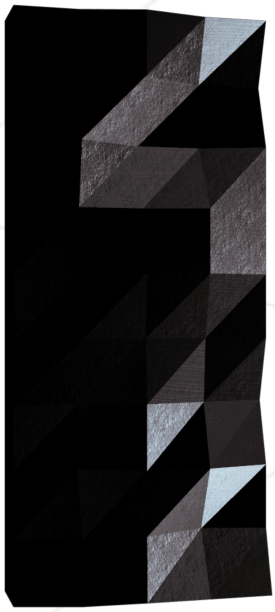
β

This project is currently under beta-test, until it is validated by a fair number of students in the 42 community. Please report any typo, incoherence, inconsistency, error, using the form https://tally.so/r/3lVDJo



ft_yacc

easy parsing

Summary: Now that you have a lexer generator, it's time to make a parser generator.

Version: 1.00

Contents

Chapter I

Forewords

3 A logical picture of facts is a thought.

3.001 'A state of affairs is thinkable': what this means is that we can picture it to ourselves.

3.01 The totality of true thoughts is a picture of the world.

3.02 A thought contains the possibility of the situation of which it is the thought. What is thinkable is possible too.

3.03 Thought can never be of anything illogical, since, if it were, we should have to think illogically.

3.031 It used to be said that God could create anything except what would be contrary to the laws of logic.The truth is that we could not say what an 'illogical' world would look like.

3.032 It is as impossible to represent in language anything that 'contradicts logic' as it is in geometry to represent by its coordinates a figure that contradicts the laws of space, or to give the coordinates of a point that does not exist.

3.0321 Though a state of affairs that would contravene the laws of physics can be represented by us spatially, one that would contravene the laws of geometry cannot.

3.04 If a thought were correct a priori, it would be a thought whose possibility ensured its truth.

3.05 A priori knowledge that a thought was true would be possible only if its truth were recognizable from the thought itself (without anything to compare it with).

3.1 In a proposition a thought finds an expression that can be perceived by the senses.

3.11 We use the perceptible sign of a proposition (spoken or written, etc.) as a projection of a possible situation. The method of projection is to think of the sense of the proposition.

3.12 I call the sign with which we express a thought a propositional sign.And a proposition is a propositional sign in its projective relation to the world.

3.13 A proposition, therefore, does not actually contain its sense, but does contain the possibility of expressing it. ('The content of a proposition' means the content of a proposition that has sense.) A proposition contains the form, but not the content, of its sense.

3.14 What constitutes a propositional sign is that in its elements (the words) stand in a determinate relation to one another. A propositional sign is a fact.

3.141 A proposition is not a blend of words.(Just as a theme in music is not a blend of notes.) A proposition is articulate.

3.142 Only facts can express a sense, a set of names cannot.

3.143 Although a propositional sign is a fact, this is obscured by the usual form of expression in writing or print. For in a printed proposition, for example, no essential difference is apparent between a propositional sign and a word. (That is what made it possible for Frege to call a proposition a composite name.)

3.1431 The essence of a propositional sign is very clearly seen if we imagine one composed of spatial objects (such as tables, chairs, and books) instead of written signs.

3.1432 Instead of 'The complex sign "aRb" says that a stands to b in the relation R', we ought to put, 'That "a" stands to "b" in a certain relation says that aRb.'

3.144 Situations can be described but not given names.

3.2 In a proposition a thought can be expressed in such a way that elements of the propositional sign correspond to the objects of the thought.

3.201 I call such elements 'simple signs', and such a proposition 'complete analysed'.

3.202 The simple signs employed in propositions are called names.

3.203 A name means an object. The object is its meaning. ('A' is the same sign as 'A'.)

3.21 The configuration of objects in a situation corresponds to the configuration of simple signs in the propositional sign.

3.221 Objects can only be named. Signs are their representatives. I can only speak about them: I cannot put them into words. Propositions can only say how things are, not what they are.

