Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по лабораторной работе №2**

Дисциплина: Транслирующие системы

Тема: Программирование синтаксического разбора на языке yacc

Выполнил студент гр. 5130901/20103 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Вагнер

(подпись)

Принял старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Цыган

(подпись)

“ ” 2024 г.

Санкт-Петербург

2024

**Оглавление**

[Цель работы 4](#_Toc179888224)

[Ход работу 4](#_Toc179888225)

[\_date/v1 5](#_Toc179888226)

[\_date/v2 11](#_Toc179888227)

[\_date/v3 14](#_Toc179888228)

[\_date/v4 20](#_Toc179888229)

[\_date/v5 24](#_Toc179888230)

[list/v0 25](#_Toc179888231)

[list/v1 33](#_Toc179888232)

[Вывод: 37](#_Toc179888233)

**Список иллюстраций**

[Листинг 1 – Простейший синтаксический анализатор на языке yacc 5](#_Toc179887918)

[Листинг 2 (v1.l) – Модуль на языке lex для синтаксического анализатора 5](#_Toc179887919)

[Рис. 1 – Текстовые файлы с тестами для v1.y 6](#_Toc179887920)

[Рис. 2 – Результаты тестирования для v1.y 6](#_Toc179887921)

[Рис. 3 – Результат работы программы в режиме отладки 7](#_Toc179887922)

[Рис.4 – файл zz.c 8](#_Toc179887923)

[Рис. 5 – Результат работы в режиме трассировки 9](#_Toc179887924)

[Рис. 6 – Фрагмент файла y.tab.h 10](#_Toc179887925)

[Рис. 7 (v2.y) – Литерал в определении нетерминального символа 10](#_Toc179887926)

[Рис. 8 (v2.l) – Передача литерала из lex-модуля 11](#_Toc179887927)

[Рис. 9 – Файлы с тестами для v2.y 11](#_Toc179887928)

[Рис. 10 - Результат тестирования для v2.y 12](#_Toc179887929)

[Рис. 11 – Модифицированный файл v2.y 12](#_Toc179887930)

[Рис. 12 – Модифицированный файл v2.l 12](#_Toc179887931)

[Рис. 13 – Результаты тестирования модифицированного v2.y 13](#_Toc179887932)

[Рис. 14 (v3.l) – Задание типа и величины сопутствующего значения 13](#_Toc179887933)

[Рис.15 (v3a.y) – Доступ к семантическим знаечниям 14](#_Toc179887934)

[Рис. 16 – Файлы с тестами для v3a.y 14](#_Toc179887935)

[Рис. 17 – Результаты тестирования для v3a.y 15](#_Toc179887936)

[Рис. 18 (v3b.y) – Проверка даты и вывод количества дней от 1970г. 15](#_Toc179887937)

[Рис. 19 - Файл abs\_state.c 16](#_Toc179887938)

[Рис. 20 – Файлы с тестами для v3b.y 17](#_Toc179887939)

[Рис. 21 – Результат тестирования v3b.y 17](#_Toc179887940)

[Рис. 22 (v3c.y) – Семантическое значение date и вычисление разницы между датами 18](#_Toc179887941)

[Рис. 23 – Текстовые файлы с тестами для v3c.y 18](#_Toc179887942)

[Рис. 24 – Результат тестирования для v3c.y 19](#_Toc179887943)

[Рис.25 – (v4a.y) – Определение сопутствующего значения нескольких типов 19](#_Toc179887944)

[Рис. 26 (v4.l) – Формирование сопутствующего значения в lex модуле 20](#_Toc179887945)

[Рис. 27 – Тестирование модуля v4a.l 21](#_Toc179887946)

[Рис. 28 (v4b.y) – Явное указание типа при обращении в $-переменной 21](#_Toc179887947)

[Рис. 29 – Файлы с тестами для v4b.y 21](#_Toc179887948)

[Рис. 30 – Результат тестирования для v4b.y 22](#_Toc179887949)

[Рис. 31 (v4c.y) – Задание типа при объявлении символа 22](#_Toc179887950)

