Sreecharan Vanam

Tokenization-and-Categorization-using-OCaml-A-Lexical-Analysis-Approach

Introduction

The most important phase in the compilation process is lexical analysis, which divides source code into tokens that the compiler or interpreter can utilize for additional processing. This work provides an OCaml-based Lexical Analyzer that can identify and classify tokens, including identifiers, operators, punctuation symbols, integer literals, and keywords.

Process steps:

Tokenization: Tokenization involves dividing the input text into discrete tokens, which include operators, keywords, punctuation, integer literals, identifiers, and unknown tokens.

Categorization: Based on its type, each token can be categorized as either a keyword, operator, punctuation symbol, integer literal, identifier, or unknown.

Printing: The tokens are organized and printed to the standard output, where each token is shown with its associated given category.

Explanation:

A token type which represents different kinds of tokens identified during lexical analysis is specified. Keywords (Keyword), operators (Operator), integer literals (IntLiteral), punctuation symbols (Punctuation), identifiers (Identifier), and unknown tokens (Unknown) are the given types.

The is_alnum function can be useful in identifying keywords and IDs since it detects if a given character is alphanumeric. In comparison, a character's status as a punctuation sign is identified by the is_punctuation function.

The lexical analyzer's key component is the tokenize function. It receives a string as input and separates it into tokens in accordance with the given criteria. The function handle operators, punctuation symbols, whitespace, and alphanumeric characters using secondary functions. It also attempts to use exception handling to convert alphanumeric substrings into integer literals.

The categorized tokens are printed to the standard output by the print_tokens function. It prints each token along with its category following looping over the list of tokens.

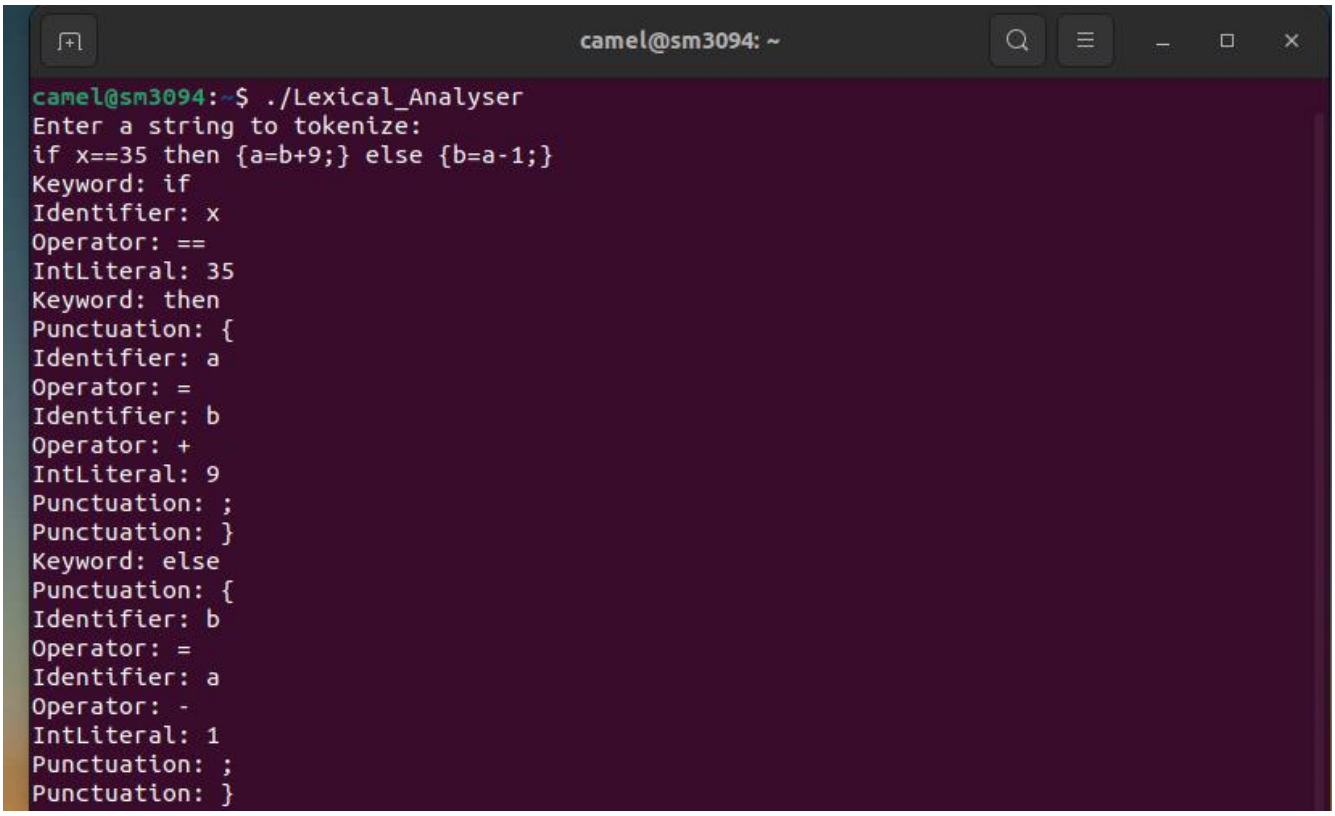
The main function asks the user to enter a string to be tokenized, reads the input string, utilizes the tokenize function to tokenize it, and then uses the print_tokens function to print the tokens that are generated.

