UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 1

INTEGRANTES

Torres Valencia Kevin Jair - 318331818
Aguilera Moreno Adrián - 421005200
Rivera Silva Marco Antonio - 318183583

PROFESORA

Karla Ramírez Pulido

AYUDANTES

Alan Alexis Martínez López
Manuel Ignacio Castillo López
Alejandra Cervera Taboada

ASIGNATURA

Lenguajes de Programación

1 de septiembre de 2022

Pregunta 1

Elige 4 lenguajes de programación (uno por cada paradigma), e indica para cada uno de ellos el año de creación, paradigma al que pertenece y principales características.

Aqui pueden comenzar a poner sus respuestas.
Prueba de respuesta.

Pregunta 2

LISP es considerado el primer lenguaje de programación funcional. Y esta basado en el Cálculo-λ que fue desarrollado por Alonzo Church.

a) Investiga y explica brevemente qué elementos del Cálculo-λ están presentes en LISP e indica por qué crees que pueden usarse en un lenguaje de programación.

▷ Estos elementos[1] realizan procedimientos principalmente con funciones, aunque una notable diferencia sería que en el cálculo-λ no se trabaja con algún tipo de datos, pues estos no existen, y en LISP tenemos tipos de datos hetereogéneos como los atómos. Los atómos podrían biyectarse con la aplicación a términos de una función lambda (sin embargo no son exactamente igual), por ejemplo

$$(\lambda_x \cdot x)\ a = a$$

puede equivaler al atómo a en LISP.

Una característica que puede ser común entre el cálculo-λ y LISP es la recursión, pues mientras en LISP se utiliza principlamente la recursión para operar sus funciones, en el cálculo-λ se utilizan las β-reducciones (estas muchas veces son recursivas con algunas reglas ya definidas).

La característica de usar paréntesis en su sintáxis es común, sin embargo, esta no se ve reflejada en su semántica. Pues en LISP representan listas y en el cálculo-λ representa un orden jerárquico en el que se aplican las funciones base.

En ambos modelos de computo se pueden tener funciones sin aridad definida[2]. También se sabe que LISP trabaja propiamente con representaciones λ's.

◁

b) Menciona cuáles de estos elementos están presentes en el lenguaje de programación JAVA. ¿Acaso estos elementos estaban en las primeras versiones del lenguaje? De no ser así ¿porqué crees que fueron añadidos?

▷ En JAVA como en muchos otros lenguajes, podemos realizar recursión (tal vez no siempre sea lo más eficiente).

A partir de la versión 8 de JAVA se añadieron λ's al lenguaje, mencionaremos 2 razones importantes para que esto haya pasado.

- Para que más personas se interesen en el lenguaje. El conjunto de personas que programan tal vez no es el más grande comparado con la humanidad, sin embargo, entre este subconjunto de personas existen otros tantos, por ejemplo, hay personas que sólo programan de manera funcional, algunos solo hacen orientación a objetos, y otros pocos son fan de la programación lógica (y luego están los otros tipos de lengujes dentro de ambos paradigmas). La intersección no es vacía, pero el que JAVA tenga λ's hace que el lenguaje sea más atractivos para quienes gustan de manejarlas.
- Facilitan el uso de funciones cuando no hay necesidad de usarlas múltiples veces y su cuerpo es pequeño. Normalmente se usan como funciones anónimas (aunque son equivalentes, no son necesariamente iguales a las λ's en cuanto a poder computacional).

◁

[1] Cálculo-λ y LISP.
[2] Multiples parámetros.

Pregunta 3

Describe las principales características de los distintos paradigmas de programación (Estructurado, Orientado a Objetos, Funcional y Lógico) y da a 2 ejemplos de lenguajes de programación de cada paradigma.
▷ Para esta pregunta, explicamos cada parádigma por separado, esto es

Parádigma Imperativo

Describe la programación en términos del estado del programa y sentecnias que cambian dicho estado. Se le debe explicar el como hacer las cosas.