[Рис. 32 – Результат тестирования для v4c.y 23](#_Toc179887951)

[Рис. 33 (v5.y) – Вычисление количества дней между двумя датами 23](#_Toc179887952)

[Рис. 34 – Результат тестирования для v5.y 24](#_Toc179887953)

[Рис. 35 (c1.l) – Лексический анализатор для разбора списка чисел 25](#_Toc179887954)

[Рис. 36 (c1.y) – Синтаксический анализатор для разбора списка чисел 25](#_Toc179887955)

[Рис. 37 – Результат тестирования c1.y 26](#_Toc179887956)

[Рис. 38 – Трассировка обработки для c1.y 26](#_Toc179887957)

[Рис. 39 –фрагмент файла y.output 27](#_Toc179887958)

[Рис. 41 (c2.l) – Включение ‘\n’ в лексический разбор 29](#_Toc179887959)

[Рис. 42 – Тестирование программы c2.l с2.y 29](#_Toc179887960)

[Рис. 43 – Тестирование программы c1.l с2.y 29](#_Toc179887961)

[Рис. 44 – Тестирование программы c2.l с1.y 30](#_Toc179887962)

[Рис. 45 – Трейсинг программы c2.l c1.y 30](#_Toc179887963)

[Рис. 46 (c3.l) – Удаление разделителей при лексическом разборе 31](#_Toc179887964)

[Рис. 47 – Тестирование программы c3.l с1.y 31](#_Toc179887965)

[Рис. 48 – Файл c.l 32](#_Toc179887966)

[Рис. 48 – Файл c1.y 32](#_Toc179887967)

[Рис.49 – Тестовый файл для c1.y 33](#_Toc179887968)

[Рис. 50 – Результат тестирования для c1.y 33](#_Toc179887969)

[Рис. 51 – Файл c2.y 34](#_Toc179887970)

[Рис. 52 – Результат тестирования для c2.y 34](#_Toc179887971)

Цель работы**.**

Ознакомление с основами разработки программ синтаксического разбора на языке Yacc.

Ход работы

Язык yacc (yet another compiler compiler) позволяет описать синтаксический разбор как набор правил, определяющих синтаксическую структуру ввода, с действиями на языке C.

Исходная программа транслируется yacc в модуль на языке C, в котором определена глобальная функция yyparse, реализующая алгоритм синтаксического разбора в соответствии с заданной грамматикой.

Функция yyparse многократно обращается к внешней функции yylex, которая должна возвращать код лексемы в виде целого положительного числа (или 0 в конце ввода). Код лексемы, возвращаемый yylex, может сопровождаться величиной в переменной yylval (т. н. сопутствующее, или семантическое значение). Интерфейс между функциями yyparse и yylex на этапе компиляции устанавливает заголовочный файл y.tab.h, сгенерированный yacc; там содержатся определения кодов лексем и типа переменной yylval.

Функция yyparse возвращает 0, если конец ввода обнаружен тогда, когда входная последовательность лексем соответствует правилу для символа верхнего уровня грамматики (стартовый символ). Ненулевой результат yyparse говорит о синтаксической ошибке: либо входная последовательность не соответствует ни одному из правил, либо в конце ввода не выполнено правило для стартового символа. В этом случае вызывается функция yyerror, которая должна быть, наряду с main, определена пользователем.

Важными для языка yacc являются лексемы, которые называются также терминальными символами, и символы, определенные через другие символы (т. е. конструкции из символов), которые называются нетерминальными.

