb = $aaa * x^4
4
5 print (-x^2-1+2*x)* ( x +1)
6 $ccc = 111111
7
8 print $ccc
9 $ccc = 333x
10 $ddd = ($aaa * $ccc + $bbb) * 100
11
12 PRINT $aaa * 1234x^13
13 PRINT $bbb
14 print $ccc
15 PRINT $ddd
16
```

Рисунок 1 – Пример синтаксиса и корректной работы калькулятора

Калькулятор считывает входные данные из файла и обрабатывает их. Каждое действие пишется с новой строки. В случае неверного синтаксиса программа выводит ошибки.

Виды ошибок:

– семантические ошибки (рисунки 2-3): выполнение операций с различными переменными ($x*y$), неопределенность (0^0) и др.;

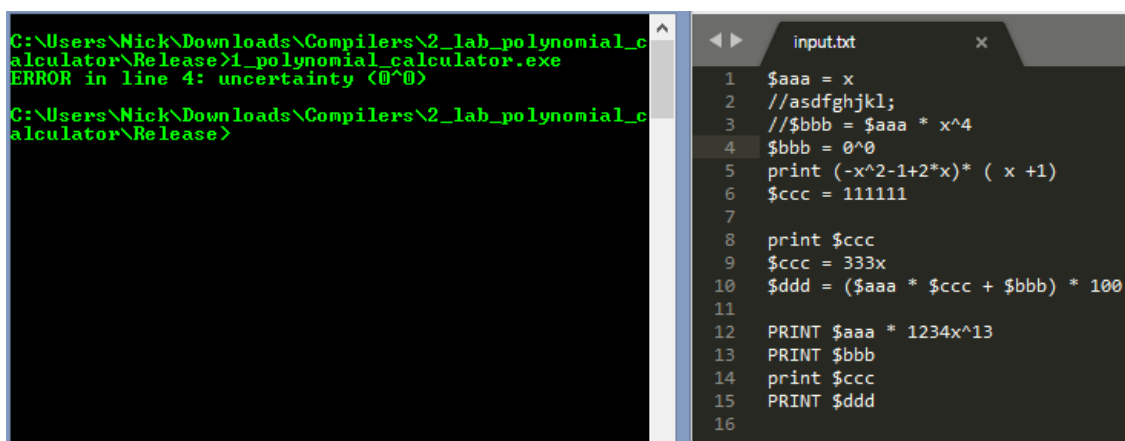


```
C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>i_polynomial_calculator.exe
ERROR in line 3: incorrect variable

C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>_
```

```
input.txt
1 $aaa = x
2 //
3 $bbb = $aaa * y^4
4
5 print (-x^2-1+2*x)* ( x +1)
6 $ccc = 111111
7 print $ccc
8 $ccc = 333x
9 $ddd = ($aaa * $ccc + $bbb) * 100
10
11 PRINT $aaa
12 PRINT $bbb
13 print $ccc
14 PRINT $ddd
15
```

Рисунок 2 – Пример семантической ошибки, операция с различными переменными



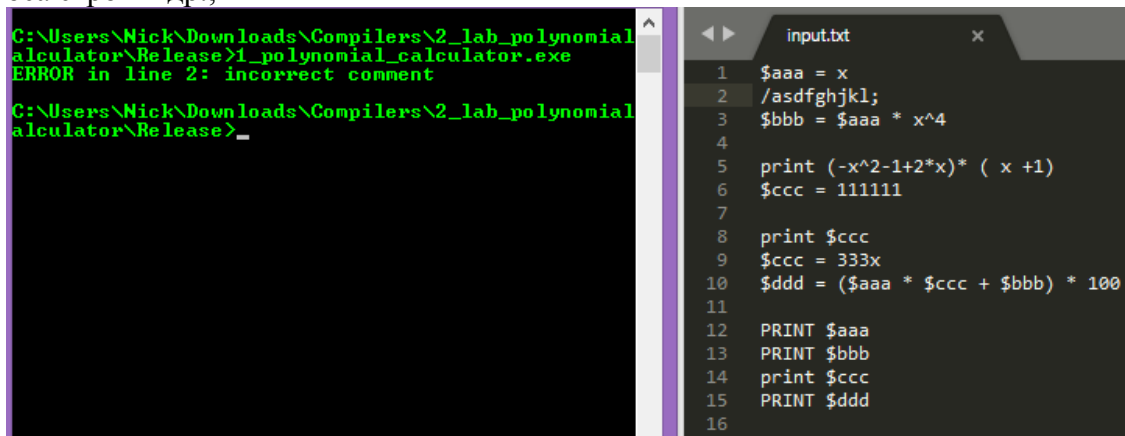
```
C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>i_polynomial_calculator.exe
ERROR in line 4: uncertainty <0^0>

