Логические интерпретатор $\mathrm{Log}_{3.0}$

Кирилл Корнеев

29 декабря 2024 г.

Оглавление

0.1 Введение

0.1.1 Цели проекта

Основной целью проекта является написание программы для решения логических выражений и систем, чтобы улучшить навыки в создании алгоритмов и повысить квалификацию в программировании интерпретаторов в целом.

0.1.2 Задачи

1. Разбор принципов работы интерпретаторов в целом.
2. Создание алгоритма для работы логического интерпретатора.
    - Разработка алгоритма для подсчета количества решений в логических выражениях и системах
    - Написание программы на языке программирования СИ с использованием стандартных библиотек: stdlib.h, stdio.h, string.h.
3. Повышение навыков в программировании.

0.2 Алгоритм работы интерпретатора

0.2.1 Алгорит перевода выражения стандартного вида в обратную польскую нотацию

Два плюс три умножить на три?

Думаю многие, в том числе и я, мучились в начальной школе, пытаясь разобраться с приоритетом: сначала складывать 2 и 3 или же сначала умножать 3 на 3? Позже, начиная писать программу, как и любой начинающий программист, я мучался с приоритетом операций и скобками, когда писал свой первый интерпретатор.

В итоге, оказалось, что мучался я зря, ведь существует прекрасный механизм, именуемый обратной польской записью или же обратной польской нотацией, который сводит эти мучения на нет как для программиста, так и, тем более, для компьютера.

В алгебре бы привыкли видеть выражения типа $x + y$, где знак операции стоит между операндами. Такая запись называется инфиксной. Запись же типа $xy+$, где знак стоит после операндов называется постфиксной, или обратной польской записью. Обратная польская нотация

имеет ряд преимуществ перед инфиксной записью. Во-первых, инфиксная форма записи сводит на нет знания об приоритетах операций, все, что нам нужно, это идти слева направо и выполнять все по очереди. Во-вторых, любая формула может быть выражена без скобок.

Обратная польская запись/нотация, далее ОПЗ

Общий принцип работы алгоритма

Задача: на вход подается выражение в инфиксной форме, состоящее из односимвольных операндов и логических операций. Следует преобразовать эту запись в ОПЗ.

Пусть наша формула состоит из двухоперандных операторов: $+$ – операция дизъюнкции, $*$ – операция конъюнкции, $-$ – импликация, $=$ – эквивалентность; одного одноопередного оператора (! – логическое отрицание), а также левой и провой скобок.

Получив переменные типа char, мы начинаем их преобразовывать. Рассматриваем поочередно каждый символ:

1. Если это константа, то просто помещаем ее в выводную строку
2. Если символ - знак операции (+, * , -, =, !, (значение смотреть выше), то проверяем приоритет. Операция НЕ имеет наивысший приоритет (допустим, он равен 5). Операция И имеет меньший приоритет (равеный 4). Средний приоритет имеет логическое ИЛИ (равеный 3). Ниже располагаются операции импликации и эквиваленции (приоритет равный 2). Самый маленький приоритет у открывающей скобки (1). Получив один из этих символов, мы должны проверить стэк:
    - Если стэк все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стэк.
    - Если символ, находящийся на вершине стэка имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стэка в выходную строку до тех пор, пока выполняется это условие.
3. Если текущий символ - открывающая скобка, то помещаем ее в стэк.
4. Если текущий символ - закрывающая скобка, то извлекаем символы из стэка в выходную строку до тех пор, пока не встретим в

стэке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

5. Если вся входная строка разобрана, а в стэке еще остаются знаки операций, извлекаем их из стэка в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения:
Дано выражение:
a + (b - c) * d, где a, b, c - константы; + – логическое ИЛИ, * – логическое И, - – операция импликации.

Символ	Действие	Стэк
a	Константа. Помещаем в исходную строку.	Ничего.
+	Знак операции. Так как стэк пуст, то помещаем его туда.	+
(	Открывающая скобка. Помещаем в стэк.	+(
b	Константа. Помещаем в исходное строку.	+(
-	Знак операции, имеет приоритет 2. Проверяем стэк, на вершине находится знак скобки, ее приоритет 1. Следовательно, помещаем на вершину стэка.	+(-
c	Константа. Помещаем в исходное строку.	+(-
)	Извлекаем из стэка в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки	+
*	знак операции, который имеет приоритет 4. Проверяем стэк: на вершине находится символ '+', приоритет которого равен 3, т.е. меньший, чем приоритет текущего символа '*'. Следовательно, мы должны просто поместить текущий символ '*' в стэк.	+*
d	Константа. Помещаем в исходную строку.	+*

Теперь вся входная строка разобрана, но в стэке еще остаются знаки операций, которые мы должны просто извлечь в выходную строку.
Исходная строка в ОПЗ: **a b c - d * +**

