1. **Определите предикат, проверяющий: Является ли одно множество подмножеством другого.**

**(defun is-subset (set1 set2)**

**(cond**

**((null set1) t)**

**((null set2) nil)**

**((member (car set1) set2)**

**(is-subset (cdr set1) set2))**

**(t nil)**

**)**

**)**

Вызов:

**(is-subset '(3 2) '(1 2 3))**

Первое условие проверяет, является ли **set1** пустым списком (пустым множеством). Если **set1** пуст, то возвращается **t**, что означает, что **set1** является подмножеством **set2**.

Второе условие проверяет, является ли **set2** пустым списком. Если **set2** пуст, а **set1** не пуст, то возвращается **nil**, так как непустое множество не может быть подмножеством пустого множества.

Если **member** вернет **t**, что означает, что текущий элемент содержится в **set2**, то вызывается рекурсивно функция **is-subset**, передавая хвост **set1** и **set2**. Это позволяет продолжить проверку оставшихся элементов множества **set1**.

Эта строка означает "иначе" в блоке **cond**. Если ни одно из предыдущих условий не выполняется, то функция возвращает **nil**, что означает, что **set1** не является подмножеством **set2**.

1. **Определите функцию возвращающую разность двух множеств, т.е. множество из элементов первого множества, не входящих во второе.**

(defun diff (set1 set2 &optional (result '()))

(cond

((null set1) (nreverse result))

((member (car set1) set2)

(diff (cdr set1) set2 result))

(t

(diff (cdr set1) set2 (cons (car set1) result))

)

)

)

Вызов:  
(diff '(3 4 5 6) '(1 2))

**defun** - это специальный оператор в Common Lisp, используемый для определения новой функции с именем **diff**. В данном случае, функция принимает два аргумента **set1** и **set2**, представляющие два множества, а также необязательный аргумент **result**, который используется для накопления результата.

**&optional** - это специальный параметр в языке Common Lisp, который используется для определения необязательных аргументов в определении функции. Когда функция вызывается, аргументы, указанные с **&optional**, могут быть опущены, и при этом будут использоваться значения по умолчанию, которые указаны после аргумента.

В вашей функции **diff**, **&optional (result '())** указывает, что **result** - это необязательный аргумент. Если он не указан при вызове функции, то будет использоваться пустой список **()**. Если **result** передается при вызове функции, то он будет использоваться как начальное значение для накопления результата.

((null set1) (nreverse result))

Это первое условие в блоке **cond**. Оно проверяет, является ли **set1** пустым списком (пустым множеством). Если **set1** пуст, то возвращается текущий **result**, что означает, что разность множеств **set1** и **set2** найдена.

((member (car set1) set2)

Второе условие. Оно проверяет, содержит ли **set2** текущий элемент из головы **set1**. Если содержит, то элемент из **set1** присутствует в **set2**, и он пропускается.

(diff (cdr set1) set2 result))

Если текущий элемент из **set1** присутствует в **set2**, то мы вызываем рекурсивно функцию **diff**, передавая хвост **set1** и **set2**, и оставляем текущий **result** без изменений.

(t

(diff (cdr set1) set2 (cons (car set1) result)))

Если текущий элемент из **set1** не входит в **set2**, мы вызываем рекурсивно функцию **diff**, передавая хвост **set1**, **set2**, и добавляем текущий элемент в начало **result** с помощью функции **cons**. Это позволяет нам накапливать элементы из **set1**, которые не входят в **set2**, в результате.

**(cons (car set1) result)** - **cons** - это функция, которая используется для создания нового списка, добавляя элемент в начало существующего списка **result**. В данном случае, мы берем текущий элемент из **set1** (первый элемент) и добавляем его в начало списка **result**.

Функция **nreverse** в Common Lisp используется для изменения порядка элементов в списке, переворачивая его в обратном направлении. Это означает, что первый элемент становится последним, второй - предпоследним, и так далее. **nreverse** выполняет операцию инверсии списка "на месте", что означает, что она изменяет переданный список, а не создает новый.

1. **Определите функционал, вставляющий перед каждым элементом списка, обладающим определенным свойством, символ \*. Проверьте работу функционала для предикатов: - неотрицательное число (функциональный аргумент – лямбда выражение); - четное число (функциональный аргумент – имя встроенного предиката evenp).**

Задание заключается в том, чтобы создать функцию на языке программирования Lisp (или его диалекта, такого как Common Lisp) без использования операторов **set**, **let**, и **setq**. Вместо этого, вы должны использовать рекурсивный подход.

Функция **insert-asterisk** должна вставлять символ "\*" перед каждым элементом списка, который соответствует определенному свойству или условию, заданному в виде предиката. Предикат - это функция, которая принимает элемент списка в качестве аргумента и возвращает true или false в зависимости от того, соответствует ли элемент заданному условию.

1. **Функционал:**
   * Функционал (или функциональ) - это термин, используемый для описания функции или процедуры в функциональном программировании. Функционалы включают в себя функции, которые могут быть переданы как аргументы другим функциям, возвращены из других функций, и использованы для построения более сложных вычислений.
   * Функциональное программирование подчеркивает работу с функциями как с данными, и функционалы играют важную роль в этой парадигме. Примером функционала может быть функция высшего порядка, которая принимает другие функции в качестве аргументов, такие как **map**, **filter** или **reduce**.
2. **Предикат:**
   * Предикат - это функция или выражение, которое возвращает булевское значение (true или false) в зависимости от выполнения определенного условия. В контексте программирования, предикаты используются для проверки условий и принятия решений в коде.
   * Например, предикат может проверять, является ли число четным или нечетным, является ли строка пустой или содержит текст, соответствует ли элемент списка определенному критерию и так далее. В Lisp, вы можете использовать предикаты в функциях для фильтрации, сортировки и обработки данных.