C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>
```

```
input.txt
1 $aaa = x
2 //asdfghjkl;
3 //$bbb = $aaa * x^4
4 $bbb = 0^0
5 print (-x^2-1+2*x)* ( x +1)
6 $ccc = 111111
7
8 print $ccc
9 $ccc = 333x
10 $ddd = ($aaa * $ccc + $bbb) * 100
11
12 PRINT $aaa * 1234x^13
13 PRINT $bbb
14 print $ccc
15 PRINT $ddd
16
```

Рисунок 3 – Пример семантической ошибки, неопределенность

– синтаксические ошибки (рисунки 4-5): неправильные комментарии, отсутствие переноса строк и др.;



```
C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>i_polynomial_calculator.exe
ERROR in line 2: incorrect comment

C:\Users\Nick\Downloads\Compilers\2_lab_polynomial_calculator\Release>_
```

```
input.txt
1 $aaa = x
2 /asdfghjkl;
3 $bbb = $aaa * x^4
4
5 print (-x^2-1+2*x)* ( x +1)
6 $ccc = 111111
7
8 print $ccc
9 $ccc = 333x
10 $ddd = ($aaa * $ccc + $bbb) * 100
11
12 PRINT $aaa
13 PRINT $bbb
14 print $ccc
15 PRINT $ddd
16
```

Рисунок 4 – Пример синтаксической ошибки, неправильный комментарий

The terminal window on the left shows the execution of a polynomial calculator. It reports a 'parse error' on line 4. The input.txt file on the right contains the following code:

```

1 $aaa = x
2 //asdfghjkl;
3 $bbb = $aaa * x^4
4 print (-x^2-1+2*x)* ( x +1) $ccc = 111111
5
6 print $ccc
7 $ccc = 333x
8 $ddd = ($aaa * $ccc + $bbb) * 100
9
10 PRINT $aaa * 1234x^13
11 PRINT $bbb
12 print $ccc
13 PRINT $ddd
14

```

Рисунок 5 – Пример синтаксической ошибки, перенос строки

– лексические ошибки (рисунки 6-5): неинициализированные переменные отсутствие знака присваивания и др.;

The terminal window on the left shows the execution of the polynomial calculator. It reports an error: 'ERROR in line 10: not initialize variable bbb'. The input.txt file on the right contains the following code:

```

1 $aaa = x
2 //asdfghjkl;
3 //$bbb = $aaa * x^4
4
5 print (-x^2-1+2*x)* ( x +1)
6 $ccc = 111111
7
8 print $ccc
9 $ccc = 333x
10 $ddd = ($aaa * $ccc + $bbb) * 100
11
12 PRINT $aaa
13 PRINT $bbb
14 print $ccc
15 PRINT $ddd
16

```

Рисунок 6 – Пример лексической ошибки, неинициализированная переменная

При составлении грамматики были использованы токены и нетерминальные символы:

- variable – имя переменной типа полином;
- polynomial – структура полинома;
- monomial – структура одночлена;
- DIGIT – токен числа;
- LETTER – токен строки;
- PRINT – токен оператора вывода;
- '+', '-' – операция сложения, вычитания;
- '*' – операция умножения;
- UMINUS – операция унарного минуса;
- '^' – операция возведения в степень;
- '(', ')' – скобки;
- list – точка входа.

Правила грамматики:

- перед переменной типа полином ставится '\$', имя переменной строка, после объявления должен следовать знак '=' и полином (например, "\$abc = x");
- поддерживаются комментарии, которые начинаются с символов "//";
- имя переменных полинома строка;
- оператор PRINT и print равнозначны, и после них следует полином или переменная, которую нужно распечатать (например, "PRINT \$abc"/"print (x+1)^2");
- после каждой строки следует перенос строки.

3 ВЫВОДЫ

В результате работы изучены принципы составления грамматик на языке YACC (Bison) и создания лексического анализатора.

Разработанная языковая грамматика позволяет обрабатывать полиномы с несколькими переменными, запись полиномов приближена к записи в классической математике.

Калькулятор поддерживает базовые языковые конструкции.

Листинг 1:**polyc_yacc.y**

