CSE341 HW1 Report

C-to-Lisp Converter

R.Hilal SAYGIN

Introduction

Developed for C to Lisp code converting task. It focuses on preserving the structure of C programs while converting them into valid Lisp code.
The converter handles key conversion steps

- if statements
- logical and arithmetical operation conversions from c to lisp
- for loops
- variable assignments
- variable definitions in nested structures
- function declarations
- function definitions
- function calls
- printf
- variable assignment by function return.

The code relies on `cl-ppcre` built-in Common Lisp functions for basic operations, lists manipulation, cond structures, regular expression-based string manipulation etc.

1. Line Type Detection and Handling

- **determine-line-type**: Analyses a line of C code and determines its type by using regex matching system (e.g., function definition, if-statement, variable assignment). Regular expressions and string operations are carefully used to ditinguisg between line types and categorise each line.
- **convert-line**: Based on the line type identified by `determine-line-type`, this function calls the appropriate conversion function. It also manages block tracking, indentation, and ensures that blocks like `if` statements, loops, and functions are correctly opened and closed. Conversion-foo functionality integrated in this function as well.

2. Custom String Manipulation

- Several helper functions like `string-contains`, `string-starts-with`, `string-trim-whitespace`, and `split-params` are implemented to handle common string operations used throughout the project.

3. C to Lisp Conversions for Specific Code Constructs

- **convert-if-statement**: Converts C-style `if` statements to Lisp's prefix-style conditional expressions. It handles the translation of comparison operators (e.g., `!=` to `/=`).

- **convert-condition:** C conditional expressions (e.g., ==, !=, &&, ||, !) into Lisp equivalents (=, /=, and, or, not) and rearranges them into Lisp's prefix notation.
- **convert-printf:** Converts C's printf statements to Lisp's format function, handling format specifiers and arguments.
- **convert-for-loop**: Transforms C `for` loops into Lisp `loop` constructs. The conversion handles initialization, condition checking, and incrementing by converting these into equivalent Lisp loop keywords.
- **convert-func-definition**: Converts C function definitions to Lisp's `defun` format. It extracts the function name, parameters, and handles block-level structures within the function.
- **convert-func-call**: Converts C function calls to their Lisp equivalents, with proper handling of arguments and formatting.
- **convert-func-prototype**: Converts C function prototypes into Lisp's `declaim` statements, which define function signatures.
- **convert-var-definition**: Converts C variable definitions into Lisp's `let` or `setf` bindings for handling variable declarations and initializations.
- **convert-arithmetic**: Translates C-style arithmetic expressions into equivalent Lisp expressions. It handles the conversion of infix arithmetic to Lisp's prefix notation.
- **Convert-func-return:** Converts C return statements into Lisp return expressions, handling any involved arithmetic or function calls.

4. File Handling

- **read-file**: Reads the contents of a C file line by line and returns them as a list of strings.
- **write-file**: Writes the converted Lisp code for each line to the given output file name.
- **convert-c-to-lisp**: The main entry point of the program. It reads the input C file, processes each line using `convert-c-to-lisp-recursive`, and writes the converted Lisp code to the output file.

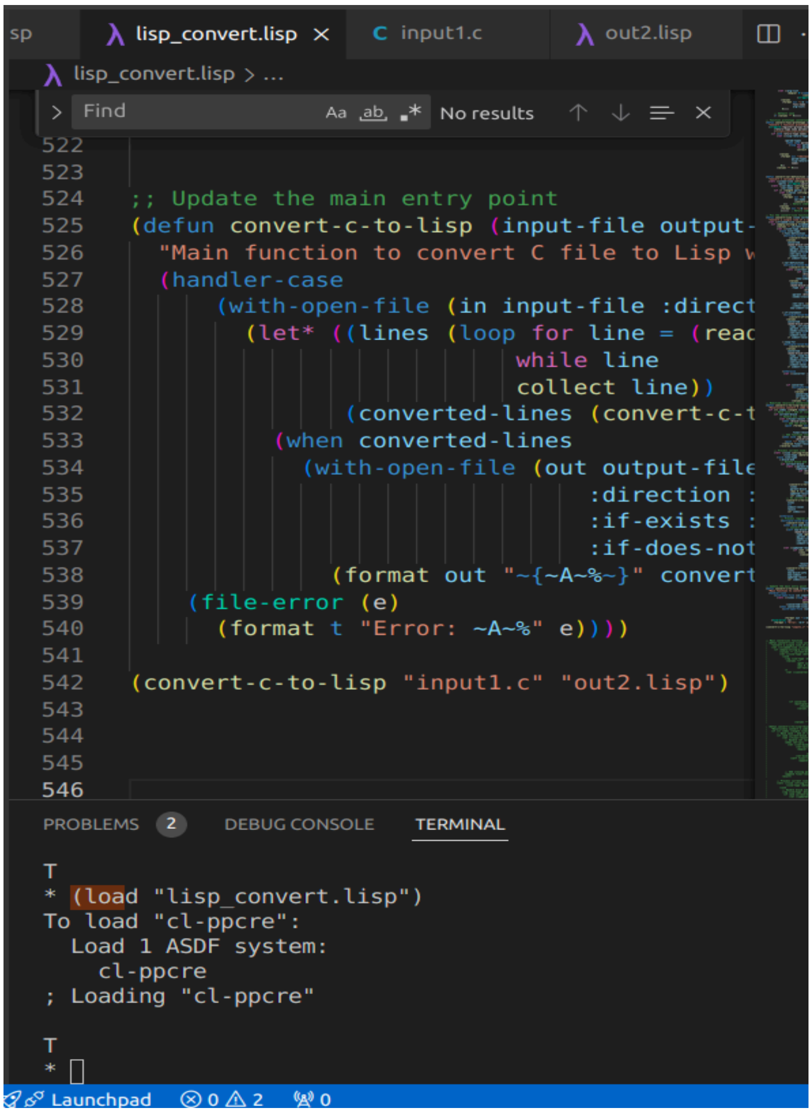
5. Recursive Code Conversion

- **convert-c-to-lisp-recursive**: The main recursive conversion logic. It implements the recursive logic carefully to process each line of C code and convert it to Lisp.
  It keeps track of blocks (e.g., `if`, `for`, `function`) and their associated indentation levels. As blocks are opened and closed, the indentation and structure of the code are adjusted accordingly.