3.23 The requirement that simple signs be possible is the requirement that sense be determinate.

3.24 A proposition about a complex stands in an internal relation to a proposition about

a constituent of the complex. A complex can be given only by its description, which will be right or wrong. A proposition that mentions a complex will not be nonsensical if the complex does not exist, but simply false. When a propositional element signifies a complex, this can be seen from an indeterminacy in the propositions in which it occurs. In such cases, we know that the proposition leaves something undetermined. (In fact, the notation for generality contains a prototype.) The contraction of a symbol for a complex into a simple symbol can be expressed in a definition.

3.25 A proposition cannot be dissected any further by means of a definition: it is a primitive sign.

3.261 Every sign that has a definition signifies via the signs that serve to define it; and the definitions point the way. Two signs cannot signify in the same manner if one is primitive and the other is defined by means of primitive signs. Names cannot be anatomized by means of definitions. (Nor can any sign that has a meaning independently and on its own.)

3.262 What signs fail to express, their application shows. What signs slur over, their application says clearly.

3.263 The meanings of primitive signs can be explained by means of elucidations. Elucidations are propositions that stood if the meanings of those signs are already known.

3.3 Only propositions have sense; only in the nexus of a proposition does a name have meaning.

3.31 I call any part of a proposition that characterizes its sense an expression (or a symbol). (A proposition is itself an expression.) Everything essential to their sense that propositions can have in common with one another is an expression. An expression is the mark of a form and a content.

3.311 An expression presupposes the forms of all the propositions in which it can occur. It is the common characteristic mark of a class of propositions.

3.312 It is therefore presented by means of the general form of the propositions that it characterizes. In fact, in this form the expression will be constant and everything else variable.

3.313 Thus an expression is presented by means of a variable whose values are the propositions that contain the expression. (In the limiting case the variable becomes a constant, the expression becomes a proposition.) I call such a variable a 'propositional variable'.

3.314 An expression has meaning only in a proposition. All variables can be construed as propositional variables. (Even variable names.)

3.315 If we turn a constituent of a proposition into a variable, there is a class of propositions all of which are values of the resulting variable proposition. In general, this class too will be dependent on the meaning that our arbitrary conventions have given to parts of the original proposition. But if all the signs in it that have arbitrarily determined

meanings are turned into variables, we shall still get a class of this kind. This one, however, is not dependent on any convention, but solely on the nature of the pro position. It corresponds to a logical form–a logical prototype.

3.316 What values a propositional variable may take is something that is stipulated. The stipulation of values is the variable.

3.317 To stipulate values for a propositional variable is to give the propositions whose common characteristic the variable is. The stipulation is a description of those propositions. The stipulation will therefore be concerned only with symbols, not with their meaning. And the only thing essential to the stipulation is that it is merely a description of symbols and states nothing about what is signified. How the description of the propositions is produced is not essential.

3.318 Like Frege and Russell I construe a proposition as a function of the expressions contained in it.

3.32 A sign is what can be perceived of a symbol.

3.321 So one and the same sign (written or spoken, etc.) can be common to two different symbols–in which case they will signify in different ways.

3.322 Our use of the same sign to signify two different objects can never indicate a common characteristic of the two, if we use it with two different modes of signification. For the sign, of course, is arbitrary. So we could choose two different signs instead, and then what would be left in common on the signifying side?

3.323 In everyday language it very frequently happens that the same word has different modes of signification–and so belongs to different symbols– or that two words that have different modes of signification are employed in propositions in what is superficially the same way. Thus the word 'is' figures as the copula, as a sign for identity, and as an expression for existence; 'exist' figures as an intransitive verb like 'go', and 'identical' as an adjective; we speak of something, but also of something's happening. (In the proposition, 'Green is green'–where the first word is the proper name of a person and the last an adjective–these words do not merely have different meanings: they are different symbols.)

3.324 In this way the most fundamental confusions are easily produced (the whole of philosophy is full of them).

3.325 In order to avoid such errors we must make use of a sign-language that excludes them by not using the same sign for different symbols and by not using in a superficially similar way signs that have different modes of signification: that is to say, a sign-language that is governed by logical grammar–by logical syntax. (The conceptual notation of Frege and Russell is such a language, though, it is true, it fails to exclude all mistakes.)

3.326 In order to recognize a symbol by its sign we must observe how it is used with a sense.

3.327 A sign does not determine a logical form unless it is taken together with its logico-syntactical employment.

3.328 If a sign is useless, it is meaningless. That is the point of Occam's maxim. (If everything behaves as if a sign had meaning, then it does have meaning.)

3.33 In logical syntax the meaning of a sign should never play a role. It must be possible to establish logical syntax without mentioning the meaning of a sign: only the description of expressions may be presupposed.

3.331 From this observation we turn to Russell's 'theory of types'. It can be seen that Russell must be wrong, because he had to mention the meaning of signs when establishing the rules for them.

3.332 No proposition can make a statement about itself, because a propositional sign cannot be contained in itself (that is the whole of the 'theory of types').

3.333 The reason why a function cannot be its own argument is that the sign for a function already contains the prototype of its argument, and it cannot contain itself. For let us suppose that the function $F(fx)$ could be its own argument: in that case there would be a proposition '$F(F(fx))$', in which the outer function F and the inner function F must have different meanings, since the inner one has the form $\phi(f(x))$ and the outer one has the form $\psi(\phi(fx))$. Only the letter 'F' is common to the two functions, but the letter by itself signifies nothing. This immediately becomes clear if instead of '$F(Fu)$' we write '$(\exists\phi) : F(\phi u).\phi u = Fu$'. That disposes of Russell's paradox.

3.334 The rules of logical syntax must go without saying, once we know how each individual sign signifies.

3.34 A proposition possesses essential and accidental features. Accidental features are those that result from the particular way in which the propositional sign is produced. Essential features are those without which the proposition could not express its sense.

3.341 So what is essential in a proposition is what all propositions that can express the same sense have in common. And similarly, in general, what is essential in a symbol is what all symbols that can serve the same purpose have in common.

3.3411 So one could say that the real name of an object was what all symbols that signified it had in common. Thus, one by one, all kinds of composition would prove to be unessential to a name.

3.342 Although there is something arbitrary in our notations, this much is not arbitrary– that when we have determined one thing arbitrarily, something else is necessarily the case. (This derives from the essence of notation.)

3.3421 A particular mode of signifying may be unimportant but it is always important that it is a possible mode of signifying. And that is generally so in philosophy: again and again the individual case turns out to be unimportant, but the possibility of each

individual case discloses something about the essence of the world.

3.343 Definitions are rules for translating from one language into another. Any correct sign-language must be translatable into any other in accordance with such rules: it is this that they all have in common.

3.344 What signifies in a symbol is what is common to all the symbols that the rules of logical syntax allow us to substitute for it.

3.3441 For instance, we can express what is common to all notations for truth-functions in the following way: they have in common that, for example, the notation that uses '$\sim p$' ('*not* p') and '$p \vee g$' ('p *or* g') can be substituted for any of them. (This serves to characterize the way in which something general can be disclosed by the possibility of a specific notation.)

3.3442 Nor does analysis resolve the sign for a complex in an arbitrary way, so that it would have a different resolution every time that it was incorporated in a different proposition.

3.4 A proposition determines a place in logical space. The existence of this logical place is guaranteed by the mere existence of the constituents– by the existence of the proposition with a sense.

3.41 The propositional sign with logical co-ordinates–that is the logical place.

3.411 In geometry and logic alike a place is a possibility: something can exist in it.

3.42 A proposition can determine only one place in logical space: nevertheless the whole of logical space must already be given by it. (Otherwise negation, logical sum, logical product, etc., would introduce more and more new elements in co-ordination.) (The logical scaffolding surrounding a picture determines logical space. The force of a proposition reaches through the whole of logical space.)

3.5 A propositional sign, applied and thought out, is a thought.

Ludwig Wittgenstein *Tractatus Logico-Philosophicus*

Chapter II

Objectives

In this project, you will implement the POSIX yacc utility.

During this project you will learn about automata theory, formal language theory, Chomsky hierarchy, dragons, and a lot of weird mathematical symbols and Greek letters.

II.1 Description

Yacc is a program that generates parsers.

It takes a .y file (a yacc-specific format) as input and outputs a y.tab.c file containing the C source code that represents the parser.