```

%{
    #include "polyc.h"
    #define alloca malloc
    int yylex (void);
    void yyerror(char const *s);
%}

%union // what types will be used further
{
    struct _polynomial poly;
    struct _monomial mono;
    int num;
    char *str;
}

%type <str> variable
%type <poly> polynomial
%type <mono> monomial

%token <num> DIGIT
%token <str> LETTER PRINT

%left '+' '-'
%left '*'
%right UMINUS
%right '^'

%start list
%% // follows grammatics
list :
    // empty
    |
    list
    '\n'
    |
    list
    '\r'
    |
    list
    begin
    ;
begin :
    variable
    '='
    polynomial
    '\n'
    {
        Insert_Variable_In_Global_List($1, $3);
    }
    |
    PRINT
    variable
    '\n'
    {
        printf("variable \"%s\" ", $2);
        struct _polynomial tmp = Search_Variable_In_Global_List($2);
        Print_Polynomial(&tmp);
    }
    |
    PRINT
    polynomial
    '\n'
    {
        printf("polynomial ");
        Print_Polynomial(&$2);
    }
    ;
polynomial :
    '('
    polynomial
    ')'
    {
        $$ = $2;
    }

```

```

| polynomial
| '+'
| polynomial
| {
|     $$ = Add_Polynomials($1, $3);
| }
| polynomial
| '-'
| polynomial
| {
|     $$ = Subtract_Polynomials($1, $3);
| }
| polynomial
| '*'
| polynomial
| {
|     $$ = Multiply_Polynomials($1, $3);
| }
| '.'
| polynomial
| %prec
| UMINUS
| {
|     $$ = Uminus_Polynomial(&$2);
| }
| polynomial
| '^'
| DIGIT
| {
|     if ($3 == 0)
|     {
|         if ($1.begin->item.coefficient == 0)
|         {
|             // Semantic error
|             Report_Bug("uncertainty (0^0)", "");
|         }
|         $$ = Initialize_Polynomial();
|         $$ = Add_Monomial_in_Polynomial($$, Set_Monomial(1, "", 0));
|     }
|     else if ($3 == 1)
|     {
|         $$ = $1;
|     }
|     else if ($3 >= 2)
|     {
|         for (int i = 0; i < $3 - 1; i++)
|         {
|             $$ = Multiply_Polynomials($$, $1);
|         }
|     }
| }
| monomial
| {
|     $$ = Initialize_Polynomial();
|     $$ = Add_Monomial_in_Polynomial($$, $1);
| }
| variable
| {
|     struct _polynomial tmp;
|     tmp = Initialize_Polynomial();
|     tmp = Add_Monomial_in_Polynomial(tmp, Set_Monomial(1, "", 0));
|     $$ = Search_Variable_In_Global_List($1);
|     $$ = Multiply_Polynomials($$, tmp);
| }
| ;
monomial:
DIGIT
| {
|     $$ = Set_Monomial($1, "", 0);
| }
| LETTER
| {
|     $$ = Set_Monomial(1, $1, 1);

```

```

        }
        |
        DIGIT
        LETTER
        {
            $$ = Set_Monomial($1, $2, 1);
        }
        |
        LETTER
        '^'
        DIGIT
        {
            $$ = Set_Monomial(1, $1, $3);
        }
        |
        DIGIT
        LETTER
        '^'
        DIGIT
        {
            $$ = Set_Monomial($1, $2, $4);
        }
    ;

variable :
    '$'
    LETTER
    {
        $$ = $2;
    }
;