Реализация ОПЗ в моей программе

В данном разделе представлен листинг моей подпрограммы для реализации алгоритма ОПЗ.

```
int priority (char a) ///Функция выявления приоритета (готова)
{
    int n=0;
    switch (a)
    {
    case '!': ///Если это ! , то наивысшее значение приоритета (5) (НЕ)
        { n = 5; break; }
    case '^': ///Если это ^, то высшее значение приоритета (4) (конъюнкция)
        { n = 4; break; }
    case '+': ///Если это + , то среднее значение приоритета (3) (дизъюнкция)
        { n = 3; break; }
    case '-': ///Если это -/=, то низкое значение приоритета (2) (импликация,
    //эквивалентность)
    case '=':
        { n = 2; break; }
    case '(': ///Если это (, то самое низкое значение
        { n = 1; break; } ///приоритета (1)
    }
    return n;
}

void operation_plus (int *top, int x) ///Функция для заноса операций в стэк
{
    *top=*top+1; ///Собственно когда приходим с переменной, то повышаем значение top на 1
    st[*top]=x;  ///И присваиваем этому значению (макс.) значение новой операции
}

char operation_minus (int *top) ///Функция для выноса операций из стэк
{
    char x; ///Выводной символ
    x=st[*top]; ///Он равен последнему значению в стэк
    *top=*top-1; ///Теперь опускаемся ниже
    return x; ///Передаем операцию
}

char opz(char* a, char* out, int top, int k, int p) ///Функция преобразования в ОПЗ
{
    int i=0, j=0, r, v=0;
    int h;
    unsigned char sym;
    int c[256]={0};
    char digits[256];
    while ( a[k]!='\0' ) ///Пока строка не закончилась
    {
        if ( a[k]==')' ) ///Если встречаем ), то
        {
            while( st[top]!='(' ) ///Все значения в стэке до ( выполняем
            {
                out[p++]=operation_minus(&top);
```

```
        }
        operation_minus(&top); ///Чистим эту скобочку
    }

    if (( a[k] >='a' && a[k]<='z') || ( a[k] >='A' && a[k]<='Z')) ///Если значение
    //символа - константа (буква), то
    {
        out[p++]=a[k]; ///Заносим в выводную строку сразу
    }

    if ( a[k] == '(' ) ///Если значение символа это (, то
    {
        operation_plus(&top, '('); ///Передаем в ф-ю количество операций в стэке и
        //код (
    }

    if ( a[k]=='+' || a[k]=='-' || a[k]=='^' || a[k]=='=' || a[k]=='!' ) ///Если
    //значение символа это знак, то
    {
        if ( top==0 ) ///Если в стэке ничего нет, то сразу туда и заносим
        //наше значение
        {
            operation_plus(&top, a[k]);
        }

        else ///Если же нет, то проверяем приоритет
        {
            if ( priority(st[top]) < priority(a[k]) ) ///Если приоритет того
            //что есть, больше, того, что сейчас, то
            {
                operation_plus(&top, a[k]); ///Ставим то, что есть сейчас,
                //на топ стэка
            }
            else ///Если приоритет того, что есть, меньше того, что сейчас, то
            {
                while ( priority(st[top]) >= priority(a[k]) ) ///Пока приоритет того,
                //что в стэке, больше (равен) того, что сейчас
                {
                    out[p++]=operation_minus(&top); ///Присваиваем значение операции
                    //к последнему в выводной строке.
                }
                operation_plus(&top, a[k]);
            }
        }
    }

    k++; ///Переходим на другой символ
}
```

```
    while(top!=0 || top > 0)
    {
        out[p++]=operation_minus(&top);
    }
    out[p++] = 0;
    return *out;
}
```

0.2.2 Алгоритм счета логических операций

В данном параграфе описан алгоритм счета логических операций: общий принцип работы и моя реализация.

Общий принцип работы алгоритма

Задача: на вход поступает логическая запись, состоящая из цифр (единица и ноль) и операций, записанная в форме обратной польской нотации. Требуется подсчитать конечный ответ, это будет либо единица - истина, либо ноль - ложь. Так как мой интерпретатор решает систему логических уравнений, то на вход функции может поступить не только одно, но и более уравнений. Если в нашей системе два уравнения или больше, то программа записывает их в одну строчку, ставя между разными уравнениями знак ";". Например:
Исходная система:

$11 + 1 =$

$11 - 1 =$

Она будет преобразована в строку:

$11 + 1 =; 11 - 1 =$

В этом случае интерпретатор считает значения до знака ";". Каждый раз, проводя операции, мы записываем ответ на одну из позиций и убираем из строки ненужные символы. Получив конечные ответы, мы их перемножаем, если на выходе единица, то вся система истина, если ноль, то, наоборот - ложна.

Ниже приведена трассировочная таблица алгоритма на примере строки выше:

Текущая строка	Действие	Выход
11+1=;11-1=	1+1=1	11=;11-1=
11=;11-1=	1=1=1	1;11-1=
1;11-1=	1-1=1	1;11=
1;11=	1=1=1	1;1
1;1	1*1=1	1

Как мы видим, ответ - единица, то есть истина, значит, вся система истина (что верно).

