CSE 341 Programming Languages HW1 Report

line-type (line):

Takes the line that’s being processed and returns a symbol as to what kind of line it is. Works using “cl-ppcre” library (perl-like regex for common lisp). This function has a precedence that’s being handled by the (cond ) by order in which the code is written in.

conversion-foo (line-type):

Takes a symbol line-type can return. With a case determines the correct function to call and returns that function.

convert (line conversion-fn):

Takes a line and function to call with that line, calls the function with line parameter.

convert-lines (lines):

Until the end of lines, recursively processes each line, convert and construct a list of lines.

c-to-lisp-operator (operator):

Convert a c operator into a lisp symbol bound to functions with the same functionality.

c-to-lisp-type-names (type):

Convert types such as int float to lisp equivalent. Lisp typically does not need types but for function declarations it is needed. It makes the compilers job easier.

c-to-lisp-arithmetic(expression):

Convert a C arithmetic expression to Lisp such as 'a + b' to '(+ a b)', or 'a > b' to '(> a b)'.

If there is only one token, return it surrounded by parentheses, otherwise, handle it as an operator with two operands. Rearrange the order, convert C operator to lisp equivalent function with c-to-lisp-operator.

extract-if (str):

takes a string, uses register-groups-bind to parse parameters of if expression, returns the parsed expression.

convert-if(line):

Calls extract-if and c-to-lisp-arithmetic-expression to convert if parameters in a lisp way.

extract-for (line):

Takes a C for line and parses it into 3 groups: “init, cond, incr”

Continues parsing that 3 groups into their lisp equivalents: start end var-name.

Returns a list of that equivalents.

convert-for (line):

Uses a list returned by extract-for to create lisp-style for loop.

extract-while (line):

Parses and returns condition of C while.

convert-while (line):

Uses extract-while and c-to-lisp-arithmetic to write lisp equivalent.

extract-assignment (line):

Uses register-groups-bind, discards type name, filters out expression and returns name and value as a list.

convert-assignment (line):

Uses extract-assignment, does a scan to determine which kind of assignment it is  
(No argument function assignment, multi-argumented function assignment, basic assigment). Creates lisp equivalent assignment (Lisp actually does not have assignments, converting them to “let” is not really possible without knowing lower line or extremely specific inputs. So i convert them into “setf” statements.

convert-function-call(line):

Parses parameters and name of a function to construct the lisp equivalent. Uses split and register-groups-bind to do so.

convert-print (line):

Converts a printf call to format, with limited functionality. This is an override upon convert-function-call.

split-param-helper (params):

 Recursive helper to split the parameters, discarding types such as int, float, void\* etc.

 Works by splitting the parameters by comma and then extracting the parameter names.

extract-function (str):

Split a single C line such as “int main(int a, int b){“ into the function name and parameter names. Returns a list with the function name as the first element and the parameter names as the other elements.

convert-function-definition (line):

Converts a function definition to a (defun) using extract-function.

convert-function-declaration (line):

Uses split-param-helper and c-to-lisp-type-names to convert a function declaration line into a (declaim) line, not really necessary, currently not working.

extract-return (line):

Extract the return value from a C-style return statement, parses into groups and handles arithmetic return (such as “return a+b;”) using c-to-lisp-arithmetic if necessary. Returns filtered line.

convert-return (line):

Uses output of extract-return to construct lisp return.

convert-block-end (line):

Since C is an imperative language program should be executed line by line, lisp is not like that but it supports doing that with a (progn) call, since all “blocks” need progn’s and we cant really check out other lines or keep track of block start-ends, we convert them into “))”, assuming all blocks start with progn. Ex: “int main(){“ to “(defun main (progn”

convert-block-start:

convert block start (“{“) to “(progn “.

convert-unknown (line):

comment out unknown lines by adding “;;” to start of the line and keep it as is.

read-file (filename):

Read file line by line into a list.

write-file (filename content):

Write the converted Lisp code inside “content” to the output file.

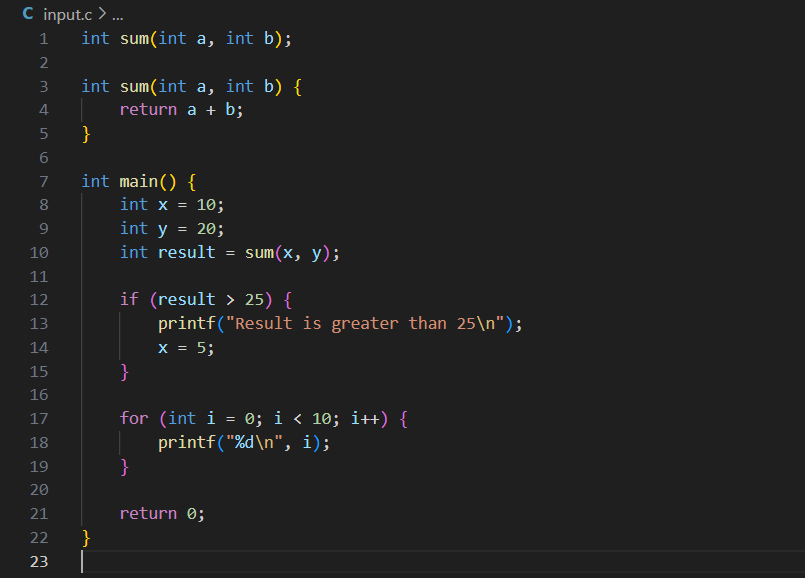
main (input-file output-file):

process “input-file” (a C file) and convert its content to “output-file” (a lisp file).

m:

A macro for calling (main “input.c” “output.lisp”)

Input:



Output:

