

**Министерство образования и науки Российской Федерации**

**Федеральное агентство по образованию**

**Государственное образовательное учреждение высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Робототехника и комплексная автоматизация» (РК)**

**Кафедра «Системы автоматизированного проектирования» (РК6)**

****

**Отчет по лабораторной работе \_\_\_\_\_Y14\_\_\_\_\_\_\_ по курсу**

**«САПР»**

**Студент: Петраков Станислав**

**Группа:** РК6-36Б

**Преподаватель:** Волосатова Т.М.

Проверил:

Дата:

**Задание**

**Текст задания:**

Разработать синтаксический анализатор для грамматического разбора входного потока цифровых данных с целью распознавания записей двоичных наборов, которые соответствуют формальному языку L={1n02n | n>0}. Записи двоичных наборов должны передаваться синтаксическому анализатору строками потока стандартного ввода. Результаты грамматического разбора входного потока должны отображать диагностические сообщения в потоке стандартного вывода.

**Введение**

Настоящий документ определяет техническое задание на разработку программы синтаксического анализатора (далее по тексту - программа

VOK) для проверки соответствия двоичного числа формату L={1n02n | n>0}.

**Основания для разработки**

Программа VOK разрабатывается в рамках лабораторной работы по курсу "Программное обеспечение САПР" для практического изучения этапа синтаксического анализа в процедурах трансляции формальных языков.

**Назначение разработки**

Программа VOK предназначается для реализации грамматического разбора двоичных чисел.

**Требования к программе**

1. **Требования к функциональным характеристикам**
   1. Программа VOK должна обеспечивать синтаксический анализ цифр в произвольном двоичном числе, которое передается символьной строкой из потока стандартного ввода.
   2. Программа VOK должна предусматривать возможность обработки 2-ух цифр: нуля и единицы, которые могут одновременно присутствовать и повторяться во входном двоичном числе в произвольном числе экземпляров.
   3. Программа VOK должна осуществлять контроль двоичного числа.
   4. Программа VOK должна проверять, что число соответсвует заданному формату.
   5. Программа VOK должна учитывать общепринятые математические правила по записи двоичных чисел.
   6. Программа VOK должна обнаруживать любые нарушения, которые противоречат требованиям пп. 1.3 - 1.5.
   7. Любые нарушения записи двоичного числа, не соответствующие требованиям пп. 1.3 - 1.5, во входной строке из потока стандартного ввода должны сопровождаться информационным сообщением "syntax error" в потоке стандартного вывода.
   8. Результат распознавания программой VOK двоичного числа, соответствующего требованиям пп. 1.3 - 1.5, должен сопровождаться отображением в потоке стандартного вывода информационным сообщением "Correct!".
2. **Требования к надежности**
   1. Программа VOK не должна иметь каких-либо ограничений по длине двоичного числа, кроме внутренних ограничений инструментальных средств, использованных для ее реализации.
3. **Требования к составу и параметрам технических средств** 
   1. Программа VOK должна быть разработана исходя из возможности реализации на стандартном составе технических средств компьютеров любой архитектуры, после соответствующей трансляции исходного кода.
4. **Условия эксплуатации**
   1. Программа VOK должна быть ориентирована на эксплуатацию в операционной среде OS UNIX любой версии.
   2. Программа VOK должна быть реализована в виде выполняемого файла, по которому она должна вызываться средствами любого командного процессора OS UNIX.
   3. Программа VOK должна эксплуатироваться в интерактивном режиме, читая строку входного выражения из потока стандартного ввода и отображая результат ее обработки в потоке стандартного вывода.
5. **Требования к информационной и программной совместимости**
   1. Чтобы обеспечить выполнение требуемых технических характеристик, программа VOK должна реализовывать синтаксический анализ любой входной строки двоичного числа из потока стандартного ввода.
   2. Синтаксический анализатор программы VOK должен обеспечивать грамматический разбор цифр с целью установить соответствие или несоответствие содержащей их строки потока стандартного ввода требуемому формату.
   3. Для выполнения грамматического разбора синтаксический анализатор программы VOK должен реализовывать однозначную контекстно-свободную грамматику простого предшествования, которая ориентирована на обработку строки заданного двоичного числа из потока стандартного ввода и далее по тексту называется грамматикой двоичного числа.
   4. Грамматику двоичного числа синтаксического анализатора программы VOK должны составлять следующие элементы:

* терминальные символы (терминалы), соответствующие структурным единицам (лексемам) входного двоичного числа;
* начальный нетерминальный символ (начальный нетерминал), к которому приводится входное двоичное число;
* нетерминальные символы (нетерминалы), обозначающие допустимые варианты комбинации лексем во входном двоичном числе;
* система продукций (правил вывода), обеспечивающая грамматический разбор входного двоичного числа.
  1. Терминальные символы грамматики двоичного числа синтаксического анализатора программы VOK должны представляться лексемами, специфицированными следующими литералами: ‘0’, ’1’, которые обозначают коды ASCII соответствующих цифр нуля и единицы.
  2. Нетерминальные символы грамматики скобок должны обозначаться следующими именами: exp0, exp1.

exp0: блок нулей;

exp1: блок единиц.