%% // end's grammatics

```

polyc.h

```

#ifndef POLYC_H_
#define POLYC_H_
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
struct _monomial
{
    int coefficient; // коэффициент
    char * variable; // имя переменной
    int power; // степень
};
struct _polynomial
{
    struct _node * begin; // указатель на первый одночлен в полиноме
    int count; // число элементов в полиноме
};
struct _node
{
    struct _monomial item; // указатель на одночлен
    struct _node * next; // указатель на след. узел
    struct _node * prev; // указатель на пред. узел
};
struct _variable
{
    char * variable; // имя переменной
    struct _polynomial polynomial; // копия полинома
    struct _variable * next; // указатель на следующую переменную
    struct _variable * prev; // указатель на следующую переменную
};

void Report_Bug (const char *, const char *);
void Insert_Variable_In_Global_List (char *, struct _polynomial);
struct _polynomial Search_Variable_In_Global_List (char *);
struct _monomial Set_Monomial (int, char *, int);
struct _polynomial Initialize_Polynomial (void);
struct _polynomial Add_Monomial_in_Polynomial (struct _polynomial, struct _monomial);
struct _node * Remove_Node (struct _polynomial *, struct _node *);
struct _polynomial Remove_Similar_Summands (struct _polynomial);
struct _polynomial Add_Polynomial (struct _polynomial, struct _polynomial);
struct _polynomial Subtract_Polynomial (struct _polynomial, struct _polynomial);
struct _polynomial Multiply_Polynomial (struct _polynomial, struct _polynomial);
struct _polynomial Uminus_Polynomial (struct _polynomial *);
void Print_Monomial (struct _monomial *);
void Print_Polynomial (struct _polynomial *);
#endif

```


source.c

```
#define _CRT_SECURE_NO_WARNINGS // fopen
#define _CRT_NONSTDC_NO_DEPRECATED // strdup
#include "polyc.h"
int g_num_lines = 0;
extern struct _variable * g_list_variables = NULL;
void Insert_Variable_In_Global_List(char *letter, struct _polynomial
polynomial)
{
    struct _variable *tmp = g_list_variables,
    *tmp_variable = (struct _variable
*)malloc(sizeof(struct _variable));
    tmp_variable->variable = letter;
    tmp_variable->polynomial = polynomial;
    tmp_variable->next = NULL;
    tmp_variable->prev = NULL;
    // Если ни разу не объявляли переменные
    if (g_list_variables == NULL)
    {
        g_list_variables = tmp_variable;
        return;
    }
    while (tmp->next != NULL)
    {
        // Если такая переменная уже есть. то
        перезаписываем ее
        if (!strcmp(tmp->variable, letter))
        {
            tmp->polynomial = polynomial;
            return;
        }
        tmp = tmp->next;
    }
    if (!strcmp(tmp->variable, letter))
    {
        tmp->polynomial = polynomial;
        return;
    }
    // Если же нет, то вставляем в конец
    tmp_variable->prev = tmp;
    tmp->next = tmp_variable;
}
struct _polynomial Search_Variable_In_Global_List(char *variable)
{
    struct _variable * tmp = g_list_variables;
    while (tmp != NULL)
    {
        if (!strcmp(tmp->variable, variable))
        {
            return tmp->polynomial;
        }
        tmp = tmp->next;
    }
    Report_Bug("not initialize variable ", variable); //
Lexical error
}
struct _monomial Set_Monomial(int coefficient, char *letter, int
power)
{
    struct _monomial result;
    result.coefficient = coefficient;
    result.variable = letter;
    result.power = power;
    return result;
}
struct _polynomial Initialize_Polynomial(void)
{
    struct _polynomial result;
    result.begin = (struct _node *) malloc(sizeof(struct
_node));
    result.begin->prev = NULL;
    result.begin->next = NULL;
    result.count = 0;
    return result;
}
struct _polynomial Add_Monomial_in_Polynomial(struct _polynomial
polynomial, struct _monomial monomial)
{
    struct _polynomial result;
    struct _node * tmp;

    result = polynomial;
    tmp = result.begin;
    // Если в многочлене нет одночленов
```

