LISP

(LISt Programming Language)

ACSL Contest #2

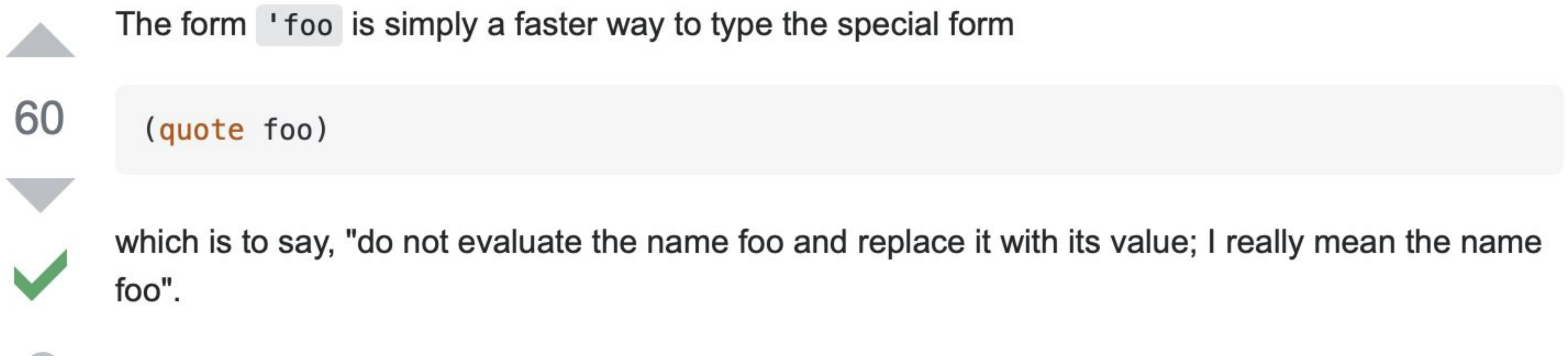
What is LISP?

- Stands for “LISt Programming language:
  - Developed in the mid-1950s by John McCarthy at MIT
  - Historically used for artificial intelligence programs and is often the environment of choice for applications which require a powerful interactive working environment.
  - There’s a number of different “dialects” of LISP
- Presents a very different way to think about programming in comparison to “algorithmic” languages such as Python, C++, and Java.

Syntax

- The basis of LISP is a list, which is construct by enumerating elements inside a pair of parentheses.
  - `(23 (this is easy) hello 821)`
- The elements in the list, which are not lists, are called atoms.
  - The atoms in the list above are 23, ‘this, ‘hello, 821, ‘easy, and ‘is
  - Literals are defined with a single leading quote:
- Everything in LISP is either an atom or a list (but not both)
  - The only exception is “NIL” which is both an atom and a list.
    - NIL can also be written as “()”

A little more on the significance of ‘


60

The form `'foo` is simply a faster way to type the special form

```
(quote foo)
```

which is to say, "do not evaluate the name foo and replace it with its value; I really mean the name foo".