\_date/v1

Рассмотрим взаимодействие модулей, написанных на lex и yacc, на примере программы из каталога \_date/v1. Эта программа только проверяет структуру ввода.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Листинг – Простейший синтаксический анализатор на языке yacc

В этой спецификации определены лексемы NUMBER и MONTH и задан стартовый символ — date. (Стартовый символ — это один из нетерминальных символов, обнаружение которого представляет цель синтаксического разбора.) Затем следует определение date через три терминальных символа. Точка с запятой в конце определения, отделяющая правила друг от друга, в примере отсутствует, т. к. правило здесь единственное. Лексический анализ сводится к выявлению чисел и строк с названиями месяцев, что задано следующей lex-спецификацией.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Листинг (v1.l) – Модуль на языке lex для синтаксического анализатора

Если на входе появится литера, не относящаяся к числам и названиям месяцев и не являющаяся разделителем (пробелом, табуляцией или новой строкой), функция yylex вернет ноль — признак конца ввода для yacc-модуля. Имена NUMBER и MONTH — это константы из файла y.tab.h, полученного в результате трансляции v1.y.

В отличие от примеров на тему lex, в данном случае функция yylex, обнаружив лексему, сразу возвращает ее код — он обрабатывается в вызывающей функции yyparse.

Для получения исполняемой программы был вызван сценарий build.sh. В нем задан вызов lex для всех файлов с расширением -l из текущего каталога (у нас один — v1.l), вызов yacc для модулей с расширением -y и, наконец, вызов cc для всех модулей на языке C, а именно: v1.c (результат трансляции v1.l), y.tab.c (результат трансляции v1.y) и zz.c. Последний играет ту же роль, что yy.c в примерах на тему lex: он содержит определение функции main, которая вызывает функцию синтаксического разбора yyparse. Также в zz.c определена функция yyerror(char \*) — функция yyparse вызовет ее при синтаксической ошибке, с указателем на строку "syntax error". Здесь же определена переменная yydebug, для включения режима отладки.

Для проверки правильности работы программы были написаны следующие тесты:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, число, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рис. 1 – Текстовые файлы с тестами для v1.y

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 2 – Результаты тестирования для v1.y

Программа v1.y была успешно протестирована. Результаты испытаний соответствуют спецификации.

Для наглядности можно продемонстрировать работу программы в отладочном режиме. Для этого в файле build.sh была заменена строка **lex –s \*.l** на строку **lex –d –s \*.l** — и протестируем программу.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 3 – Результат работы программы в режиме отладки

В отладочном режиме в выходной поток выводится трассировка работы лексического анализатора. Как только срабатывает какой-то шаблон (совпало содержание входной цепочки с описанным в шаблоне), то выводится и сама эта цепочка в выходной поток, и выводится номер строки программы, которой соответствует этот шаблон.

В модуле zz.c можно включить режим трассировки, т. е. вывод правил, применяемых при разборе. Для этого нужно в определении переменной yydebug исправить 0 на 1 и заново выполнить компиляцию.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис.4 – файл zz.c

Далее демонстрируется вывод для программы в режиме трассировки, на вход программе были даны тестовые последовательности из файла test.in.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рис. 5 – Результат работы в режиме трассировки

В сообщениях трассировки shift означает продолжение разбора с переходом в другое состояние, а reduce — свертку последовательности символов, замену ее одним символом в результате применения правила. В ходе разбора автомат меняет свои состояния (state); возможные состояния перечислены в файле y.output.

Из заголовочного файла y.tab.h видно, что коды терминальных символов, определенных при помощи ключевого слова %token, начинаются с 257. Код 0 зарезервирован для признака конца ввода, а коды от 1 до 256 — для литеральных лексем, или "литералов''.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 6 – Фрагмент файла y.tab.h

Использование литералов иллюстрируется примером из каталога \_date/v2. В определении date появилась запятая в одиночных кавычках — это и есть литерал, то есть терминальный символ, код которого равен ASCII-коду запятой.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 7 (v2.y) – Литерал в определении нетерминального символа

\_date/v2

В предыдущем примере функция yylex лексического анализатора при чтении запятой возвращала результат 0. Теперь в lex-модуль добавлено правило, которое в этом случае возвращает код запятой.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 8 (v2.l) – Передача литерала из lex-модуля

Для проверки правильности работы программы были написаны следующие тесты:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рис. 9 – Файлы с тестами для v2.y

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 10 - Результат тестирования для v2.y

Программа v2.y была успешно протестирована. Результаты испытаний соответствуют спецификации.