6. Block Information Management

- **block-info**: A custom structure that holds metadata about blocks of code (e.g., functions, loops). It tracks whether a block needs to be closed, the variables declared in the block, and the current indentation level.

Results



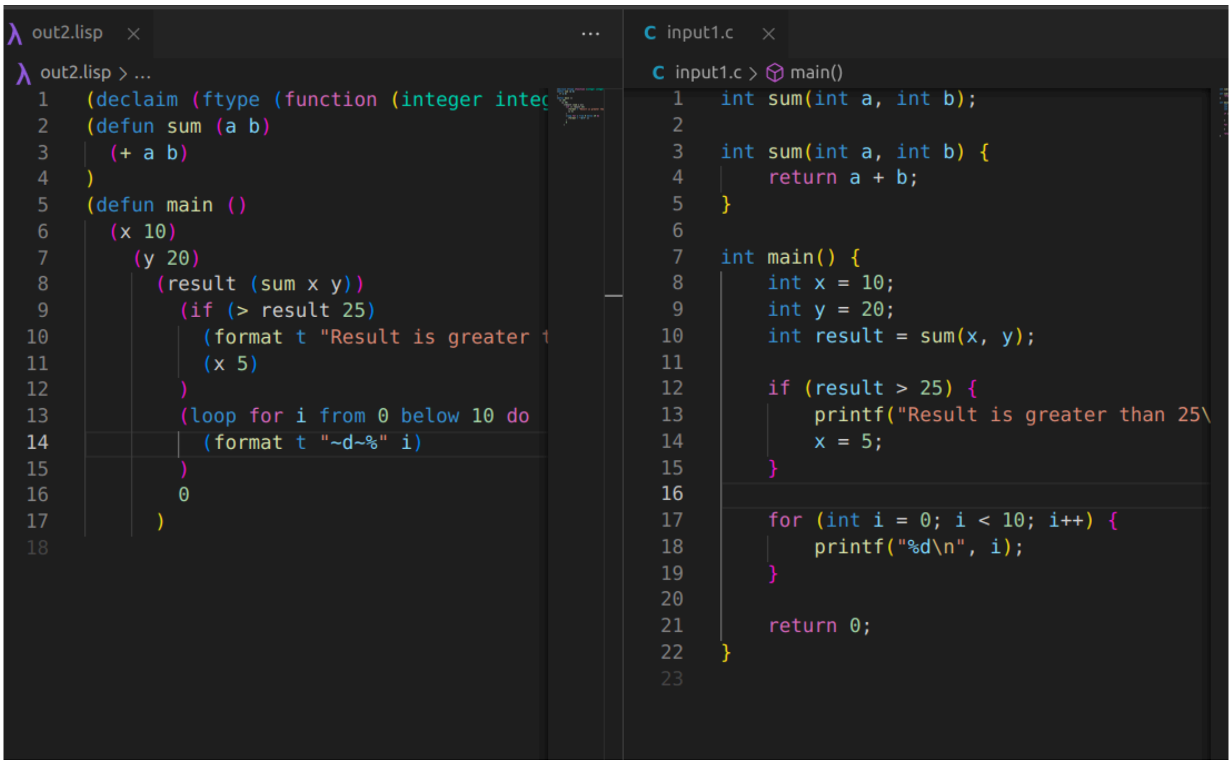
```lisp

;; Update the main entry point
(defun convert-c-to-lisp (input-file output-
  "Main function to convert C file to Lisp w
  (handler-case
      (with-open-file (in input-file :direct
        (let* ((lines (loop for line = (reac
                            while line
                            collect line))
               (converted-lines (convert-c-t
          (when converted-lines
            (with-open-file (out output-file
                                 :direction :
                                 :if-exists :
                                 :if-does-not
              (format out "~{~A~%~}" convert
    (file-error (e)
      (format t "Error: ~A~%" e))))

(convert-c-to-lisp "input1.c" "out2.lisp")
```

```
T
* (load "lisp_convert.lisp")
To load "cl-ppcre":
  Load 1 ASDF system:
    cl-ppcre
; Loading "cl-ppcre"

T
*
```



```lisp
(declaim (ftype (function (integer integ
(defun sum (a b)
  (+ a b)
)
(defun main ()
  (x 10)
    (y 20)
      (result (sum x y))
        (if (> result 25)
          (format t "Result is greater t
          (x 5)
        )
        (loop for i from 0 below 10 do
          (format t "~d~%" i)
        )
        0
      )
```

```c
int sum(int a, int b);

int sum(int a, int b) {
    return a + b;
}

int main() {
    int x = 10;
    int y = 20;
    int result = sum(x, y);

    if (result > 25) {
        printf("Result is greater than 25\
        x = 5;
    }

    for (int i = 0; i < 10; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```