```
if (result.count == 0)
{
    tmp->item = monomial;
    result.count++;
    return result;
}
// Доходим до конца и добавляем
while (tmp->next != NULL)
{
    tmp = tmp->next;
}
tmp->next = (struct _node *) malloc(sizeof(struct _node));
tmp->next->prev = tmp;
tmp->next->next = NULL;
tmp = tmp->next;
tmp->item = monomial;
result.count++;
return result;
}
struct _node * Remove_Node(struct _polynomial *polynomial, struct
_node *node)
{
    struct _node * result = node;
    // Если удаляемый узел начало полинома
    if (polynomial->begin == node)
    {
        // Если последний узел в полиноме, возврат
        ноль
        if (node->next == NULL)
        {
            free(node);
            // error!!!
            return NULL;
        }
        node->next->prev = NULL;
        result = node->next;
        free(node);
        polynomial->begin = result;
        return result;
    }
    // Если дошли до конца полинома, вернуть
    предыдущий, т.к. все равно выполнится "tmp2 = tmp2->next;"
    if (node->next == NULL)
    {
        node->prev->next = NULL;
        result = node->prev;
        free(node);
        return result;
    }
    node->next->prev = node->prev;
    node->prev->next = node->next;
    // Удаление узла, вернуть предыдущий, т.к. в
    притидении подобных все равно перейдет на следующий
    result = node->prev;
    free(node);
    return result;
}
struct _polynomial Remove_Similar_Summands(struct _polynomial
polynomial)
{
    struct _polynomial result = Initialize_Polynomial();
    struct _monomial tmp_monom;
    struct _node * tmp1 = polynomial.begin,
    *tmp2;
    while (tmp1 != NULL)
    {
        // Ищем все подобные для этого элемента
        tmp_monom = tmp1->item;
        tmp2 = polynomial.begin;
        while (tmp2 != NULL)
        {
            if (!strcmp(tmp1->item.variable,
tmp2->item.variable) &&
tmp2->item.power ==
tmp1->item.power ==
tmp1 != tmp2)
            {
                tmp_monom.coefficient +=
tmp2->item.coefficient;
                // Удаляем узел, т.к.
добавили значение коэффициента
tmp2 =
Remove_Node(&polynomial, tmp2);
```

```

    }
    tmp2 = tmp2->next;
}
// Удаляем узел, т.к. результирующий
одночлен будет занесен в результирующий многочлен
tmp1 = Remove_Node(&polynomial, tmp1);
if (tmp_monom.coefficient == 0)
{
    continue;
}
result = Add_Monomial_in_Polynomial(result,
tmp_monom);
}
return result;
}
struct _polynomial Add_Polynomials(struct _polynomial
polynomial_one, struct _polynomial polynomial_two)
{
    struct _polynomial result = polynomial_one;
    struct _node * tmp = polynomial_two.begin;
    result = Add_Monomial_in_Polynomial(result, tmp->item);
    // Сначала добавить одночлен из "головы", потом все
    остальные
    while (tmp->next != NULL)
    {
        tmp = tmp->next;
        result = Add_Monomial_in_Polynomial(result,
tmp->item);
    }
    return result = Remove_Similar_Summands(result);
}
struct _polynomial Uminus_Polynomial(struct _polynomial
*polynomial)
{
    struct _node * tmp = polynomial->begin;
    while (tmp != NULL)
    {
        tmp->item.coefficient *= (-1);
        tmp = tmp->next;
    }
    return *polynomial;
}
struct _polynomial Subtract_Polynomials(struct _polynomial
polynomial_one, struct _polynomial polynomial_two)
{
    struct _polynomial result =
Add_Polynomials(polynomial_one,
Uminus_Polynomial(&polynomial_two));
    return result;
}
struct _polynomial Multiply_Polynomials(struct _polynomial
polynomial_one, struct _polynomial polynomial_two)
{
    struct _polynomial result = Initialize_Polynomial();
    struct _monomial tmp_monom;
    struct _node * tmp1 = polynomial_one.begin,
*tmp2;
    while (tmp1 != NULL)
    {
        tmp2 = polynomial_two.begin;
        while (tmp2 != NULL)
        {
            // Раскрываем "фонтанчиком"
            tmp_monom = tmp1->item;

            tmp_monom.coefficient *= tmp2-
>item.coefficient;
            if (!strcmp(tmp_monom.variable,
tmp2->item.variable))
            {
                tmp_monom.power +=
tmp2->item.power;
            }
            else
            {
                if (!
strcmp(tmp_monom.variable, ""))
                {
                    tmp_monom.variable =
strdup(tmp2->item.variable);
                    tmp_monom.power +=
tmp2->item.power;
                }
            }
        }
    }
}

```

```

else if (!strcmp(tmp2-
>item.variable, ""))
{
    ;
}
else
{
    // Если
умножаем разные переменные, и они не совпадают
// Semantic
error
g_num_lines--;