https://stackoverflow.com/questions/1539144/what-is-apostrophe-in-lisp-scheme

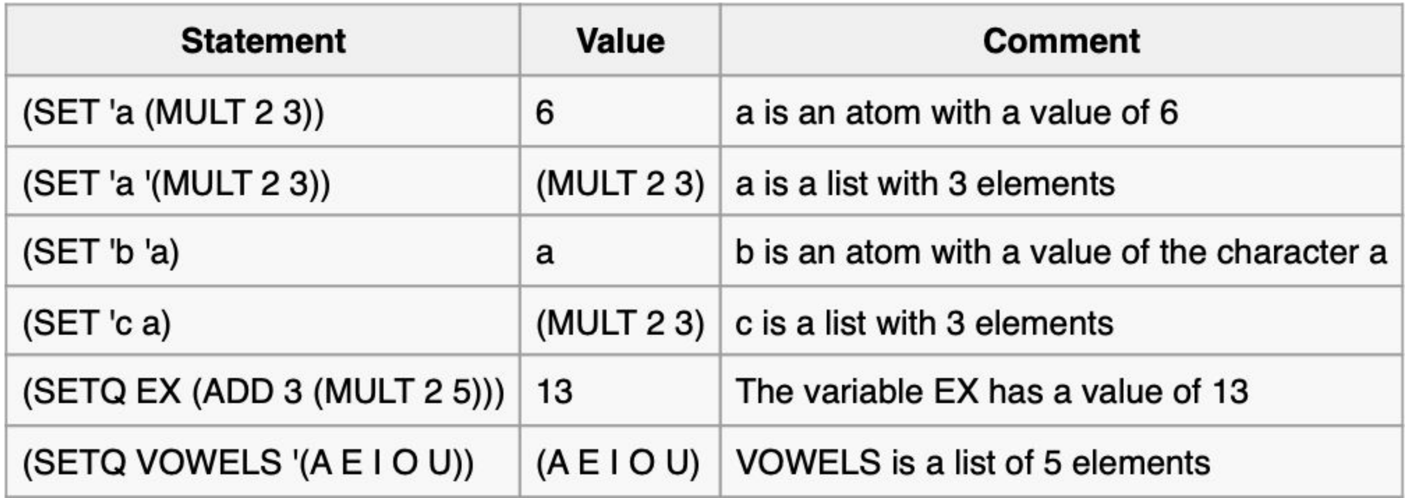
Syntax

- All statements in LISP are function calls with the following syntax:(function arg1 arg2 arg3 … argn)
- To evaluate a LISP statement, each of the arguments (possibly functions themselves) are evaluated, and then the function is invoked with the arguments.
    - For example, (MULT (ADD 2 3) (ADD 1 4 2)) has a value of 35, since (ADD 2 3) has a value of 5, (ADD 1 4 2) has a value of 7, and (MULT 5 7) has a value of 35.
- Some functions have an arbitrary number of arguments while others require a fixed number.
- All statements return a value, which is either an atom or a list.

Basic Functions: SET and SETQ

We may assign values to variables using the function SET. For example, the statement (SET 'test 6) would have a value of a 6, and (more importantly, however) would also cause the atom 'test to be bound to the atom 6. The function SETQ is the same as SET, but it causes LISP to act as if the first argument was quoted.

Link to info on why there’s a difference between SET and SETQ: http://ergoemacs.org/emacs/elisp_symbol.html

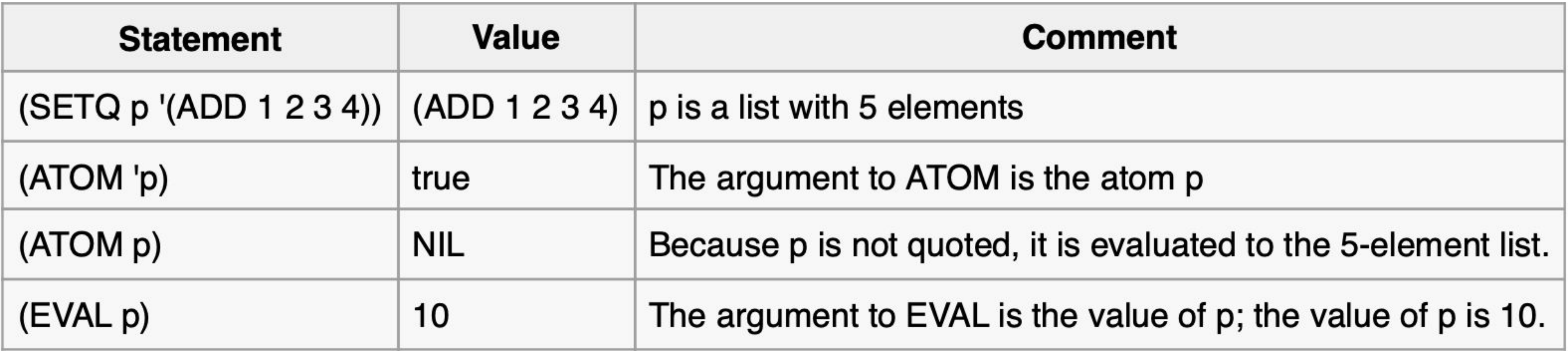


Statement	Value	Comment
(SET 'a (MULT 2 3))	6	a is an atom with a value of 6
(SET 'a '(MULT 2 3))	(MULT 2 3)	a is a list with 3 elements
(SET 'b 'a)	a	b is an atom with a value of the character a
(SET 'c a)	(MULT 2 3)	c is a list with 3 elements
(SETQ EX (ADD 3 (MULT 2 5)))	13	The variable EX has a value of 13
(SETQ VOWELS '(A E I O U))	(A E I O U)	VOWELS is a list of 5 elements

Basic Functions: EVAL and ATOM

- The function EVAL returns the value of its argument, after it has been evaluated.
    - For example, (SETQ z '(ADD 2 3)) has a value of the list (ADD 2 3); the function (EVAL 'z) has a value of (ADD 2 3); the function (EVAL z) has a value of 5 (but the binding of the atom z has not changed).
- In this last example, you can think of z being "resolved" twice: once because it is an argument to a function and LISP evaluates all arguments to functions before the function is invoked, and once when the function EVAL is invoked to resolve arguments.
- The function ATOM can be used to determine whether an item is an atom or a list. It returns either "true", or "NIL" for false.

Basic Functions: EVAL and ATOM



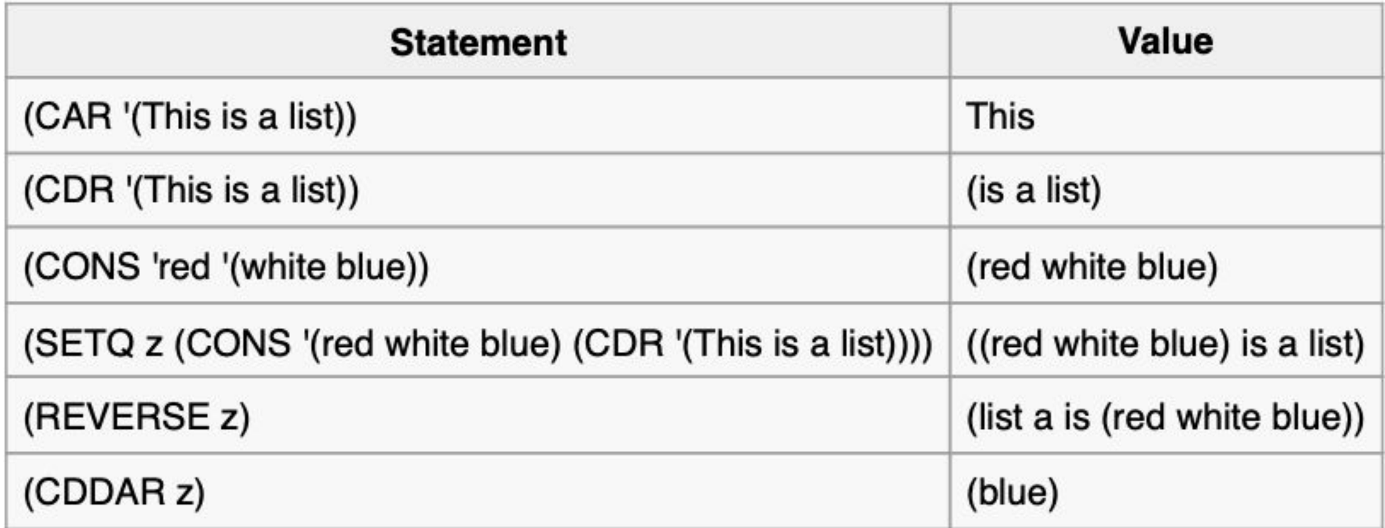
Statement	Value	Comment
(SETQ p '(ADD 1 2 3 4))	(ADD 1 2 3 4)	p is a list with 5 elements
(ATOM 'p)	true	The argument to ATOM is the atom p
(ATOM p)	NIL	Because p is not quoted, it is evaluated to the 5-element list.
(EVAL p)	10	The argument to EVAL is the value of p; the value of p is 10.

List Functions: CAR, CDR, CONS, REVERSE

- The two most famous LISP functions are CAR and CDR (pronounced: could-er), named after registers of a now long-forgotten IBM machine on which LISP was first developed.
- The function (CAR x) returns the first item of the list x (and x must be a list or an error will occur); (CDR x) returns the list without its first element (again, x must be a list).
- The function CONS takes two arguments, of which the second must be a list. It returns a list which is composed by placing the first argument as the first element in the second argument's list.
- The function REVERSE returns a list which its arguments are in reverse order.

List Functions: CAR, CDR, CONS, REVERSE

- The CAR and CDR functions are used extensively to grab specific elements of a list or sublist, that there's a shorthand for this: (CADR x) is the same as (CAR (CDR x)), which retrieves the second element of the list x; (CAADDAR x) is a shorthand for (CAR (CAR (CDR (CDR (CAR x))))), and so on.



Statement	Value
(CAR '(This is a list))	This
(CDR '(This is a list))	(is a list)
(CONS 'red '(white blue))	(red white blue)
(SETQ z (CONS '(red white blue) (CDR '(This is a list))))	((red white blue) is a list)
(REVERSE z)	(list a is (red white blue))
(CDDAR z)	(blue)

Arithmetic Functions

Function	Result
(ADD x1 x2 …)	sum of all arguments
(SUB a b)	a-b
(MULT x1 x2 …)	product of all arguments
(DIV a b)	a/b
(SQUARE a)	a*a
(EXP a n)	a^n
(EQ a b)	true if a and b are equal, NIL otherwise
(POS a)	true if a is positive, NIL otherwise
(NEG a)	true if a is negative, NIL otherwise

Statement	Value
(ADD (EXP 2 3) (SUB 4 1) (DIV 54 4))	24.5
(- (* 3 2) (- 12 (+ 1 2 1)))	-2
(ADD (SQUARE 3) (SQUARE 4))	25

User-defined Functions

- LISP also allows us to create our own functions using the DEF function. (ACSL will sometimes use DEFUN rather than DEF, as it is a bit more standard terminology.)
- Example: `(DEF SECOND (args) (CAR (CDR args)))`
  - Defines a function called SECOND which operates on a single parameter named “args.”
  - SECOND will take the CDR of the parameter and the CAR of that result.
  - SECOND ‘(a b c d e) would first CDR the list to return (b c d e) and then CAR that value to return “b”

User-defined Functions

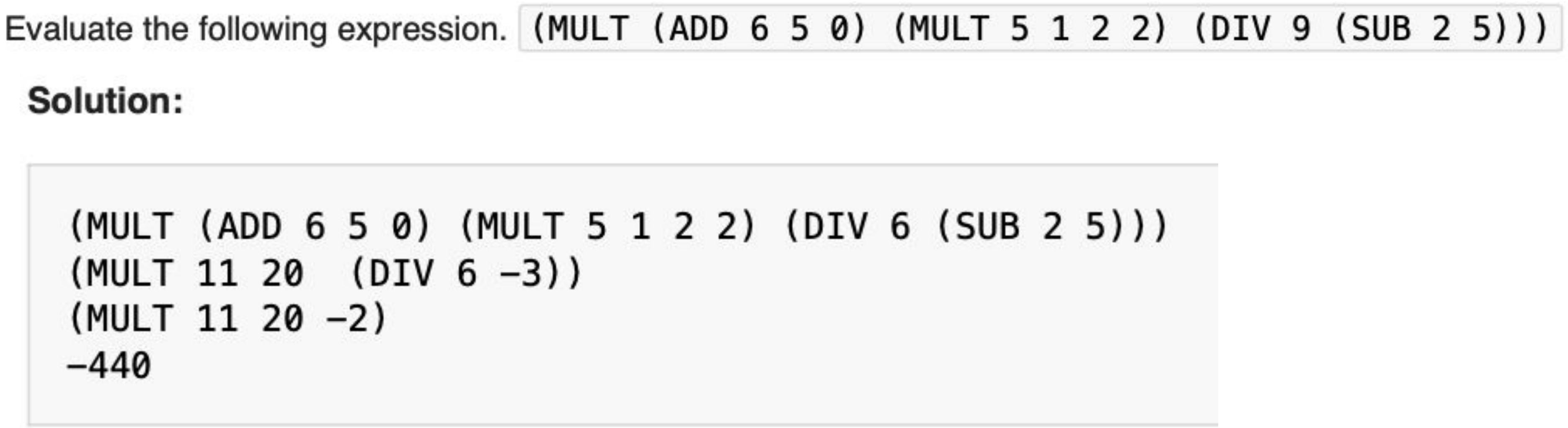
```
(SETQ X '(a c s l))
(DEF WHAT(args) (CONS args (REVERSE (CDR args))))
(DEF SECOND(args) (CONS (CAR (CDR args)) NIL))
```

Statement	Value
(WHAT X)	((a c s l) l s c)
(SECOND X)	(c)
(SECOND (WHAT X))	(l)
(WHAT (SECOND X))	((c))

ACSL Info

There are many online LISP interpreters available on the Internet. The one that ACSL uses for testing its programs is CLISP that is accessible from JDoodle. This interpreter is nice because it is quite peppy in running programs, and functions are not case sensitive. So, `(CAR (CDR x))` is legal as is `(car (cdr x))` One drawback of this interpreter is the print function changes lowercase input into uppercase.

Sample Problem 1



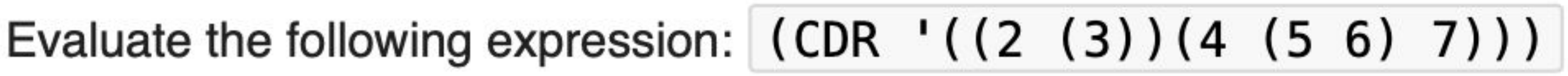
Evaluate the following expression. `(MULT (ADD 6 5 0) (MULT 5 1 2 2) (DIV 9 (SUB 2 5)))`

Solution:

```
(MULT (ADD 6 5 0) (MULT 5 1 2 2) (DIV 6 (SUB 2 5)))
(MULT 11 20  (DIV 6 -3))
(MULT 11 20 -2)
-440
```

Note: ACSL’s solution above has switched the 9 in DIV to a 6, hence the discrepancy in answers.

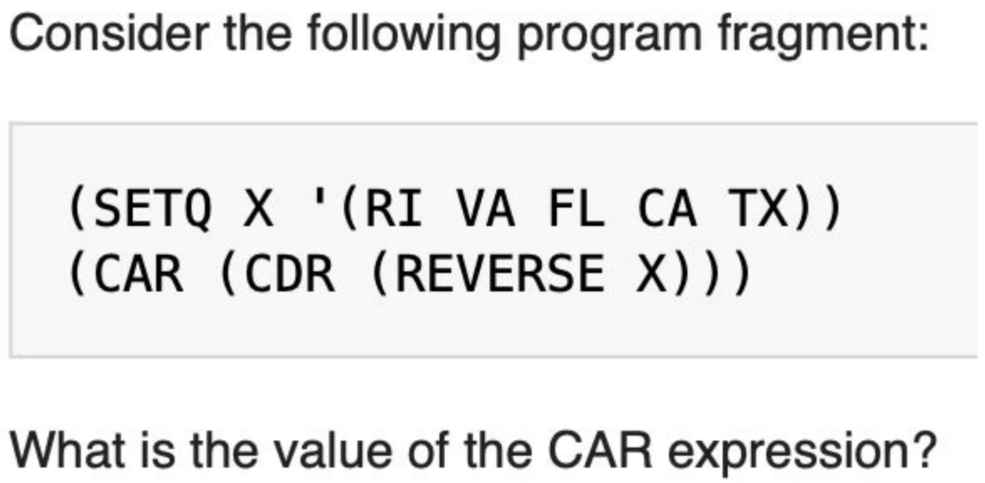
Problem 2



Evaluate the following expression: `(CDR '((2 (3))(4 (5 6) 7)))`

Solution: The CDR function takes the first element of its parameter (which is assumed to be a list) and returns the list with the first element removed. The first element of the list is (2 (3)) and the list without this element is ((4 (5 6) 7)), a list with one element.

Problem 3



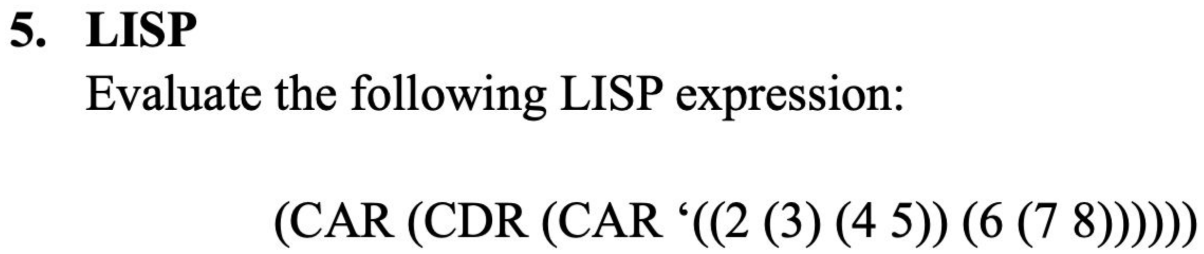
Consider the following program fragment:

```
(SETQ X '(RI VA FL CA TX))
(CAR (CDR (REVERSE X)))
```

What is the value of the CAR expression?

Solution: The first statement binds variable X to the list '(RI VA FL CA TX). The REVERSE of this list is '(TX CA FL VA RI) whose CDR is '(CA FL VA RI). The CAR of this list is just the atom CA (without the quote).

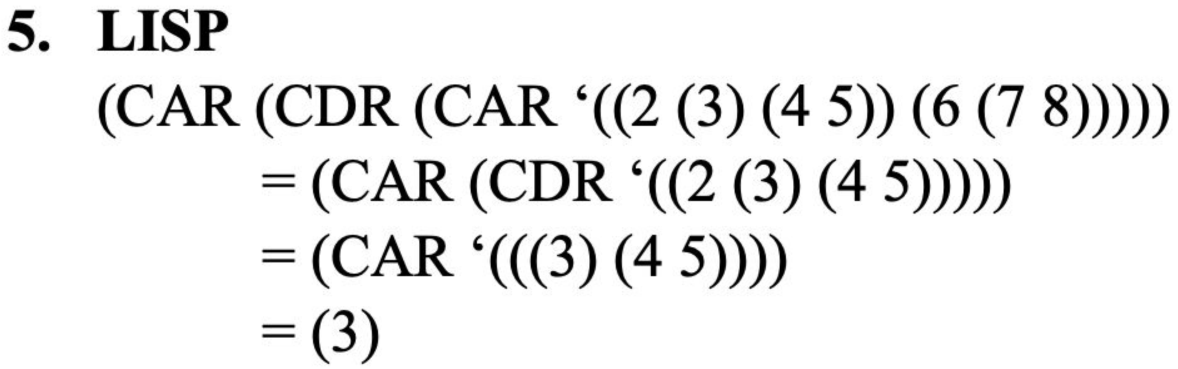
Past Contests: Senior



5. LISP

Evaluate the following LISP expression:

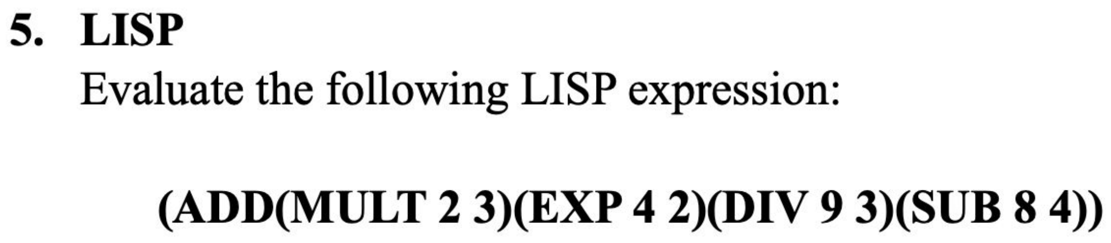
```
(CAR (CDR (CAR ‘((2 (3) (4 5)) (6 (7 8))))))
```



5. LISP

```
(CAR (CDR (CAR ‘((2 (3) (4 5)) (6 (7 8))))))
        = (CAR (CDR ‘((2 (3) (4 5)))))
        = (CAR ‘(((3) (4 5))))
        = (3)
```

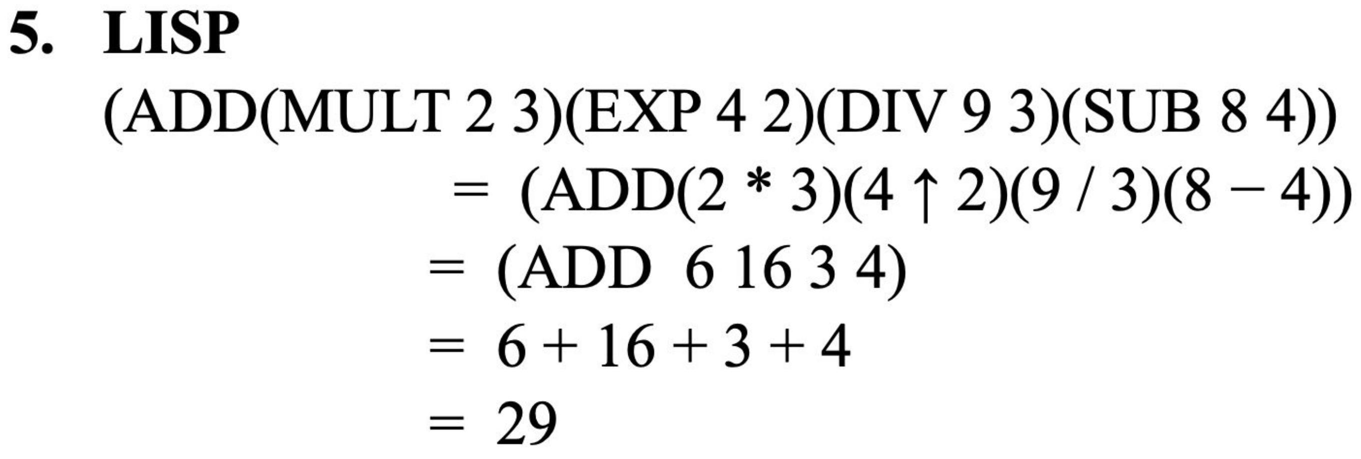
Past Contests: Intermediate



5. LISP

Evaluate the following LISP expression:

(ADD(MULT 2 3)(EXP 4 2)(DIV 9 3)(SUB 8 4))



5. LISP

(ADD(MULT 2 3)(EXP 4 2)(DIV 9 3)(SUB 8 4))

$= (\text{ADD}(2 * 3)(4 \uparrow 2)(9 / 3)(8 - 4))$

$= (\text{ADD}\ \ 6\ 16\ 3\ 4)$

$= 6 + 16 + 3 + 4$

$= 29$

Source Links

https://www.categories.acsl.org/wiki/index.php?title=Main_Page

https://www.acsl.org/get-started/study-materials