CSE/IT 324 Final Project 2021 Spring

Individual Project: Create a LISP interpreter in C++

A makefile MUST be provided that works on CS machines. If your files do not compile on the CS machines you will receive a 0 for the code section of the project!

Introduction & Goal:

The goal of this project is to learn about the basic constructs of Lisp while simultaneously understanding the basic concepts behind creating an interpreter in LISP, a simple functional programming language. This project will bring together many concepts that have been taught in this class over the entire semester.

The final result of this project will be a C++ code that executes LISP style commands from a prompt all while within a console environment. In addition, you must write a report describing the inner workings of your code, and implementation details.

Code Implementation Requirements:

C++ code MUST successfully parse and execute LISP code as outlined below in the table. The code you implement MUST functions like an interactive LISP interpreter and output all intermediate results to a file. The code MUST be able to compile and execute on the CS machines.

An example of execution flow is as follows:

Make command (*make*)

```
Welcome to the fancy new Prompt LISP INTERPRETER, type in LISP commands!
> ( + (+ 2 3) 5)
> 10
> (define b 2)
> b
> (defun ADD (x y) (+ x y))
> ADD
> (ADD (8 3))
> 11
> (quit)
> bye
EXECUTION STOPPED
```

```
/** You need to have an output file to record all outputs of a Lisp interpreter session **/
/** You can have your own format for the output file as long as it is readable.**/
/** file name: "results.file": **/
10
b
ADD
11
EOF
/** end of file**/
```

Your interpreter MUST allow for the following Lisp constructs, each construct accounts for 5 pts:

Lisp Construct	**Syntax**	**Semantics and Example**
Variable reference	*var*	A symbol is interpreted as a variable name; its value is the variable's value. Example: r → 10
Constant literal	*number&boolean*	A number evaluates to itself. Example: 12 → 12, T→ T, NIL→ NIL
Quotation	'*exp*	Return the *exp* literally, do not valuate it Example: '(+ 1 2) → (+ 1 2)
Conditional	(if *test conseq alt*)	Evaluate *test;* if true, evaluate and return *conseq;* otherwise *alt*. Example: (if (> 10 20) (+ 1 1) (+ 3 3)) → 6
Variable Definition	(*define var exp*)	Define a new variable and give it the value of evaluating the expression *exp*. Examples: (define r 10)
Function call	(*func arg…*)	If *func* is anything other than one of the other symbols (if, define, or quote) then treat it as a function. Evaluate *func* and all the *args*, and then the function is applied to the list of *arg* values. Example: (sqrt (* 2 8)) → 4.0
Assignment	(*set! var exp*)	Evaluate *exp* and assign that value to *var*, which must have been previously defined (with a *define* or as a "formal" parameter). Example: (set! r2 (* r r))
Function Definition/ declaration	(*defun foo* (*var*$_1$ *var*$_2$…) *exp*)	Create a function *foo* with formal parameter(s) named *var*$_1$, *var*$_2$… , and *exp* as the body. Create this using static scoping. Example: (defun (r) (* pi (* r r))) [Note: here *defun* is used to denote function declaration]

Table 1. Lisp Constructs

You must implement the following pre-defined operators and functions. You will get maximum 15 pts (3 pts each).

Predefined Operator&Function	**Semantics and Example**
The following arithmetic +, -, *, / operators on integer type.	Integer arithmetic computation using **32-bit integers**. You must handle the "divided by zero" error, and overflow for 32-bit integers in the processing of the given arithmetic operations, above.
"car" and "cdr"	Refer to Lisp lecture how the "car" and "cdr" work.
The built-in function: "cons"	Examples: (cons 'to '(be or not)) → (to be or not) (cons '(to be) '(or not)) → ((to be) or
sqrt, pow	Math built-in functions. Examples: (sqrt (+ 2 2)) → 2 (pow -2 2) →4
>, <, =, !=, and, or, not	Logic Relation operators : /**In CLISP, **T** is True, **NIL** is False**/ Examples: (< 2 3) → T (and NIL T) → NIL (or T NIL) → T

Table 2. Pre-defined operators and functions

Your code must pass the provided the primary test file posted on Canvas to receive full credits of code implementation.

One file have uploaded containing test cases. e.g. "test_cases.txt".

Report Requirements:

The report must be in the most recent standard ACM format, a template of which can be found at https://www.acm.org/publications/proceedings-template. You can also use the template from project 1. No example document will be posted on the Canvas, document sectioning is up to you.