Реализация счета в моей программе

В данном разделе представлен листинг моей подпрограммы для реализации счета логических операций.

```
      char digit (char* out) ///Функция счета (готова)
{
    int t=0, j=0, k=0, v=0, l=0, v1=0; ///Разные счетчики и т.д.
    int x, y, rez=0;
    char symbol;
    v1=strlen(out);
    t=0;
    while (out[t]!='\0') ///Пока не закончена строка идем
    {
        while (out[t]!=';')
        {
        ///t++;
        if ((out[t] != '+') || (out[t] != '=') || (out[t] != '-') || (out[t] != '^')
        || (out[t] != '!')) ///Если константа - идем дальше
        {
            t++;
        }
        if ((out[t] == '+') || (out[t] == '=') || (out[t] == '-') || (out[t] == '^')
        || (out[t] == '!')) ///Если нет, идем считать
        {
            j=t-1; ///Берем две константы рядом с операцией
            k=t-2;
            if (out[t] == '+') ///Если это +
            {
                x=out[k]-'0'; ///Переводим два операнда в инт
                y=out[j]-'0';
                if ((x == 1) && (y == 1)) ///Считаем результат и переводим символ
                {
                    rez = 1;
                }
                else
                {
                    rez = x+y;
                }
                symbol=rez+'0';
                out[k]=symbol; ///Ставим символ в строку
                out[j]=out[t+1];
                j=j+1;
                for (long i = j; i < v1; ++i) ///Убираем ненужные два символа
                {
```

```
      out[i] = out[i + 1];
    }
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    t=0;

}
if (out[t] == '=') ///Если это =
{
    x=out[k]-'0'; ///Переводим два операнда в инт
    y=out[j]-'0';
    if (((x == 1) && (y == 1)) || ((x == 0) && (y == 0))) ///Считаем результат
    ///и переводим символ
    {
        rez = 1;
    }
    else
    {
        rez = 0;
    }
    symbol=rez+'0';
    out[k]=symbol; ///Ставим символ в строку
    for (long i = j; i < v1; ++i) ///Убираем ненужные два символа
    {
      out[i] = out[i + 1];
    }
    --v1;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    t=0;
}
if (out[t] == '-') ///Если это -
{
    x=out[k]-'0'; ///Переводим два операнда в инт
    y=out[j]-'0';
    if ((x == 1) && (y == 0)) ///Считаем результат и переводим символ
    {
        rez = 0;
    }
    else
    {
        rez = 1;
    }
```

```
    symbol=rez+'0';
    out[k]=symbol; ///Ставим символ в строку
    for (long i = j; i < v1; ++i) ///Убираем ненужные два символа
    {
      out[i] = out[i + 1];
    }
    --v1;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    t=0;
}
if (out[t] == '^') ///Если это ^
{
    x=out[k]-'0'; ///Переводим два операнда в инт
    y=out[j]-'0';
    rez = x * y; ///Считаем результат и переводим символ
    symbol=rez+'0';
    out[k]=symbol; ///Ставим символ в строку
    for (long i = j; i < v1; ++i) ///Убираем ненужные два символа
    {
      out[i] = out[i + 1];
    }
    --v1;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];

    }
    --v1;
    t=0;
}
if (out[t] == '!') ///Если это !
{
    y=out[j]-'0'; ///Переводим операнд в инт
    if (y == 1) ///Считаем результат и переводим символ
    {
        rez = 0;
    }
    else
    {
        rez = 1;
    }
    symbol=rez+'0'; ///Ставим символ в строку
    out[j]=symbol;
    for (long i = j+1; i < v1; ++i) ///Убираем ненужный символа
    {
```

```
                out[i] = out[i + 1];
            }

            --v1;
            t=0;
        }
    }

    }
    t++;
}
v1=1;
while (out[v]!='\0')
{
    if (out[v] >= '0' && out[v] <= '1')
    {
        l=out[v]-'0';
        v1=v1*l;
    }
    v++;
}
symbol=v1+'0';
return symbol;
}
```

0.2.3 Алгоритм составления таблицы истинности

В этом параграфе описан алгоритм составления таблицы истинности.

Канонично таблицы истинности имееют вид (где a и b - это операнды, a + - это логическая операция "И"):

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

Можно заметить, что количество строк в таблице - число, равное квадрату количества переменных, а каждая строка - порядковый номер от 0 до этого квадрата. На основе этих данных, составим алгоритм, описанный в следующем пункте.

0.2.4 Общий принцип работы алгоритма

В общем виде алгоритм выглядит так:

На вход мы получаем строку, состоящую из последовательно идущих

операндов и операций, количество переменных заранее известно. Прежде всего, мы возводим в квадрат количество переменных, то есть узнаем количество наших "строк". Далее, поочередно идем от 0 до квадрата количества переменных, попутно останавливаясь на каждом числе. При остановке мы переводим порядковую цифру в бинарную систему, получаяя при этом "ячейки"нашей таблицы, и присваиваем эти значения числам. Если выход подпрограммы будет 1, то увеличиваем общий овет а единицу и печатаем ответ, иначе все остается неизменным.