Report_Bug("incorrect variable", "");
}
}
result =
Add_Monomial_in_Polynomial(result, tmp_monom);
tmp2 = tmp2->next;
}
tmp1 = tmp1->next;
}
return result = Remove_Similar_Summands(result);
}
void Print_Monomial(struct _monomial *monomial)
{
    if (!strcmp(monomial->variable, ""))
        printf("%d", monomial->coefficient);
    else
        if (monomial->coefficient == 1)
            if (monomial->power == 1)
                printf("%s", monomial-
>variable);
            else
                printf("%s^%d", monomial-
>variable, monomial->power);
            else
                if (monomial->power == 1)
                    printf("%d%s", monomial-
>coefficient, monomial->variable);
                else
                    if (monomial->coefficient
== 1)
                        printf("%s^%d",
monomial->variable, monomial->power);
                    else
                        printf("%d%s^
%d", monomial->coefficient, monomial->variable, monomial->power);
}
void Print_Polynomial(struct _polynomial *polynomial)
{
    struct _node *tmp = polynomial->begin;
    printf("= ");
    Print_Monomial(&(tmp->item));
    tmp = tmp->next;
    while (tmp != NULL)
    {
        if (tmp->item.coefficient < 0)
        {
            ;
        }
        else
        {
            printf("+");
        }
        Print_Monomial(&(tmp->item));
        tmp = tmp->next;
    }
    printf("\n");
}
#include "y_tab.h"
FILE *inputStream;
#define yyin inputStream
#define tmp_scanf(f_, ...) fscanf(yyin, (f_), __VA_ARGS__)
void yyerror(char const *s)
{
    printf("ERROR in line %d: '%s' on token ...ln",
g_num_lines + 1, s); // Syntax error
}
void Report_Bug(const char *s, const char *s2)
{
    printf("ERROR in line %d: %s%s\n", g_num_lines + 1, s,
s2);
    exit(-1);
}

```

```

}
int yylex(void)
{
    int get;
    while ((get = fgetc(yyin)) == ' ' || get == '\t')
    {
        ;
    }
    while (get == '/')
    {
        get = fgetc(yyin);
        if (get == '/')
        {
            while ((get = fgetc(yyin)) != '\n')
            {
                ;
            }
            g_num_lines++;
        }
        else
        {
            Report_Bug("incorrect comment", "");
        }
        get = fgetc(yyin);
    }
    if (get == EOF)
    {
        return 0;
    }
    else if (isdigit(get))
    {
        yyval.num = 0;
        ungetc(get, yyin);
        tmp_scanf("%d", &yyval.num);
        return DIGIT;
    }
    else if (isalpha(get))
    {
        static char *symbol = 0;
        int length = 0;
        int i = 0;
        // Initially make the buffer long enough
        // for a 40-character symbol name.
        if (length == 0)
        {
            length = 40;
            symbol = (char *)malloc(length + 1);
        }
        do
        {
            // If buffer is full, make it bigger.
            if (i == length)
            {
                length *= 2;
                symbol = (char *)realloc(symbol, length + 1);
            }
            // Add this character to the buffer.
            symbol[i++] = get;
            // Get another character.
            get = fgetc(yyin);
        } while (isalnum(get));
        ungetc(get, yyin);
        symbol[i] = '\0';
        if (!strcmp("PRINT", symbol) || !strcmp("print",
symbol))

```

```

{
    return PRINT;
}
yyval.str = symbol;
return LETTER;
}
else
{
    switch (get)
    {
        case '+':
        {
            return get;
        }
        case '-':
        {
            return get;
        }
        case '^':
        {
            return get;
        }
        case '*':
        {
            return get;
        }
        case '(':
        {
            return get;
        }
        case ')':
        {
            return get;
        }
        case '=':
        {
            return get;
        }
        case '$':
        {
            return get;
        }
        case '\n':
        {
            g_num_lines++;
            return get;
        }
    }
    return get;
}
int main(void)
{
    inputStream = fopen("input.txt", "r");
    if (inputStream == NULL)
    {
        printf("Not found file. Creadte file \"input.txt\"
and input in file polynom.\n");
        exit(-1);
    }
    yyparse();
    fclose(inputStream);
}

```