(defun func (predicate lst)

(cond

((null lst) nil)

((funcall predicate (car lst))

(cons '\* (cons (car lst) (func predicate (cdr lst))))

)

(t

(cons (car lst) (func predicate (cdr lst)))

)

)

)

; Определение списка и предикатов

(setq numbers '(1 2 3 4 5 6 7 8 9 10))

(setq odd (lambda (x) (not (evenp x)))

neg (lambda (x) (>= x 0)))

; Вызов функции с предикатом для нечетных чисел

(setq result-odd (func odd numbers))

(format t "Результат с предикатом для нечетных: ~a~%" result-odd)

; Вызов функции с предикатом для неотрицательных чисел

(setq result-neg (func neg numbers))

(format t "Результат с предикатом для неотрицательных: ~a~%" result-neg)

(defun func (predicate lst)  
**defun func (predicate lst)** - Это объявление функции **func**, которая принимает два аргумента: **predicate** (предикат) и **lst** (список).

((null lst) nil)

**((null lst) nil)** - Это первая ветвь **cond** и проверка на то, если **lst** пуст (первый аргумент функции), то возвращается пустой список (**nil**).

((funcall predicate (car lst))

**((funcall predicate (car lst))** - Это вторая ветвь **cond** и проверка, если предикат (функция, переданная в **predicate**) выполняется для первого элемента списка (**car lst**).

(cons '\* (cons (car lst) (func predicate (cdr lst))))

)

**(cons '\* (cons (car lst) (func predicate (cdr lst))))** - Если предикат выполняется для первого элемента, то вставляется символ **\*** перед текущим элементом (первым элементом), а затем вызывается рекурсивно функция **func** для остальных элементов списка (**cdr lst**).

(t

(cons (car lst) (func predicate (cdr lst)))

)

)

**(t (cons (car lst) (func predicate (cdr lst))))** - Ветвь **t** используется, если предикат не выполняется для первого элемента. В этом случае текущий элемент просто добавляется к результату, и функция вызывается рекурсивно для остальных элементов списка.

Теперь разбираем вывод:  
  
(setq numbers '(1 2 3 4 5 6 7 8 9 10))  
**(setq numbers '(1 2 3 4 5 6 7 8 9 10))** - Эта строка кода использует функцию **setq** для определения переменной **numbers**. **numbers** устанавливается в список **(1 2 3 4 5 6 7 8 9 10)**. Этот список представляет собой набор чисел от 1 до 10.  
  
(setq odd (lambda (x) (not (evenp x)))  
**(setq odd (lambda (x) (not (evenp x)))** - Здесь мы определяем переменную **odd** и присваиваем ей функцию, созданную с использованием **lambda**. Эта функция проверяет, является ли число нечетным. Она принимает аргумент **x** и использует встроенную функцию **evenp**, чтобы проверить, является ли число четным, а затем инвертирует результат с помощью **not**. Таким образом, **odd** представляет собой функцию, которая возвращает **t**, если число нечетное, и **nil**, если число четное.  
  
neg (lambda (x) (>= x 0)))  
**neg (lambda (x) (>= x 0))** - Здесь мы определяем переменную **neg** и присваиваем ей функцию, созданную с использованием **lambda**. Эта функция проверяет, является ли число неотрицательным. Она принимает аргумент **x** и использует оператор **>=** (больше или равно), чтобы проверить, больше ли число или равно нулю. Таким образом, **neg** представляет собой функцию, которая возвращает **t**, если число неотрицательное, и **nil**, если число отрицательное.  
  
далее

(setq result-odd (func odd numbers))

**(setq result-odd (func odd numbers))** - Эта строка кода использует функцию **setq** для вызова функции **func** с предикатом **odd** (который проверяет, является ли число нечетным) для списка **numbers**. Результат этого вызова сохраняется в переменной **result-odd**. Таким образом, мы применяем функцию **func** к списку **numbers** с предикатом **odd**, чтобы получить результат.

(format t "Результат с предикатом для нечетных: ~a~%" result-odd)

**(format t "Результат с предикатом для нечетных: ~a~%" result-odd)** - В этой строке мы используем функцию **format**, чтобы вывести результат выполнения функции на экран. **format** позволяет форматировать вывод. Мы указываем **"Результат с предикатом для нечетных: ~a~%"** как строку форматирования, где **~a** будет заменено значением переменной **result-odd**. Таким образом, в выводе мы увидим строку "Результат с предикатом для нечетных:" и результат выполнения функции, который будет вставлен вместо **~a**.

Далее

(setq result-neg (func neg numbers))  
**(setq result-neg (func neg numbers))** - Эта строка кода использует функцию **setq** для вызова функции **func** с предикатом **neg** (который проверяет, является ли число неотрицательным) для списка **numbers**. Результат этого вызова сохраняется в переменной **result-neg**. Таким образом, мы применяем функцию **func** к списку **numbers** с предикатом **neg**, чтобы получить результат.  
  
(format t "Результат с предикатом для неотрицательных: ~a~%" result-neg)

1. **(format t "Результат с предикатом для неотрицательных: ~a~%" result-neg)** - В этой строке мы используем функцию **format**, чтобы вывести результат выполнения функции на экран. **format** позволяет форматировать вывод. Мы указываем **"Результат с предикатом для неотрицательных: ~a~%"** как строку форматирования, где **~a** будет заменено значением переменной **result-neg**. Таким образом, в выводе мы увидим строку "Результат с предикатом для неотрицательных:" и результат выполнения функции, который будет вставлен вместо **~a**.