**Практическая работа**

**Задание. Разработать парсер для своего языка программирования.**

# Что нужно для создания нового языка программирования, какие средства и какие языки программирование нужны будут участвовать при создания нового языка?

Транслятор можно написать на любом языке. Код будет состоять из нескольких модулей.   
1. Лексер - преобразует исходный текст в последовательность токенов.   
a = b + 1; превратит в что-то типа {variable("a"), assign, variable("b"), add, number("1")}  
2. Парсер строится на основе [грамматики](http://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0). Преобразует последовательность токенов в дерево.

. assign

/ \

var('a') add

/ \

var('b') number(1)

3. Далее обходя полученное дерево генерируем код

Новый язык программирования можно написать на любом другом существующем языке программирования. Соответственно средства зависят от языка.

Внимание вопрос - "А ЗАЧЕМ?"?  
При создании того или иного инструмента - преследется какая то цель, не важно, что это колесо, свисток или ручка.  
Какая цель у вас?

Зависит от того, какой части Вы хотите уделить особое внимание. Если просто создать свой язык с необычным синтаксисом, то напишите фронтэнд к llvm (или gcc, или jvm, или parrot, или что Вам больше нравится). Если хотите потренировать кодогенерацию - соответственно бекенд к llvm или др на Ваш выбор. Если хотите полностью свой компилятор , то придётся ещё и свой оптимизатор и кодогенератор делать. Но если Вам это для обучения, то попробуйте просто поучаствовать в разработке llvm/gcc. Если совсем делать нечего - почитайте как они работают и попробуйте сделать что-то в таком роде.

***Подзадачи***

1. Составить свой синтаксис языка
   1. Определить операторы и разделители
   2. Признак начала и конца программы
2. Написать программу на своем языке в текстовом файле
3. Написать модуль построчного считывания программы из файла
4. Написать модуль выделения оператора в последовательности строк
5. Написать модуль синтаксического анализа текста в строке

Работа с файлами в С++

Для работы с файлами используются специальные типы данных, называемые *потоками.*Поток **ifstream**служит для работы с файлами в режиме чтения, а **ofstream** в режиме записи. Для работы с файлами в режиме как записи, так и чтения служит поток **fstream**.

В программах на C++ при работе с текстовыми файлами необходимо подключать библиотеки **iostream** и**fstream**.

Для того чтобы записывать данные в текстовый файл, необходимо:

1. описать переменную типа **ofstream**.
2. открыть файл с помощью функции **open**.
3. вывести информацию в файл.
4. обязательно закрыть файл.

Для считывания данных из текстового файла, необходимо:

1. описать переменную типа **ifstream**.
2. открыть файл с помощью функции **open**.
3. считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла.
4. закрыть файл.

**Чтение информации из текстового файла**

Для того чтобы прочитать информацию из текстового файла, необходимо описать переменную типа **ifstream**. После этого нужно открыть файл для чтения с помощью оператора **open**. Если переменную назвать **F**, то первые два оператора будут такими:

|  |  |
| --- | --- |
| 1 2 | ifstream F; F.open("D:\\sites\\accounts.txt", ios::in); |

После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры, только вместо **cin** нужно указать имя потока, из которого будет происходить чтение данных.

Например, для чтения данных из потока **F** в переменную**a**, оператор ввода будет выглядеть так:

**F>>a;**

Два числа в текстовом редакторе считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки. Хорошо, когда программисту заранее известно, сколько и какие значения хранятся в текстовом файле. Однако часто известен лишь тип значений, хранящихся в файле, при этом их количество может быть различным. Для решения данной проблемы необходимо считывать значения из файла поочередно, а перед каждым считыванием проверять, достигнут ли конец файла. А поможет сделать это функция **F.eof()**. Здесь **F**- имя потока функция возвращает логическое значение: **true** или **false**, в зависимости от того достигнут ли конец файла.

Следовательно, цикл для чтения содержимого всего файла можно записать так:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | //организуем для чтения значений из файла, выполнение //цикла прервется, когда достигнем конец файла, //в этом случае F.eof() вернет истину while (!F.eof()) { //чтение очередного значения из потока F в переменную a F>>a; //далее идет обработка значения переменной a } |

***Посимвольное чтение текста.***

Функция get(), которая является методом istream, применяется для посимвольного чтения текстового файла.

**Пример 15.1.** Посимвольное чтение файла и вывод его на экран.

# include <fstream>

# include <iostream>

using namespace std;

int main()

{ char sym;

ifstream in\_file (“Input\_file.txt”, ios::in);

if (! in\_file) { cerr<<”Error input file”<<endl;

exit(1);

}

while (in\_file) { in\_file.get(sym);

cout<<sym;

}

cout<<endl;

return 0;

}

***Построчное чтение файла.***

Обычно построчное чтение и запись файлов работают быстрее посимвольных действий. Для чтения строки из файла воспользуемся функцией getline(), которая является методом класса ifstream. Функция читает строку (в том числе и разделители), пока не встретит символ новой строки `\n', помещая ее в буфер (первый аргумент функции). Максимальный размер буфера задается как второй аргумент функции.

**Пример 15.3.**

# include <fstream>

# include <iostream>

using namespace std;

void main()

{ const int LEN=80;

char BUF[LEN];

ifstream in\_file (“Input\_file.txt”, ios::in);

if (! in\_file) { cerr<<”Error input file”<<endl;

exit(1);

}

while (in\_file) { in\_file.getline(BUF,LEN);

cout<<BUF<<endl;

}

}

Другой вариант

**Код C++ Вывести текст из файла на экран**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | #include <conio.h>  #include <iostream.h>  #include <fstream.h>  const N=256; //Константный размер строки  char \*FName="C:\MyFile.txt"; //Путь к файлу  /\*Вывод текста из файла на экран\*/  void ReadFile()  {    cout<<endl<<"ReadFile:  "; //Для красоты    char S[N]={""}; //В S будут считываться строки        ifstream in1(FName); //Открыли файл для чтения  while (!in1.eof()) //Будем читать информацию пока не дойдем до конца файла     {       in1.getline(S,N); //Построчное считывание информации в S       cout<<S<<endl; //Вывод очередной строки на экран     }         in1.close();  //Закрыли открытый файл  }  /\*Ввод текста в файл\*/  void CreateFile()  {  cout<<"Create File:  "<<endl; //Для красоты  char S[256]=""; //В S будет считываться строка  int count=0; //Число строк для ввода в файл  cout<<"Strok budet:  " ;  cin>>count; //Определили сколько строк вводить  /\*Главная часть функции\*/       ofstream out1(FName);  //Открыли файл для записи    for (int i=0;i<count+1;i++)  //Цикл по количеству нужных строк    {      cin.getline(S,N); //Запоминаем в S то что ввели с клавиатуры      out1<<S<<endl; //Записали S в файл и дописали перенос строки    }    out1.close();  }      void main()  {     clrscr();       CreateFile();//Функция записывает строки в файл       ReadFile();   //Функция читает строки из файла и выводит их на экран     getch();  } |