CSE341 – Homework1 Report

Name-Last Name: Emre Kibar Student Number: 210104004093

General Structure Functions

- **main:** Firstly, using read-file function reads the lines input file and then, converts the lines in the recursive-process function. After all lines converted, prints them to the output-file using write-file function.
- **read-file:** Reads the input file line by line and returns them.
- **recursive-process:** Firstly, based on selected line, line-type is found, then the conversion function is found based on line-type and then, the initial line is converted from C-type to Lisp-type. To put "(progn" after "if" line if-flag and if-close variables are used. If-flag will send as 1 on the next recursive call to print first line in the if block with incrementing indentation-level with 2(because of "(progn" block increments indentation by 1), if-close variable holds closing statement for the if block through the recursive flow, until a closing bracket comes while if-close is 1. After all of that, based on conditions, the converted line is constructed with necessary indentation and calls the recursive-statement with necessary parameters.
- **line-type:** Trims initial and trailing spaces and determines the line-type based on keywords and characters.
- **conversion-foo:** Returns the conversion-function based on lineType comparing types.
  convert: Calls the conversion-function with the line.
- **write-file:** Writes all converted lines into the output file at the end.

Conversion Functions

- **convert-if:** Tokenize the line based on whitespace and then, takes the operands and operator which is converted lisp operators using replace-operators function. Then, returns if statement in lisp version.
  Convertible Line Example: "`if (x == a) {`"
- **convert-for:** Tokenize the line based on whitespace and then, extracts the initialization,condition,update parts and create subsequences to form lisp version of for loop.
  Convertible Line Example: "`for (int i = 0; i < 10; i++) {`"
- **convert-while:** Tokenize the line based on whitespace and then, takes the operands and operator which is converted lisp operators using replace-operators function. Then, returns while loop in lisp version by converting condition prefix form.
  Convertible Line Example: "`while (i < 10) {`"

- **convert-assignment-by-function:** Extracts the variable name, function name and the arguments of the function by tokenizing and creating subsequences within the line. They check if return value of function assigned to a newly created variable as definition or it is just an assignment by using var-def-check flag and based on that, returns corresponding lisp version of function assignment. It takes arguments as much as user want.
  Convertible Line Example: "`var1 = sum(2, 3);`""`int var1 = sum(2, 3)`"
- **convert-var-def:** First, checks if the line contains an assignment by function if it contains calls convert-assignment-function else continues its own process and extracts the relevant part of the line based on the line structure using conditions. Then, creates lisp version of equivalent line and returns it.
  Convertible Line Example: "`int var1;`" "`int var1 = 10;`"
  "`int var1 = a + b;`" "`int var1 = sum(2, 3);`"
- **convert-assignment:** First, checks if the line contains an assignment by function if it contains calls convert-assignment-function else continues its own process and extracts the relevant part of the line based on the line structure using conditions. Then, creates lisp version of equivalent line and returns it.
  Convertible Line Example: "`var1 = 10;`" "`var1 = a + b;`"
  "`var1 = sum(2, 3);`"
- **convert-func-def:** Extracts the func-name and param-list from the line by using tokenization of line based on whitespace and commas. Returns lisp version of function definition. It takes arguments as much as user want.
  Convertible Line Example: "`int sum(int a, int b) {`" "`float mult(int a, int b, int c) {`"
- **convert-func-pro-def:** Extracts the func-name, func-parameter types and return types from the line by using selecting subsequences and tokenization based on whitespace and commas. Converts the C-types to Lisp-types and returns the lisp version of function prototype definition. It takes arguments as much as user want.
  Convertible Line Example: "`int sum(int a, int b);`" "`float mult(int a, int b, int c);`"
- **convert-func-call:** Extracts the func-name and arguments from the line based on taking relevant parts of the line by checking the character positions and tokenization of line based on commas. Returns lisp version of function call. It takes arguments as much as user want.
  Convertible Line Example: "`sum(a, b);`" "`mult(a, b, c, d);`"
- **convert-print:** Extracts the format-string and arguments from the line based on taking relevant parts of the line by checking the character positions and tokenization of line based on commas. Also, converts the C-type specifiers to the Lisp-type specifiers (%d -> ~d) by using replace-format-specifiers helper function. Returns lisp version of printf. It takes arguments as much as user want.

Convertible Line Example: “`printf("Hello, World! %d", variable);`” “`printf("%d %c %f %s", int1, char1, float1, string1);`”

- **convert-return:** Extracts the relevant part of the line by seperating uncessary parts and tokenize the line based on whitespace. Returns the lisp version of return. Convertiable Line Example: “`return a;`” “`return 3;`” “`return a + b;`”
- **convert-block-end:** Returns a closing parathesis as end of a block. Convertible Line Example: “}”
- **convert_unknown:** For error check purpose, it returns empty string.

Helper Functions

- **contains-substring:** Search the string for a substring and returns true if there is else returns false.
- **split-by-delimiter:** The split-by-delimiter function splits a string line by a specified delimiter and removes any empty substrings that may result from consecutive delimiters. It loops through line, finding each delimiter's position, extracting the substring before it, and adding this to a result list. After processing all delimiters, it pushes any remaining portion of line to result. Finally, the list is reversed to maintain the order of the original string’s segments.
- **replace-operators:** The replace-operators function checks if a given token is a C-style comparison operator and, if so, replaces it with its Lisp equivalent (e.g., "==" becomes "=", "!=" becomes "/="). If the token doesn’t match any specified operator, it returns the token unchanged.
- **replace-format-specifiers:** Converts C-style format specifiers in a string to Lisp-compatible ones. It iterates over a list of predefined replacement pairs (e.g., `"%d"` to `"~d"`) and searches for each C-style specifier within the input string. When a match is found, it replaces the C-style specifier with its Lisp equivalent and updates the string. This process continues until all specifiers are replaced, making the string compatible with Lisp format requirements.
- **calculate-indentation:** Creates a string formed with whitespaces for selected indentation-level.