Реализация таблицы истинности в моей программе

В данном разделе представлен листинг моей подпрограммы для реализации алгоритма таблицы истинности.

```
void f(char* out, char y) ///Функция таблицы истинности(готова)
{
    int i=0, size, j=0, f=0, t=1; ///Счетчики
    int a=0, h=0, g=0, v=0, c=0;
    char out1[200];///Буффер массива
    char symbol;///Символ
    output = fopen("data_out.txt", "w");
    if (output == NULL)
    {
        printf("Eror [cant open output file");
    }
    size = 1 << kol;///Узнаем количество строк в таблице истинности
    v=strlen(out);
    for(a = 0; a < size; a++)///Проходим каждую строку в таблице
    {
        j=0;
        for (f = 0; f < v; f++)///Кидаем основную строку в буффер
            out1[f] = out[f];
        out1[f]='\0';
        for (i = kol - 1; i >= 0; i--)///Проходим каждый разряд,
        ///исходя из количества переменных, для перебора возможных комбинаций
        {
        for(c = 0; c < strlen(out); c++)
            {
                if (((out1[j] >= 'a' && out1[j] <= 'z') || (out1[c] >= 'a' && out1[c] <= 'z'))
                {
                if ((out1[c] >= 'a' && out1[c] <= 'z') || (out1[c] >= 'A' && out1[c] <= 'Z'))
                        j=c;
                    h=j+1;
                    while (out1[h]!='\0')///В этом цикле ищем похожие переменные
                    {
                        if (out1[h] == out1[j])///Находим - ставим на их значение константу
                        {
```

```
                            g = (a >> i) & 1;
                            symbol = g + '0';
                            out1[h] = symbol;
                            h++;
                        }
                        h++;
                    }
                    g = (a >> i) & 1;///Ставим константу в искомую переменную
                    symbol = g + '0';
                    out1[j] = symbol;
                    j++;
                    break;
                }
            }
        }
        if (digit(out1) == '1')///Если выражение истинно
        {
            counter++;///То повышаем значение счетчика
            fprintf(output, "%d", t);
            fprintf(output, ". ");
            for (i = kol - 1; i >=0; i--)
            {
                fprintf(output, "%d ", ((a>>i)&1) );///И печатаем этот ответ в файл
            }
            fprintf(output, "\n");
            t++;
        }
    }
}
```

0.2.5 Алгоритм проверки синтаксиса

Главной задачей данного алгоритма является проверка базовых синтаксических ошибок. Например: 1) два идущих подряд операнда или две идущех подряд операции, 2) забытая скобка, 3) сразу же закрытая скобка.

Реализация проверки синтаксиса в моей программе

В данном разделе представлен листинг моей подпрограммы для реализации алгоритма проверки синтаксиса.

```
 int check(char* a1) ///Функция проверки синтаксиса (готова)
{
    int i=0, j=0;
    for (i=0; i<strlen(a1); i++)///Пока строка не закончится, проверяем
    {
```

```
if (a1[i]  == '(')///Если мы нашли (, то
{
///
    j=i+1;
    if (a1[j] == ')')///Если за ней сразу идет ), то ошибка
    {
        printf("Syntax Eror [there is a ) right after the (]!");
        return 0;
    }
    ///
    else
    {
        while (a1[j]!='\0')
        {
            if (a1[j] == ')')///Если мы находим ), то все правильно, иначе - ошибка
                goto metka;
            j++;
        }
        printf("Syntax Eror [there is no a ) after the (]!");
        return 0;
        ///
        metka:;
        j++;
        ///printf(" ");
    }
}
j=i+1;
if ((a1[i]>='a' && a1[i]<='z') || (a1[i]>='A' && a1[i]<='Z'))///Если два операнда
///идут подряд, то ошибка
///
{
    if ((a1[j]>='a' && a1[j]<='z') || (a1[j]>='A' && a1[j]<='Z'))
    {
        printf("Syntax Eror[there is a letter right after the another one]!");
        return 0;
    }
}
if (a1[i]>='0' && a1[i]<='9')///Если две константы идут подряд, то ошибка
{
    if (a1[j]>='0' && a1[j]<='9')
    {
        printf("Syntax Eror[there is a digit right after the another one]!");
        return 0;
    }
}
if (a1[i]=='^' || a1[i]=='+' || a1[i]=='-' || a1[i]=='=' || a1[i]=='!')///Если две
///операции идут подряд , то ошибка (доделать вариант ошибки -!x1)
{
    if (a1[j]=='^' || a1[j]=='+' || a1[j]=='-' || a1[j]=='=')
```

```
            {
                printf("Syntax Eror[there is a operation right after the another one]!");
                return 0;
            }
        }
    }
}
```