Изменим программу так, чтобы можно было бы использовать запятую и точку с запятой.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рис. 11 – Модифицированный файл v2.y

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рис. 12 – Модифицированный файл v2.l

Для проверки правильности работы программы были использованы те же тестовые последовательности, что и для тестирования программы v2.y.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 13 – Результаты тестирования модифицированного v2.y

Программа

была успешно протестирована. Результаты испытаний соответствуют спецификации.

\_date/v3

Если бы лексический анализатор вычислял величины месяцев и чисел и передавал их вместе с кодом лексемы, то синтаксический анализатор мог бы выводить дату и проверять ее допустимость. В примере \_date/v3 эти возможности использованы.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 14 (v3.l) – Задание типа и величины сопутствующего значения

Здесь добавлено определение типа сопутствующего значения YYSTYPE и ссылка на внешнюю переменную yylval. Лексеме NUMBER сопутствует значение десятичного числа, а лексеме MONTH — номер месяца в диапазоне [0..11].

Синтаксический анализатор использует сопутствующие значения следующим образом. Когда yylex возвращает управление yyparse, величина yylval записывается в стек значений; так продолжается, пока правило не будет применено. Доступ к этим значениям открыт через псевдопеременные $n. В начале кадра стека оставлено место для сопутствующего значения определяемого символа (псевдопеременная $$).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис.15 (v3a.y) – Доступ к семантическим знаечниям

Семантическое значение первого символа доступно через $1 — это номер месяца от 0 до 11, а $2 и $4 — значения дня и года. Литерал ',' в третьей позиции тоже считается символом; у него тоже есть значение, доступное через $3 — но там сейчас случайная величина, так как функция yylex, обнаружив запятую, в yylval ничего не записала.

Для проверки правильности работы программы были написаны следующие тесты:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рис. 16 – Файлы с тестами для v3a.y

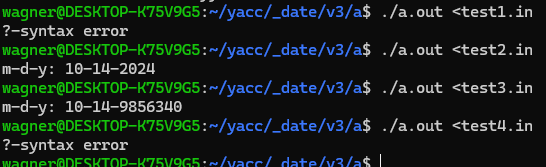


Рис. 17 – Результаты тестирования для v3a.y

Программа v3a.y была успешно протестирована. Результаты испытаний соответствуют спецификации.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 18 (v3b.y) – Проверка даты и вывод количества дней от 1970г.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рис. 19 - Файл abs\_state.c

Проверка даты и вычисление количества дней, прошедших от 01/01/1970 выполняется в функции abs\_date (см. модуль abs\_date.c) при помощи библиотечной функции mktime. Для проверки даты пригодилось умение mktime исправлять неправильную дату, хотя POSIX не рекомендует так с ней обращаться.

Для проверки правильности работы программы были написаны следующие тесты:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рис. 20 – Файлы с тестами для v3b.y

Изображение выглядит как текст, снимок экрана, меню, дизайн

Автоматически созданное описание

Рис. 21 – Результат тестирования v3b.y

Программа v3b.y была протестирована, но результаты испытаний не всегда соответствуют спецификации, это может быть связано с использованием функции mktime.

Значение числа дней можно было бы использовать в качестве сопутствующего значения для символа date.

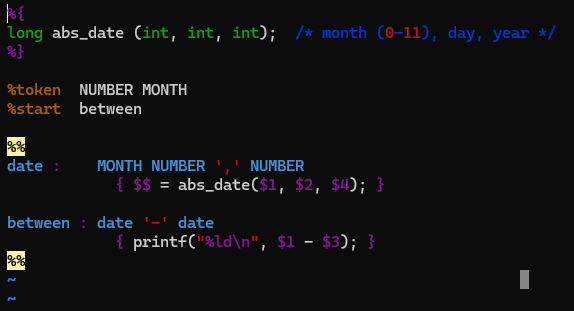
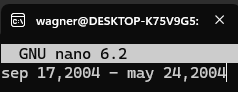
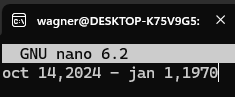


Рис. 22 (v3c.y) – Семантическое значение date и вычисление разницы между датами

Семантическое значение date формируется в конце правила для date и используется в правиле для between. Семантическое значение between не формируется за ненадобностью.