Sreecharan Vanam

Code:

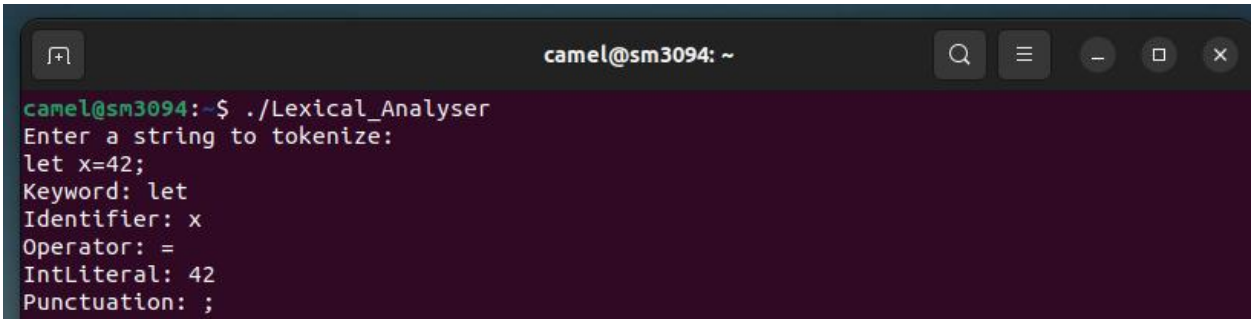
```
(* Lexical Analyzer in OCaml *)

(* Define types for tokens *)
type token =
  | Keyword of string
  | Operator of string
  | Punctuation of string
  | IntLiteral of int
  | Identifier of string
  | Unknown of string

(* Function to check if a character is alphanumeric *)
let is_alnum c =
  (c >= 'a' && c <= 'z') ||
  (c >= 'A' && c <= 'Z') ||
  (c >= '0' && c <= '9')

(* Function to check if a character is a punctuation symbol *)
let is_punctuation c =
  List.mem c ['('; ')'; '{'; '}'; ';']

(* Function to tokenize a string *)
let rec tokenize str =
  let rec consume_whitespace i =
    if i < String.length str && (str.[i] = ' ' || str.[i] = '\t') then
      consume_whitespace (i + 1)
    else
      i
  in
  let rec tokenize_helper i =
    if i >= String.length str then
      []
    else
      let next_char = str.[i] in
      if next_char = ' ' || next_char = '\t' then
        tokenize_helper (consume_whitespace i)
      else if is_alnum next_char then
        let j = ref i in
        while !j < String.length str && is_alnum str.[!j] do
          incr j
        done;
        let token_str = String.sub str i (!j - i) in
        if token_str = "if" || token_str = "else" || token_str = "for" || token_str = "if else" || token_str = "while" || token_str = "let" || token_str = "in" || token_str = "then" then
          Keyword token_str :: tokenize_helper !j
        else
          begin
            try
              IntLiteral (int_of_string token_str) :: tokenize_helper !j
            with Failure _ ->
              Identifier token_str :: tokenize_helper !j
          end
      else if is_punctuation next_char then
        Punctuation (Char.escaped next_char) :: tokenize_helper (i + 1)
      else if next_char = '+' || next_char = '-' || next_char = '*' || next_char = '/' || next_char = '=' || next_char = '!' then
        let lookahead = if i + 1 < String.length str then Some str.[i + 1] else None in
        (match lookahead with
         | Some '=' ->
             Operator (Char.escaped next_char ^ Char.escaped '=') :: tokenize_helper (i + 2)
         | _ ->
             Operator (Char.escaped next_char) :: tokenize_helper (i + 1))
      else
        Unknown (Char.escaped next_char) :: tokenize_helper (i + 1)
  in
  tokenize_helper 0

(* Function to print tokens *)
let print_tokens tokens =
  List.iter (fun token ->
    match token with
    | Keyword s -> Printf.printf "Keyword: %s\n" s
    | Operator s -> Printf.printf "Operator: %s\n" s
    | Punctuation s -> Printf.printf "Punctuation: %s\n" s
    | IntLiteral i -> Printf.printf "IntLiteral: %d\n" i
    | Identifier s -> Printf.printf "Identifier: %s\n" s
    | Unknown s -> Printf.printf "Unknown: %s\n" s
  ) tokens

(* Main function *)
let () =
  Printf.printf "Enter a string to tokenize:\n";
  let input_str = read_line () in
  let tokens = tokenize input_str in
  print_tokens tokens
```