The report must contain

1. Make a table like Table 3 belwo to show what has been implemented fully, what is partially done, what didn't be implemented. Please fill the following two by choosing one status from given options for each entry.

Lisp Construct	Status (Done, partially done, not
Variable reference	
Constant literal	
Quotation	
Conditional	
Variable Definition	
Function call	
Assignment	
Function Definition	
The arithmetic +, -, *, / operators on integer type.	
“car” and “cdr”	
The built-in function: “cons”	
sqrt, exp	
>, <, ==, <=, >=, !=	

Table 3. Project Progress/Final Status Reporting

The following functions are for extra credits, 10 points each:

10 Extra credits each	**Syntax**	**Semantics and Example**
“*mapcar*” built-in function	(mapcar ‘*func* ‘(*exp*) ‘(*exp*))	“mapcar” as defined in class. Examples: (mapcar ‘+ ‘(1 2 3) ‘(4 5 6)) → (5 7 9) (mapcar ‘sqrt ‘(4 9 16)) → (2 3 4)
“*lambda*” built-in function	(lambda (*var*...) (*exp*))	A nameless function as defined in class. Examples: (lambda (x) (+ x 2)) > (FUNCTION: LAMBDA (x)) ((lambda (x) (+ x 1 00) 100) > 200

Table 4. Extra Credits Functions

2. Descriptions of how each expression in the above table was implemented. Most descriptions should include the types of C++ constructs you used for your implementation and other various information pertaining to the specific expression that is being described. (Example: The implementation of definition involved using an STL library that…)
3. Descriptions of class topics that apply to each expression implementation. (Example: We learned about static scoping and how information on the stack is represented in the class. This idea is applied when using different environments for various functions…)
4. Descriptions of implementation details that are important/pertain to the completion of the project but don't necessarily fall into the two previous categories. (Example: In order to implement local/global environments in the usage of functions I did the following…; I defined various lisp symbols (like +, -, *, <, etc.) using…)
5. Anything else you think worth mentioning (why your solution is exceptionally good, or areas that your implementation lack in if you couldn't get one of the expressions to work)
6. A final section that describes your concluding thoughts on this project and if you thought it was fun/interesting.
7. A list of references (website, tutorial, code source, or otherwise) used when implementing this project. You are free to use any source you find on the Internet, just make sure to reference it!

Submission Requirements:

Your assignment must be submitted before the scheduled time and date on the Canvas website. Your report should be submitted as a PDF file and your code should be submitted as a tarball. The PDF must conform to the naming convention "**lastname_project2.pdf**". The tarball must conform to the naming convention "**lastname_firstname.tar.gz**". In both of these cases, obviously, replace "lastname" with YOUR last name and "firstname" with YOUR first name.

In your code tarball, there must be a **MAKEFILE** that compiles your code and creates the executable that, when run, will come up with a prompt similar to the above example. This prompt should be able to execute LISP-style code matching the format of the expressions in the table defined above. Also, include a **README** that contains specific details on the running environment and how to use your code if you cannot get it to conform to the requirements. If your code works exactly as requested you can omit the README, but if it doesn't work as intended you will lose points.

Grading:

Each submission will be graded as follows:

75pts Code Implementation

- 15pts Style & commenting (Readability)
- 7pts For outputting to a file
- 6pts For having a prompt that evaluates expressions
- 15pts Pre-defined operators&functions, 4pts each
- 32pts Correct implementation to each expression, 4pts each

25pts Report

- 5pts Grammar
- 5pts Formatting and Citation
- 15pts Analyses and Descriptions of required content described above

15pts Project progress report

You must submit a project progress report before the due date posted on Canvas. Your report should be submitted as a PDF file. The PDF must conform to the naming convention "**lastname_project2_progress_report.pdf**". The progress report must contain what have you done so far to be written in **Table 3,** as shown above. All Lisp constructs you mark complete in your progress report must be completed and included in your final project report, otherwise points will be deducted.

You need not to be reminded of the following directions about the implementation of your final LISP project:

Academic Honesty

Assigned class projects and Programming assignments MUST be done **on your own**. You may **NOT** share solutions or drafts of solutions, or even attempts at solutions with other students in this class or **previous semesters' classes**. Make absolutely sure that you understand NMT's plagiarism policy (in the NMT catalog), any violation of the aforementioned direction will result in ZERO grade assigned for the project, and possibly further actions as reporting to the administration through the undergraduate dean of students.

Have fun and good luck!