* 1. Начальный нетерминал грамматики двоичных чисел синтаксического анализатора программы VOK должен обозначаться именем input. Он должен выводиться из любой корректной последовательности произвольных двоичных блоков. Грамматический вывод начального нетерминала из любой входной строки, содержащей корректную запись числа, должна обеспечивать система продукций грамматики двоичных чисел.
  2. Система продукций (правил вывода) грамматики двоичных чисел синтаксического анализатора программы VOK должна обеспечивать грамматический разбор произвольной входной строки потока стандартного ввода путем приведения терминалов и нетерминалов к начальному нетерминалу методом анализа снизу-вверх по дереву вывода. Приведение входной строки к начальному нетерминалу по правилам вывода грамматики двоичных чисел должно означать успех грамматического разбора двоичного числа, т.е. распознавание корректного двоичного числа. Альтернативный результат должен рассматриваться как синтаксическая ошибка последовательности нулей и единиц во введенной строке.
  3. Для разработки синтаксического анализатора программы VOK, необходимо использовать генератор синтаксических анализаторов (далее по тексту - YACC) из состава OS UNIX, инструментальные средства которого ориентированы на обработку файла спецификаций (далее по тексту, Yacc- файл) проектируемого синтаксического анализатора.
  4. При разработке синтаксического анализатора программы VOK необходимо составить Yacc-файл, отражающий специфику грамматического разбора двоичного числа, и сохранить его в выбранном доступном рабочем каталоге файловой системы OS UNIX.
  5. Проектируемый Yacc-файл должен состоять из 3-х секций: деклараций, правил и функций. Разделителем секций должны быть символические пары %%, расположенные в начальных позициях содержащих их строк Yacc-файла. Каждая секция Yacc-файла должна содержать соответствующее число комментариев в формате, принятом для системы программирования C.
  6. Секция деклараций Yacc-файла должна включать:
     + спецификацию блока внешних описаний, ограниченную директивами %{ и %}.
  7. В секции правил Yacc-файла должны быть приведены описания продукций приведения нетерминалов грамматики двоичных чисел в соответсвии с требованием пп. 5.5 - 5.8.
  8. Каждая продукция секции правил Yacc-файла должна быть задана в нотации, близкой к форме Бэкуса-Наура, где в левой части указывается приводимый нетерминал, а в правой - последовательность терминалов и/или нетерминалов грамматики двоичных чисел, которые перечисляются через пробел. Для разделения частей продукции должен использоваться символ двоеточия (:). Каждую продукцию нужно начинать с новой строки и завершать либо символом точки с запятой (;), либо блоком действий в фигурных скобках.
  9. Продукции секции правил Yacc-файла, приведение нетерминалов которых необходимо сопровождать функциональной обработкой, должны содержать блоки действий. Блоки действий должны располагаться в правых частях продукций и ограничиваться парой фигурных скобок. Внутри блоков действий можно использовать любые конструкции и вызовы функций системы программирования C.
  10. Продукции секции правил Yacc-файла, необходимые для приведения нетерминалов, должны быть специфицированы с помощью альтернативных правил. В каждом из них альтернативы свертки различных нетерминалов правой части должны быть объединены с помощью оператора ИЛИ, который обозначается символом вертикальной черты (|).
  11. Продукции секции правил Yacc-файла, необходимые для приведения нетерминалов, должны задаваться в форме леворекурсивных правил.
  12. Чтобы иметь возможность абстрагироваться от одиночного блока, в лево-рекурсивных продукциях секции правил Yacc-файла необходимо предусмотреть альтернативные правила разименования, которые должны обеспечивать приведение нетерминалов.
  13. Секция функций Yacc-файла должна содержать исходный код, оформленный по правилам системы программирования C, для 3-х функций с предопределенными именами: yylex(), yyerror() и main(), которые должны иметь целочисленный (типа int) код возврата. Исходный код перечисленных функций используется непосредственно для формирования кода программы DIGITS.
  14. Функция yylex() секции функций Yacc-файла должна выполнять лексический анализ входной строки из потока стандартного ввода со следующими целями: обнаружить двоичную цифру, распознать 0 или 1 и зафиксировать последовательность следования двоичных цифр. Вызов функции yylex() будет автоматически производиться при грамматическом разборе входной строки для получения очередных цифр, пока не достигнут символ перевода строки ('\n').
  15. Для достижения целей лексического анализа, исходный код функции yylex() должен предусматривать:
* побайтное чтение любой заданной строки входного двоичного числа из потока стандартного ввода системы программирования C;
* распознавание цифр 0 или 1 во входной строке системы программирования C;
* возврат кодов ASCII обнаруженных цифр с помощью оператора return системы программирования C;
* возврат нулевого кода при достижении конца входной строки с помощью оператора return(0) системы программирования C.
  1. Функция yyerror() секции функций Yacc-файла должна обеспечивать обработку ошибок, обнаруженных при грамматическом разборе входной строки двоичного числа. Ее вызов будет происходить автоматически, когда двоичное число не может быть приведено к начальному нетерминальному символу грамматики двоичного числа, обеспечивая аварийное прерывание грамматического разбора текущей входной строки потока стандартного ввода.
  2. Спецификация функции yyerror() в Yacc-файле, должна иметь единственный аргумент типа (char \*), который по умолчанию содержит адрес предопределенной символьной строки: "syntax error". Аргумент функции yyerror() следует использовать для формирования диагностического сообщения в соответствии с требованием п. 1.7. Для отображения диагностического сообщения в потоке стандартного вывода, рекомендуется применить библиотечную функцию printf() системы программирования С.
  3. Спецификация функции main() в секции функций Yacc-файла должна содержать исходный код основной функции программы DIGITS, который обеспечивает обращение к функции синтаксического анализа yyparse() и передачу ее кода возврата во внешнюю операционную среду. Исходный код функции yyparse() формируется генератором синтаксических анализаторов YACC по Yacc-файлу и предусматривает вызов функции yylex() для выполнения лексического анализа, также как передачу управления функции yyerror() для обработки ошибок грамматического разбора.
  4. Исходный код секции функций Yacc-файла и функции yyparse() образует исходный код программы VOK, который должен формироваться генератором синтаксических анализаторов YACC в файле с предопределенным именем y.tab.c в текущем рабочем каталоге файловой системы OS UNIX. Выполняемый модуль программы VOK должен быть создан по файлу исходного кода y.tab.c в выполняемом файле средствами компилирующей системы программирования C.