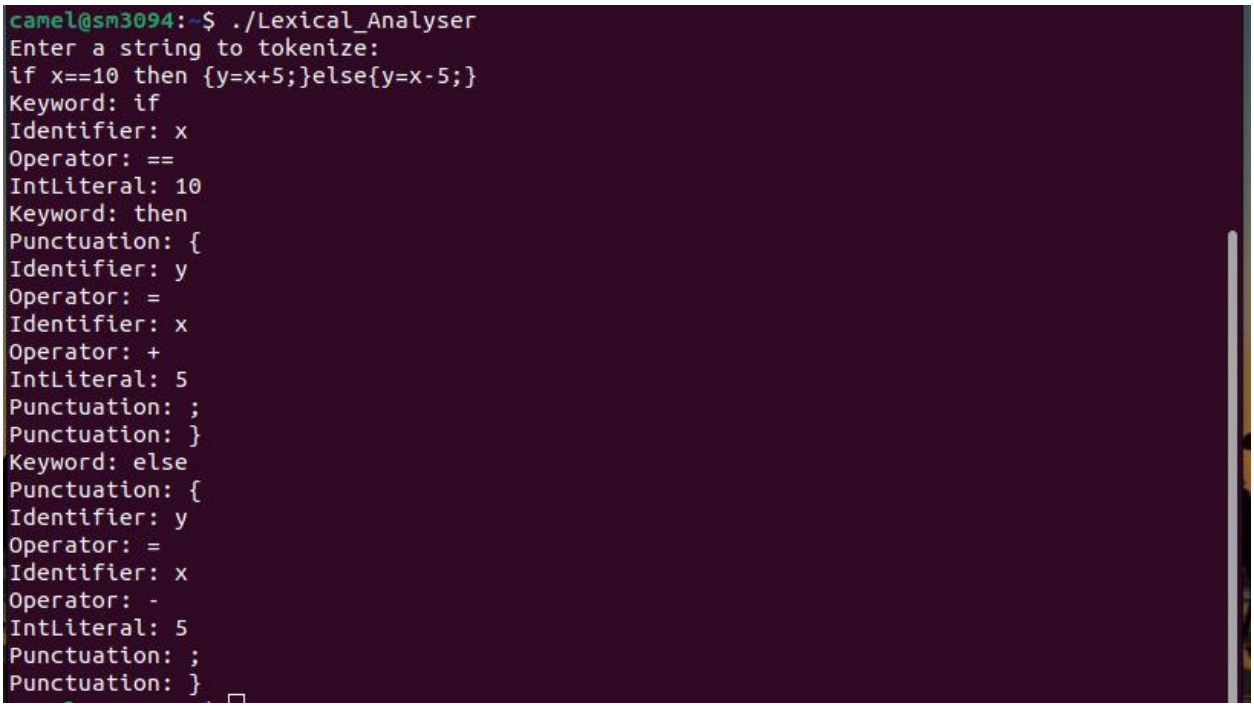
Test case 1:



Test case 2:

Random Test case:





References:

https://v2.ocaml.org/manual/lexyacc.html

https://caml.inria.fr/pub/old_caml_site/ocaml/htmlman/manual026.html