0.2.6 Служебные подпрограммы

В данном разделе представлены подпрограммы, которые, на мой взгляд, не нуждаются в комментариях, они выполняют сугубо служебную функцию.

```
 char convert(char* a1) ///Функция перевода переменных (готова)
{
    int i=0, j=0, h=0, k=0, v1=0;
    ///v1=strlen(a1);
    while (a1[i]!='\0') ///Тут заменяем двухсимвольные переменные на односимвольные
    {
        if (a1[i] == 'x')
        {
            j=i+1;
            k=i+2;
            if (a1[j] == '1')
                a1[i] = 'a';
            if (a1[j] == '2')
                a1[i] = 'b';
            if (a1[j] == '3')
                a1[i] = 'c';
            if (a1[j] == '4')
                a1[i] = 'd';
            if (a1[j] == '5')
                a1[i] = 'e';
            if (a1[j] == '6')
                a1[i] = 'f';
            if (a1[j] == '7')
                a1[i] = 'g';
            if (a1[j] == '8')
                a1[i] = 'h';
            if (a1[j] == '9')
                a1[i] = 'i';
            if (a1[j] == '1' && a1[k] == '0')
                a1[i] = 'B';
            if (a1[j] == '1' && a1[k] == '1')
                a1[i] = 'C';
            if (a1[j] == '1' && a1[k] == '2')
                a1[i] = 'D';
```

```
    if (a1[j] == '1' && a1[k] == '3')
        a1[i] = 'K';
    if ((a1[j] >= '0' && a1[j] <= '9') && (a1[k] >= '0' && a1[k] <= '9'))
    {
        for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
        {
          a1[h] = a1[h + 1];
        }
        for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
        {
          a1[h] = a1[h + 1];
        }
    }
    else
    {
        for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
        {
          a1[h] = a1[h + 1];
        }
    }
}
if (a1[i] == 'y')
{
    j=i+1;
    if (a1[j] == '1')
        a1[i] = 'j';
    if (a1[j] == '2')
        a1[i] = 'k';
    if (a1[j] == '3')
        a1[i] = 'l';
    if (a1[j] == '4')
        a1[i] = 'm';
    if (a1[j] == '5')
        a1[i] = 'n';
    if (a1[j] == '6')
        a1[i] = 'o';
    if (a1[j] == '7')
        a1[i] = 'p';
    if (a1[j] == '8')
        a1[i] = 'q';
    if (a1[j] == '9')
        a1[i] = 'r';
    if (a1[j] == '1' && a1[k] == '0')
        a1[i] = 'E';
    if (a1[j] == '1' && a1[k] == '1')
        a1[i] = 'F';
    if (a1[j] == '1' && a1[k] == '2')
        a1[i] = 'G';
    if (a1[j] == '1' && a1[k] == '3')
```

```
            a1[i] = 'L';
        if ((a1[j] >= '0' && a1[j] <= '9') && (a1[k] >= '0' && a1[k] <= '9'))
        {
            for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
            {
              a1[h] = a1[h + 1];
            }
            for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
            {
              a1[h] = a1[h + 1];
            }
        }
        else
        {
            for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
            {
              a1[h] = a1[h + 1];
            }
        }
    }
    if (a1[i] == 'z')
    {
        j=i+1;
        if (a1[j] == '1')
            a1[i] = 's';
        if (a1[j] == '2')
            a1[i] = 't';
        if (a1[j] == '3')
            a1[i] = 'u';
        if (a1[j] == '4')
            a1[i] = 'v';
        if (a1[j] == '5')
            a1[i] = 'w';
        if (a1[j] == '6')
            a1[i] = 'x';
        if (a1[j] == '7')
            a1[i] = 'y';
        if (a1[j] == '8')
            a1[i] = 'z';
        if (a1[j] == '9')
            a1[i] = 'A';
        if (a1[j] == '1' && a1[k] == '0')
            a1[i] = 'H';
        if (a1[j] == '1' && a1[k] == '1')
            a1[i] = 'I';
        if (a1[j] == '1' && a1[k] == '2')
            a1[i] = 'J';
        if (a1[j] == '1' && a1[k] == '3')
            a1[i] = 'M';
```

```
            if ((a1[j] >= '0' && a1[j] <= '9') && (a1[k] >= '0' && a1[k] <= '9'))
            {
                for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
                {
                  a1[h] = a1[h + 1];
                }
                for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
                {
                  a1[h] = a1[h + 1];
                }
            }
            else
            {
                for ( h = j; h < strlen(a1); ++h) ///Убираем ненужный символ
                {
                  a1[h] = a1[h + 1];
                }
            }
        }
        i++;
    }
}

void every (char* a1) ///Функция сцепления в строку (готова)
{
    int i=0, j=0;
    for (i=0; i<strlen(a1); i++)
    {
        all[all_int]=a1[i];///Сцепляем строку в одну большую
        all_int++;
    }
    ///all_int++;
    all[all_int]=';';///И разделяем их точкой с запятой
    all_int++;
}

int amount (char* digits) ///Функция плодсчета переменных (готова)
{
    int h=0, v=0;
    unsigned char sym;
    while (digits[v]!='\0')
    {
        sym=digits[v];
        h=sym;
        c[h]++;
        v++;
    }
}
```

0.3 Общий листинг программы

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


char st[80];
char all[256];
int n = 80;
int kol = 0;
int counter = 0;
int all_int=0;
int c[256]={0};
FILE* input;
FILE* output;


char digit (char* out);

void f(char* out, char y) ///Функция таблицы истинности
{
    int i=0, size, j=0, f=0, t=1;
    int a=0, h=0, g=0, v=0, c=0;
    char out1[200];
    char symbol;
    output = fopen("data_out.txt", "w");
    if (output == NULL)
    {
        printf("Eror [cant open output file");
    }
    size = 1 << kol;
    v=strlen(out);
    for(a = 0; a < size; a++)
    {
        j=0;
        for (f = 0; f < v; f++)
            out1[f] = out[f];
        out1[f]='\0';
        for (i = kol - 1; i >= 0; i--)
        {
        for(c = 0; c < strlen(out); c++)
            {
            if (((out1[j] >= 'a' && out1[j] <= 'z') || (out1[c] >= 'a' && out1[c] <= 'z')) ||
            ((out1[j] >= 'A' && out1[j] <= 'Z') || (out1[c] >= 'A' && out1[c] <= 'Z')))
                {
                if ((out1[c] >= 'a' && out1[c] <= 'z') || (out1[c] >= 'A' && out1[c] <= 'Z'))
                        j=c;
                    h=j+1;
```

```
                    while (out1[h]!='\0')
                    {
                        if (out1[h] == out1[j])
                        {
                            g = (a >> i) & 1;
                            symbol = g + '0';
                            out1[h] = symbol;
                            h++;
                        }
                        h++;
                    }
                    g = (a >> i) & 1;
                    symbol = g + '0';
                    out1[j] = symbol;
                    j++;
                    break;
                }
            }
        }
        if (digit(out1) == '1')
        {
            counter++;
            fprintf(output, "%d", t);
            fprintf(output, ". ");
            for (i = kol - 1; i >=0; i--)
            {
                fprintf(output, "%d ", ((a>>i)&1) );
            }
            fprintf(output, "\n");
            t++;
        }
    }
}

int priority (char a) ///Функция выявления приоритета
{
    int n=0;
    switch (a)
    {
    case '!':
        { n = 5; break; }
    case '^':
        { n = 4; break; }
    case '+':
        { n = 3; break; }
    case '-':
    case '=':
        { n = 2; break; }
    case '(':
```

```
        { n = 1; break; }
    }
    return n;
}

void operation_plus (int *top, int x) ///Функция для заноса операций в стэк
{
    *top=*top+1;
    st[*top]=x;
}

char operation_minus (int *top)
{
    char x;
    x=st[*top];
    *top=*top-1;
    return x;
}

char opz(char* a, char* out, int top, int k, int p) ///Функция преобразования в ОПЗ
{
    int i=0, j=0, r, v=0;
    int h;
    unsigned char sym;
    int c[256]={0};
    char digits[256];
    while ( a[k]!='\0' )
    {
        if ( a[k]==')' )
        {
            while( st[top]!='(' )
            {
                out[p++]=operation_minus(&top);
            }
            operation_minus(&top);
        }

        if (( a[k] >='a' && a[k]<='z') || ( a[k] >='A' && a[k]<='Z'))
        {
            out[p++]=a[k];
        }

        if ( a[k] >='0' && a[k]<='9')
        {
            out[p++]=a[k];
        }
        if ( a[k] == '(' )
        {
            operation_plus(&top, '(');
```

```
        }

        if ( a[k]=='+' || a[k]=='-' || a[k]=='^' || a[k]=='=' || a[k]=='!' )
        {
            if ( top==0 )
            {
                operation_plus(&top, a[k]);
            }

            else
            {
                if ( priority(st[top]) < priority(a[k]) )
                {
                    operation_plus(&top, a[k]);
                }
                else
                {
                    while ( priority(st[top]) >= priority(a[k]) )
                    {
                        out[p++]=operation_minus(&top);
                    }
                    operation_plus(&top, a[k]);
                }
            }
        }

        k++;
    }

    while(top!=0 || top > 0)
    {
        out[p++]=operation_minus(&top);
    }
    out[p++] = 0;
    return *out;
}

char digit (char* out) ///Функция счета
{
    int t=0, j=0, k=0, v=0, l=0, v1=0;
    int x, y, rez=0;
    char symbol;
    v1=strlen(out);
    t=0;
    while (out[t]!='\0')
    {
        while (out[t]!=';')
        {
        ///t++;
```

```
if ((out[t] != '+') || (out[t] != '=') || (out[t] != '-') || (out[t] != '^')
|| (out[t] != '!'))
{
    t++;
}
if ((out[t] == '+') || (out[t] == '=') || (out[t] == '-') || (out[t] == '^')
|| (out[t] == '!'))
{
    j=t-1;
    k=t-2;
    if (out[t] == '+')
    {
        x=out[k]-'0';
        y=out[j]-'0';
        if ((x == 1) && (y == 1))
        {
            rez = 1;
        }
        else
        {
            rez = x+y;
        }
        symbol=rez+'0';
        out[k]=symbol;
        out[j]=out[t+1];
        j=j+1;
        for (long i = j; i < v1; ++i)
        {
          out[i] = out[i + 1];
        }
        for (long i = j; i < v1; ++i)
        {
          out[i] = out[i + 1];
        }
        --v1;
        t=0;