Величина $$ изначально равна величине $1; можно считать, что присвоение $$ = $1 — это действие по умолчанию.

Для проверки правильности работы программы были написаны следующие тесты:

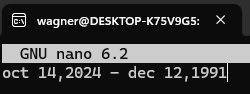
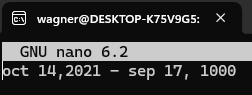
 

Рис. 23 – Текстовые файлы с тестами для v3c.y

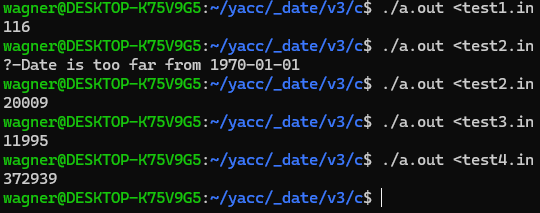


Рис. 24 – Результат тестирования для v3c.y

Программа v3c.y была протестирована, но результаты испытаний не всегда соответствуют спецификации, это может быть связано с использованием функции mktime.

Пример в каталоге \_date/v3/c некорректный в том смысле, что тип у сопутствующих значений — int, а у функции abs\_date — long. Поэтому при присвоении $$ = abs\_date(...) отбрасывается старшая часть результата.

\_date/v4

Иногда требуется возвращать сопутствующие значения разных типов, например, int и char\*, притом что канал передачи значений от yylex к yyparse единственный — переменная yylval. В этом случае используется объединение (union). Рассмотрим примеры из \_date/v4.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис.25 – (v4a.y) – Определение сопутствующего значения нескольких типов

В lex-модуле мы обращаемся к yylval как к варианту union в языке C.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 26 (v4.l) – Формирование сопутствующего значения в lex модуле

В результате использования %union определение YYSTYPE (в форме C-объединения) попадает в заголовочный файл y.tab.h. Теперь это определение не нужно дублировать в lex-модуле, достаточно директивы #include "t.tab.h".

При формировании указателя строки использована библиотечная функция strdup, копирующая содержимое yytext в динамическую память. Передача ссылки непосредственно на yytext (yylval.text = &yytext[0]) была бы ошибкой, т. к. к моменту использования этой ссылки (функцией print) содержимое yytext уже изменится — там будут цифры.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 27 – Тестирование модуля v4a.l

Трансляция yacc-модуля не прошла, поскольку в нем не задана информация о типе $1, $2 и $4 — ведь теперь у сопутствующего значения не один тип, а два. Тип можно указать при обращении к $-переменной.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 28 (v4b.y) – Явное указание типа при обращении в $-переменной

Для проверки правильности работы программы были написаны следующие тесты:

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рис. 29 – Файлы с тестами для v4b.y

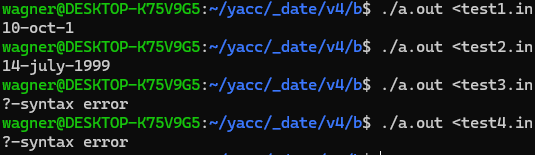


Рис. 30 – Результат тестирования для v4b.y

Программа v4b.y была успешно протестирована. Результаты испытаний соответствуют спецификации.

Тип может быть указан и при объявлении терминального символа, тогда при обращении к $-переменным уточнять его не придется. Этот вариант — предпочтительный.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис. 31 (v4c.y) – Задание типа при объявлении символа

Для проверки правильности работы программы были использованы те же тестовые последовательности, что и для тестирования программы v4b.y.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 32 – Результат тестирования для v4c.y

Программа v4c.y была успешно протестирована. Результаты испытаний соответствуют спецификации.