**Стадии и этапы разработки**

Процесс разработки программы VOK целесообразно разделяться на следующие 3 стадии:

* составить Yacc-файл vok.y в выбранном рабочем каталоге файловой системы OS UNIX, используя любой текстовый редактор для редактирования в консольном режиме;
* получить исходный код синтаксического анализатора в файле y.tab.c текущего каталога файловой системы OS UNIX, обработав Yacc-файл командой yacc, следующим образом: yacc vok.y.
* сформировать выполняемый модуль в файле текущего каталога файловой системы OS UNIX, компилируя исходный код синтаксического анализатора в файле y.tab.c следующей командой: cc -o vok y.tab.c.

Результаты разработки программы VOK необходимо оформить в виде проектного документа, содержащего описание грамматики двоичного числа и файла спецификаций для генератора синтаксических анализаторов YACC.

**Порядок контроля и приемки**

1. Для проверки функционирования программы VOK должны быть предложены контрольные примеры, предусматривающие стандартный ввод корректных и некорректных двоичных чисел.

2. Для проверки программы VOK должен быть организован вызов выполняемого файла в консольном режиме работы OS UNIX или режиме эмуляции терминала операционной среды X Window System.

**Приложение 1:**

%{  
[#include](https://vk.com/im?sel=63386677&st=%23include) <stdio.h>  
int yylex (void);  
void yyerror (char const \*s);  
%}  
  
%token ZERO  
%token ONE  
%%  
  
input: {printf("Enter the line:\n");}  
| input line;  
  
line: error '\n' {yyerrok;}  
| exp1 exp0 '\n' {  
if($2 == $1\*2)   
printf("syntax right\n");  
}  
;  
  
exp0: ZERO ZERO {$$+=2;}  
| exp0 ZERO ZERO{$$+=2;}  
;  
exp1: ONE {$$++;}  
| exp1 ONE {$$++;}  
;  
  
%%  
  
int yylex(void) {  
int c;  
c = getchar();  
  
switch (c) {  
case '0': return ZERO;  
case '1': return ONE;  
case EOF: return 0;  
default: return c;  
}  
  
}  
  
void yyerror(char const \*s) {  
fprintf(stderr, "%s\n", s);   
}  
  
int main (void) {  
return yyparse ();  
}

**Приложение 2:**

При разработке синтаксических анализаторов в OS UNIX для проектирования программы UNITS рекомендуется использовать литературные источники, перечисленные ниже:

1. Рейуорд-Смит В.Дж. Теория формальных языков. Вводный курс, M.: Радио и связь, 1988.
2. Тихомиров В.П., Давидов М.И. Операционная система ДЕМОС: инструментальные средства программирования, М.: Финансы и статистика, 1988.
3. bigor.bmstu.ru – база и генератор образовательных ресурсов
4. Лекции по курсу САПР