Syntactic analysis is the process of recognizing grammatical structures in a sequence of symbols (or tokens) generated by a lexer in order, for example, to create data structure that shows the syntactic relationships between them (most of the time it is either a parse tree (concrete syntax tree) or an abstract syntax tree).

Yacc was created alongside lex to work specifically with it. If you are smart, you will use these two programs (your implementations, of course) for parsing in your cc1.

II.2 Example

```
$> cat calc.l
%pointer
%{
#include <unistd.h>
#include "y.tab.h"
%}

%%

[0-9]+ {
    yylval = atoi(yytext);
    return NUMBER;
}


[[:blank:]] ; // skip blanks

.|\n return *yytext; // literals
$> cat calc.y

%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int  yyerror(const char *s);
%}

%token NUMBER

%start PROG

%left '-' '/' '+' '*'

%%

PROG : EXPR
     | EXPR '\n' PROG
     |
;

EXPR : ADDITION { printf("= %d\n", $1);}
     | error
;

ADDITION : ADDITION '+' ADDITION {$$ = $1 + $3;}
         | ADDITION '-' ADDITION {$$ = $1 - $3;}
         | PRODUCT
;

PRODUCT : PRODUCT '*' PRODUCT {$$ = $1 * $3;}
        | PRODUCT '/' PRODUCT {$$ = $1 / $3;}
        | '(' ADDITION ')' { $$ = $2; }
        | NUMBER
;

%%
$> lex calc.l && yacc -d calc.y && cc lex.yy.c y.tab.c -ly -ll -o calc
$> echo "42+1337+(21*19)" | ./calc
= 1778
```

Chapter III

General rules

- This project will only be evaluated by actual human beings. You are therefore free to organize and name your files as you wish, although you need to respect some requirements listed below.
- You are allowed to write your program in any of the following languages:
  - C
  - C++
  - Java
  - Zig
  - Rust
  - Caml/OCaml
  - Any **purely functional** language
  - assembly (please don't)
- In all cases, you are only allowed to use the standard library.
- The executable file must be named `ft_yacc`.
- The liby must be named liby.
- If you use a compiled language, you must submit a Makefile or a configuration for the native build system of your language.
- All the code present on the repo must be yours.
- If you use your own ft_lex you must include the sources as a subdirectory and your makefile or build config must compile it.
- You are never allowed to include generated code in your repo; if you use ft_lex, you must commit the .l files and your makefile or build config must call the generators automatically.
- You have to handle errors in a sensitive manner. In no way can your program quit in an unexpected manner (segmentation fault, bus error, double free, etc.).

- If a user error occurs (e.g., duplicated token definition in the .y file), your program must provide a detailed error message (eg: `parser.y:42 duplicated token definition "IDENTIFIER"`)

Chapter IV

Mandatory part

IV.1 The program

- You must implement the yacc utility as described by POSIX.1-2024 with all its options and functionalities.
- Therefore, you must also implement liby (a small utility library) as specified by POSIX and write a makefile for compiling it.
- You may choose any language from the list above, but the target language must be C.
- The output C file must contain a compiled **LALR(1)** parser.
- In the output file (and in the liby), you may only use the following functions:
  - all the functions from `<stdio.h>`
  - `malloc(3)`
  - `realloc(3)`
  - `calloc(3)`
  - `free(3)`

`Bison can be useful if you don't understand the documentation and need to experiment by yourself.`

IV.2 Testing

You must provide example .l and .y files, as well as files to parse that cover all functionalities of your ft_yacc.

Chapter V

Bonus part

V.1 Polyglotism

You've done the difficult part; now let's add another target language to your ft_yacc. This can be any language!
You can take inspiration from existing implementations (e.g., bison++, ocamlyacc, goyacc, kmyacc...) for the input file format, or invent a new one if you prefer.
You must add a flag to your program to switch the target language.

If you plan to realize cc1 later, think about the language that you will use to make it.

Obviously, if you create a C++ version, you can't simply reuse the C code be creative.

V.2 Generalisation

Add a flag (or configuration line) to enable handling of non-deterministic and ambiguous grammars. For this bonus, you must use an algorithm other than LALR.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double-check the names of your folders and files to ensure they are correct.