\_date/v5

Вернемся к примеру, где подсчитывается количество дней между двумя датами. В нем сопутствующие значения должны быть двух типов:

* int — для месяца, дня и года;
* long — для нетерминального символа date (количество дней от 01/01/1970).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 33 (v5.y) – Вычисление количества дней между двумя датами

Для проверки правильности работы программы были использованы те же тестовые последовательности, что и для тестирования программы v3c.y.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 34 – Результат тестирования для v5.y

Программа v5.y была протестирована, но результаты испытаний не всегда соответствуют спецификации, это может быть связано с использованием функции mktime.

list/v0

Если некая входная последовательность может быть распознана сразу несколькими шаблонами, то набор правил двусмысленный. Транслятор yacc в этих случаях выводит предупреждение:

* shift/reduce conflict — выбор между применением правила (reduce) и продолжением разбора (shift) в соответствии с другим правилом.
* reduce/reduce conflict — выбор между применением нескольких правил.

Правило выбирается по схеме, напоминающей ту, что принята в lex:

* предпочтение отдается соответствию большей длины, т. е. столкновение shift/reduce разрешается в пользу shift.
* если одна и та же последовательность соответствует нескольким правилам (конфликт reduce/reduce), предпочтение отдается тому правилу, которое задано раньше других.

Обратимся к программе в каталоге list/v0. Она разбирает список чисел, разделенных запятыми, и выводит число элементов в списке.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 35 (c1.l) – Лексический анализатор для разбора списка чисел

Этот лексический анализатор распознает цепочки десятичных цифр, вычисляет (при помощи библиотечной функции atoi) соответствующие им числовые значения, возвращая их синтаксическому анализатору через переменную yylval вместе с лексемой NUM. Все прочие литеры лексический анализатор возвращает в yacc-модуль в виде литералов.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рис. 36 (c1.y) – Синтаксический анализатор для разбора списка чисел

На вход этой программы подадим: 1,2,3<Enter>.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рис. 37 – Результат тестирования c1.y

Получено сообщение ?-syntax error. Чтобы выяснить причину, включим трассировку: найдём в zz.c определение yydebug и исправим 0 на 1, затем повторим трансляцию (в вызове yacc задайте ключи –vtd).

При запуске исполняемой программы с тем же вводом (1,2,3<Enter>) получим трассу:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 38 – Трассировка обработки для c1.y

В каждой строке, пока не появилась ошибка, показан номер состояния конечного автомата при синтаксическом разборе. Что значат эти номера и состояния, можно выяснить в файле y.output, полученном при трансляции yacc-модуля. Ниже приведен фрагмент файла y.output для рассматриваемого примера.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис. 39 –фрагмент файла y.output

В разделе Grammar перечислены правила из yacc-модуля. Дальше идет описание терминалов и нетерминалов и описание состояний. Работа автомата начинается из состояния 0. В каждом состоянии у автомата могут быть, в общем случае, несколько альтернативных целей, и выбор зависит от очередного символа.

Текущий пункт на пути к цели отмечается точкой. Например, в состоянии 0 автомат должен получить либо символ \_\_list согласно первой строке раздела State 0.

После целей перечислены ожидаемые (допустимые) символы и реакция на них. Так, запись во второй строке означает: при получении лексемы NUM перейти в состояние 1. Слово shift означает переключение состояния с накоплением данных в стеке. Действительно, одно число рано считать списком — за ним могут следовать, через запятую, другие числа.

Операции go to переключают состояние без накопления данных.

Операция reduce означает применение правила, с удалением данных из стека. Например, согласно третьей строке, конец ввода в состоянии 0 приведет к применению правила 2. Это правило, согласно третьей строке раздела Grammar, относится к пустому списку.

Вернемся к трассе программы при вводе 1,2,3<Enter>. Читаем: в состоянии 0 получена лексема NUM; в результате перешли в состояние 1. Дальше, в состоянии 1 получили ',' и т. д. — до получения неопределённой лексемы ($undefined) – это был литерал '\n', недопустимый в состоянии 1.