    }
    if (out[t] == '=')
    {
        x=out[k]-'0';
        y=out[j]-'0';
        if (((x == 1) && (y == 1)) || ((x == 0) && (y == 0)))
        {
            rez = 1;
        }
        else
        {
            rez = 0;
```

```
    }
    symbol=rez+'0';
    out[k]=symbol;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    t=0;
}
if (out[t] == '-')
{
    x=out[k]-'0';
    y=out[j]-'0';
    if ((x == 1) && (y == 0))
    {
        rez = 0;
    }
    else
    {
        rez = 1;
    }
    symbol=rez+'0';
    out[k]=symbol;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    for (long i = j; i < v1; ++i)
    {
      out[i] = out[i + 1];
    }
    --v1;
    t=0;
}
if (out[t] == '^')
{
    x=out[k]-'0';
    y=out[j]-'0';
    rez = x * y;
    symbol=rez+'0';
    out[k]=symbol;
    for (long i = j; i < v1; ++i)
```

```
            {
              out[i] = out[i + 1];
            }
            --v1;
            for (long i = j; i < v1; ++i)
            {
              out[i] = out[i + 1];

            }
            --v1;
            t=0;
        }
        if (out[t] == '!')
        {
            y=out[j]-'0';
            if (y == 1)
            {
                rez = 0;
            }
            else
            {
                rez = 1;
            }
            symbol=rez+'0';
            out[j]=symbol;
            for (long i = j+1; i < v1; ++i)
            {
              out[i] = out[i + 1];
            }

            --v1;
            t=0;
        }
    }

    }
    t++;
}
v1=1;
while (out[v]!='\0')
{
    if (out[v] >= '0' && out[v] <= '1')
    {
        l=out[v]-'0';
        v1=v1*l;
    }
    v++;
}
symbol=v1+'0';
```

```
    return symbol;
}

int amount (char* digits) ///Функция плодсчета переменных
{
    int h=0, v=0;
    unsigned char sym;
    while (digits[v]!='\0')
    {
        sym=digits[v];
        h=sym;
        c[h]++;
        v++;
    }

}

void every (char* a1) ///Функция сцепления в строку
{
    int i=0, j=0;
    for (i=0; i<strlen(a1); i++)
    {
        all[all_int]=a1[i];
        all_int++;
    }
    ///all_int++;
    all[all_int]=';';
    all_int++;
}

int check(char* a1) ///Функция проверки синтаксиса
{
    int i=0, j=0;
    for (i=0; i<strlen(a1); i++)
    {
        if (a1[i]  == '(')
        {
            j=i+1;
            if (a1[j] == ')')
            {
                printf("Syntax Eror [there is a ) right after the (]!");
                return 0;
            }
            else
            {
                while (a1[j]!='\0')
                {
                    if (a1[j] == ')')
                        goto metka;
```

```
                    j++;
                }
                printf("Syntax Eror [there is no a ) after the (]!");
                return 0;
                metka:;
                j++;
                ///printf(" ");
            }
        }
        j=i+1;
        if ((a1[i]>='a' && a1[i]<='z') || (a1[i]>='A' && a1[i]<='Z'))
        {
            if ((a1[j]>='a' && a1[j]<='z') || (a1[j]>='A' && a1[j]<='Z'))
            {
                printf("Syntax Eror[there is a letter right after the another one]!");
                return 0;
            }
        }
        if (a1[i]>='0' && a1[i]<='9')
        {
            if (a1[j]>='0' && a1[j]<='9')
            {
                printf("Syntax Eror[there is a digit right after the another one]!");
                return 0;
            }
        }
        if (a1[i]=='^' || a1[i]=='+' || a1[i]=='-' || a1[i]=='=' || a1[i]=='!')
        {
            if (a1[j]=='^' || a1[j]=='+' || a1[j]=='-' || a1[j]=='=')
            {
                printf("Syntax Eror[there is a operation right after the another one]!");
                return 0;
            }
        }
    }
}

char convert(char* a1)
{
    int i=0, j=0, h=0, k=0, v1=0;
    ///v1=strlen(a1);
    while (a1[i]!='\0')
    {
        if (a1[i] == 'x')
        {
            j=i+1;
            k=i+2;
            if (a1[j] == '1')
                a1[i] = 'a';
```

