main

line-Type:
Function takes a line of C code and determines its type by matching it against regular expressions. Returns a symbol representing the line's type, such as 'if-statement, 'for-loop, 'function-call, etc.

process-lines: (and label process-helper)
Function takes a list of lines of C code and recursively processes each line (process-helper), converting it to equivalent Lisp code (convert). Returns a list of converted lines.

main:
Function reads a C file (read-file), converts its content into Lisp code (process-lines), and writes the result to an output file (write-file).

---

read-write

read-file:
Function reads the provided file line by line and returns a list of non-whitespace lines.

write-file:
Function writes the given content to an output file and appends the line “(main)” at the end.

---

helper

parse-arithmetic:
Function takes an arithmetic expression as a string, divide it using operators then recursively converts divided into equivalent Lisp prefix notation. Return the prefix arithmetic expression. Remember that this function does not work with the parentheses.

parse-condition:
Function converts C conditions (like if or while conditions) into Lisp’s prefix notation, handling logical operators (||, &&, !) and comparisons (==, !=, >=, etc.). Remember that this function does not work with the parentheses.

parse-inc:
Function parses the increment part of a for loop, handling increment (++), decrement (--), and arithmetic assignments (calls the parse-arithmetic). Only used by convert-for function for simplicity of the code

remove-types:
Function takes a list of parameters and removes their types, leaving only the variable names. Only used by convert-function-definition function for simplicity of the code

type-converter:
Function converts C data types (e.g., int, float, bool) into their corresponding Lisp equivalents. Only used by convert-function-prototype function for simplicity of the code

split-string:
Function splits a string by a given delimiter and returns a list of substrings. Only used by parse-arithmetic function because the cl-ppcre:split function does not work as my expectation.

converters

conversion-foo:
This function identifies the appropriate conversion function to use based on the type of line being processed. For example, when it encounters an “if-statement” line, it selects the convert-if function. This mapping ensures that each type of C code line is handled correctly and transformed into its corresponding Lisp format.

convert:
This function takes a line of C code and processes it for conversion to Lisp. It removes any extra spaces and determines the line type. Based on the line type, it calls the corresponding conversion function. For instance, if it identifies a variable definition, it will call the convert-definition function to handle that specific conversion.

convert-if:
Converts a C if statement to Lisp by extracting the condition and formatting it as a Lisp if expression.

convert-for:
Transforms a C for loop into a Lisp loop by translating the initialization, condition, and increment into appropriate Lisp constructs.

convert-while:
Converts a C while loop into a Lisp loop by extracting the condition and formatting it as a Lisp loop that continues while the condition is true.

convert-definition:
This function processes variable definitions from C code. When it sees a line that defines a variable (e.g., int x = 5;), it splits the line to extract the variable name and its initial value. It checks if this is the first variable definition in the sequence; if so, it starts a new let block in Lisp. If it’s the last definition, it ends the block correctly. The result is a well-structured let binding in Lisp.

convert-assignment:
This function converts assignments like x = 5; from C to Lisp’s setf syntax. It extracts the variable on the left side of the = and the value on the right side, transforming it directly into a Lisp assignment format. The resulting code is clear and maintains the same assignment logic as in C.

convert-function-definition:
This function handles function definitions from C, such as void foo(int a, int b). It extracts the function name and its parameters while ignoring type information. In Lisp, it generates a defun statement that defines the function. For instance, it might create (defun foo (a b)...) to represent the same function definition in Lisp.

convert-function-prototype:
This function takes a C function prototype, like int foo(int a, int b);, and transforms it into a Lisp declaim statement. It extracts the return type, function name, and parameters, converting them into a format that specifies the function’s expected types in Lisp. The result clearly declares the function’s type signature for use in Lisp.

convert-function-call:
This function converts function calls from C to Lisp syntax. For example, when encountering printf("Hello, World!");, it processes the function name and its arguments, converting them into a Lisp format statement. If the function is printf, it replaces C’s format specifiers with Lisp’s equivalent, ensuring the output is formatted correctly in Lisp.

convert-assignment-by-function:
This function processes assignments that result from function calls, such as x = foo();. It identifies the variable being assigned and the function call, then converts the function call into its Lisp representation. The result is a setf expression that assigns the result of the function call to the variable.

convert-return-statement:
This function processes return statements like return x; from C. It extracts the value being returned and formats it into a Lisp expression. The result directly corresponds to the value, ensuring that the return logic is preserved in Lisp.

convert-closing-brace:
When this function encounters a closing brace (}) in C code, it outputs)) in Lisp. This reflects the end of a block of code, properly closing any surrounding progn or loop constructs that were opened previously. It ensures that the Lisp structure remains valid and maintains the flow of the program.

convert-unknown:
This function is a catch-all for any lines of C code that do not match known patterns. When it encounters such a line, it generates a comment in Lisp that indicates the line is unrecognized.