Литерал '\n' пришел из lex-модуля. Исправить ситуацию можно двумя способами.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 41 (c2.l) – Включение ‘\n’ в лексический разбор

Пример тестирования программы, собранной из сочетания: c2.l + с2.y.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 42 – Тестирование программы c2.l с2.y

Теперь при переносе строки программа продолжает свою работу, но не выводит для неё результат.

Пример тестирования программы, собранной из сочетания: c1.l + с2.y.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 43 – Тестирование программы c1.l с2.y

Теперь при первом переносе строки программа продолжает выводит правильный результат, но при дальнейшем появлении символа переноса строки программа сообщает об ошибке.

Пример тестирования программы, собранной из сочетания: c2.l + с1.y.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 44 – Тестирование программы c2.l с1.y

В таком сочетании программа работает верно. Теперь при переносе строки программа продолжает свою работу, которую можно остановить только при помощи <Ctrl+D>, после чего будет выведен результат.

Теперь выясним, как программа реагирует на разделители. Подайте на вход список чисел с пробелами: 1, 2, 5<Enter>.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

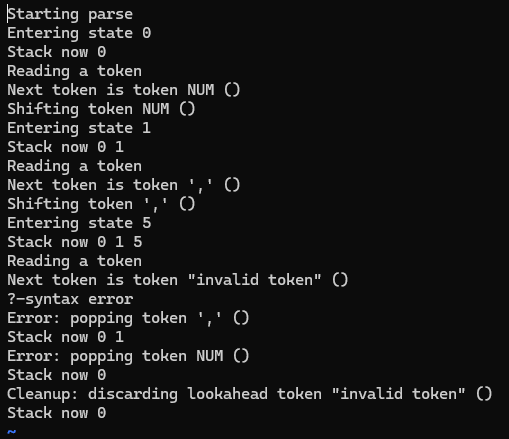


Рис. 45 – Трейсинг программы c2.l c1.y

Читаем: в состоянии 0 получена лексема NUM; в результате перешли в состояние 1. Дальше, в состоянии 1 получили ',' затем была получена неопределённая лексема ($undefined) – это был литерал ' ' (пробел), недопустимый в состоянии 1.

Фильтрацию пробелов и табуляций имеет смысл выполнять в lex-модуле.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 46 (c3.l) – Удаление разделителей при лексическом разборе

Теперь программа работает верно не только при наличии символа переноса строки, но и при наличии символа пробела и табуляции. Программа продолжает свою работу до тех пор пока не будет остановлена при помощи <Ctrl+D>, после чего будет выведен результат.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 47 – Тестирование программы c3.l с1.y

list/v1

В каталоге list/v1 к разбору списка добавлен вывод элементов.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 48 – Файл c.l

В описании непустого списка в c1.y используется правая рекурсия, а в c2.y — левая. При левой рекурсии применение правила откладывается до конца списка, что требует больше ресурсов и может привести к исчерпанию памяти. Убедитесь, что c1 и c2 выводят элементы списка в разном порядке.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 48 – Файл c1.y

Для проверки правильности работы программы был написан следующий тест:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рис.49 – Тестовый файл для c1.y

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рис. 50 – Результат тестирования для c1.y

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 51 – Файл c2.y

Для проверки правильности работы программы были использованы те же тестовые последовательности, что и для тестирования программы c1.y.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 52 – Результат тестирования для c2.y

В первом случае список выводится от конца к началу, во втором – от начала к концу.

Вывод:

В процессе выполнения лабораторной работы были исследованы примеры программ синтаксического анализа на языке Yacc, которые проверяли соответствие своей спецификации. Также была рассмотрена структура и синтаксис этих программ на языке Yacc, а также их взаимодействие с модулями лексического разбора на языке Lex. Программы были составлены в форме набора правил, часто рекурсивных, которые определяли нетерминальные символы (языковые конструкции) через терминальные символы (лексемы) и ранее определенные нетерминальные символы. Для некоторых правил были предусмотрены действия на языке C. Кроме того, программа на языке Lex вызывалась из программы на Yacc для чтения лексем из входного потока.