```
    if (a1[j] == '2')
        a1[i] = 'b';
    if (a1[j] == '3')
        a1[i] = 'c';
    if (a1[j] == '4')
        a1[i] = 'd';
    if (a1[j] == '5')
        a1[i] = 'e';
    if (a1[j] == '6')
        a1[i] = 'f';
    if (a1[j] == '7')
        a1[i] = 'g';
    if (a1[j] == '8')
        a1[i] = 'h';
    if (a1[j] == '9')
        a1[i] = 'i';
    if (a1[j] == '1' && a1[k] == '0')
        a1[i] = 'B';
    if (a1[j] == '1' && a1[k] == '1')
        a1[i] = 'C';
    if (a1[j] == '1' && a1[k] == '2')
        a1[i] = 'D';
    if (a1[j] == '1' && a1[k] == '3')
        a1[i] = 'K';
    if ((a1[j] >= '0' && a1[j] <= '9') && (a1[k] >= '0' && a1[k] <= '9'))
    {
        for ( h = j; h < strlen(a1); ++h)
        {
          a1[h] = a1[h + 1];
        }
        for ( h = j; h < strlen(a1); ++h)
        {
          a1[h] = a1[h + 1];
        }
    }
    else
    {
        for ( h = j; h < strlen(a1); ++h)
        {
          a1[h] = a1[h + 1];
        }
    }
}
if (a1[i] == 'y')
{
    j=i+1;
    if (a1[j] == '1')
        a1[i] = 'j';
    if (a1[j] == '2')
```

```
            a1[i] = 'k';
        if (a1[j] == '3')
            a1[i] = 'l';
        if (a1[j] == '4')
            a1[i] = 'm';
        if (a1[j] == '5')
            a1[i] = 'n';
        if (a1[j] == '6')
            a1[i] = 'o';
        if (a1[j] == '7')
            a1[i] = 'p';
        if (a1[j] == '8')
            a1[i] = 'q';
        if (a1[j] == '9')
            a1[i] = 'r';
        if (a1[j] == '1' && a1[k] == '0')
            a1[i] = 'E';
        if (a1[j] == '1' && a1[k] == '1')
            a1[i] = 'F';
        if (a1[j] == '1' && a1[k] == '2')
            a1[i] = 'G';
        if (a1[j] == '1' && a1[k] == '3')
            a1[i] = 'L';
        if ((a1[j] >= '0' && a1[j] <= '9') && (a1[k] >= '0' && a1[k] <= '9'))
        {
            for ( h = j; h < strlen(a1); ++h)
            {
              a1[h] = a1[h + 1];
            }
            for ( h = j; h < strlen(a1); ++h)
            {
              a1[h] = a1[h + 1];
            }
        }
        else
        {
            for ( h = j; h < strlen(a1); ++h)
            {
              a1[h] = a1[h + 1];
            }
        }
    }
    if (a1[i] == 'z')
    {
        j=i+1;
        if (a1[j] == '1')
            a1[i] = 's';
        if (a1[j] == '2')
            a1[i] = 't';
```

```
            if (a1[j] == '3')
                a1[i] = 'u';
            if (a1[j] == '4')
                a1[i] = 'v';
            if (a1[j] == '5')
                a1[i] = 'w';
            if (a1[j] == '6')
                a1[i] = 'x';
            if (a1[j] == '7')
                a1[i] = 'y';
            if (a1[j] == '8')
                a1[i] = 'z';
            if (a1[j] == '9')
                a1[i] = 'A';
            if (a1[j] == '1' && a1[k] == '0')
                a1[i] = 'H';
            if (a1[j] == '1' && a1[k] == '1')
                a1[i] = 'I';
            if (a1[j] == '1' && a1[k] == '2')
                a1[i] = 'J';
            if (a1[j] == '1' && a1[k] == '3')
                a1[i] = 'M';
            if ((a1[j] >= '0' && a1[j] <= '9') && (a1[k] >= '0' && a1[k] <= '9'))
            {
                for ( h = j; h < strlen(a1); ++h)
                {
                  a1[h] = a1[h + 1];
                }
                for ( h = j; h < strlen(a1); ++h)
                {
                  a1[h] = a1[h + 1];
                }
            }
            else
            {
                for ( h = j; h < strlen(a1); ++h)
                {
                  a1[h] = a1[h + 1];
                }
            }
        }
        i++;
    }
}

int main()
{
    int top=0;
    char a[80],
```

```
     symbol,
     out[256];
char *a1;
int k=0,
    i=0,
    p=0;
counter =0;
input = fopen("data.txt", "r");
if (input == NULL)
{
    printf("Eror");
    return 0;
}
while (1)
{
    a1 = fgets (a, sizeof(a), input);
    if (a1 == NULL)
    {
        if(feof(input)!=0)
        {
            break;
        }
        else
        {
            printf("Eror");
            break;
        }
    }
    if(a1[strlen(a1)-1]=='\n')
    {
        a1[strlen(a1)-1]='\0';
    }
    convert(a1);
    if (check(a1) == 0)
    {
        return 0;
    }
    amount(a1);
    opz(a1, out, top, k, p);
    every(out);
}
for (i=0; i< 256; i++)
    if ((i>=65 && i<=90) || (i>=97 && i<=122))
        if(c[i]>0)
            kol++;
f(all, symbol);
printf("Answer: "); printf("%d\n", counter);
printf("Number of letters: "); printf("%d\n", kol);
printf("*Press Enter to stop the programm*");
```