- **Orientado a objetos.** Las principales características en las orientación a objetos son; el polimorfismo (diseñar objetos que compartan comportamientos, y los puedan heredar), encapsulamiento (la información referente a un objeto esta contenida en este y puede no ser accesible al exterior), reutilización de código (modularizar nos permite evitar duplicar código), los conceptos de clases y objetos (nuestros objetos tendrán comportamientos definidos en clases).

  Ejemplos de lenguajes orientados a objetos:

  1. C++.
  2. JAVA.

- **Estructurado.** Las principales características en los lenguajes estructurados son; la sintaxis (orden y reglas del lenguaje), la semántica (interpretación y significado de la sintaxis), la pragmática (modo en que el contexto o situación le entrega interpretación al significado).

  Ejemplos de lenguajes estructurados:

  1. C.
  2. PASCAL.

Parádigma Declarativo

Este paradigma de programación esta basado en el desarrollo de programas "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el que requiera y no el como hacer.

- **Funcional.** Las principales características de la programación funcional son; semántica limpia (no hay ambigüedades en el significado de las funciones, almacenamiento implícito de datos), inexistencia de efectos colaterales (no hay asignación a variables globales), utilización del cálculo-λ (independencia fuera de la función).

  Ejemplos de lenguajes funcionales:

  1. Racket.
  2. Haskell.

- **Lógico.** Las principales características de los lenguajes en el paradigma lógico son; reglas (conjunto de proposiciones lógicas escritas como clausulas de Horn), hechos (Expresiones atómicas que verifican un predicado sobre algunos entes que puede procesar el mismo), consultas (proposiciones que se verifican para algún valor de verdad), recursión.

  Ejemplos de lenguajes lógicos:

  1. GODEL.
  2. PROLOG.

$\triangleleft$

Pregunta 4

¿Cuál de los paradigmas de lenguajes de programación, es el más adecuado para resolver los siguientes problemas? Justifica en cada caso.

a) Se requiere desarrollar un sistema que simule un modelo de sociedad de organización de termitas. Este sistema se compone, de manera general de: un espacio que las termitas recorrerán y en el cuál se encuentran astillas esparcidas, las termitas siguen las siguientes reglas de comportamiento.

- Caminan aleatoriamente hasta encontrar una astilla.
- Si la termita se encuentra cargando una astilla, la suelta y continúa caminando aleatoriamente.
- Si la termita no está cargando una astilla la toma y continúa caminado aleatoriamente con la astilla

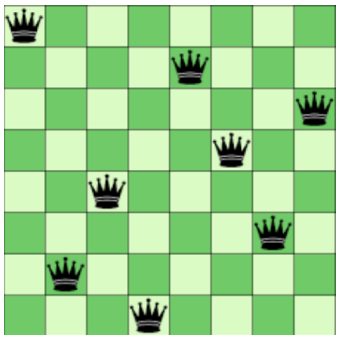
Debido a las especificaciones del problema, determinamos que para este caso, el paradigma que más nos conviene usar para resolver el problema es el paradigma de programación orientada a objetos por las siguientes razones:

- Se menciona en el primer comportamiento que las termitas caminan aleatoriamente hasta encontrar una astilla, por lo se puede entender que hay varias termitas a la vez en el modelo, entonces sería mucho más sencillo implementar objetos e instancias de las termitas.
- Se está describiendo como tal el comportamiento de las termitas, por lo que se puede controlar su acciones por medio de iteraciones mucho más facil que con recursión.

b) El Problema de las Ocho Reinas consiste en acomodar ocho reinas de ajedrez en un tablero, sin que ninguna de éstas se ataque entre sí. Una reina, puede atacar (a) de forma vertical, (b) de forma horizontal y (c) en diagonal. Usando estas reglas, indicar si el siguiente tablero es una solución al problema de las ocho reinas.

Este problema tiene varias formas de solucionarse, sin embargo la más efectiva que encontramos, es por medio del backtracking, sin embargo, el backtracking es mucho más efectivo con lenguajes de programación declarativos, ya sean lógicos o funcionales, pues como se mencionó en clase, no hay recursión más pura que la de un lenguaje de programación declarativo.

c) Escribir un programa que indique si un número natural es primo. Un número es primo.



Para este problema creemos que depende del tamaño del número natural que tenemos que queremos determinar si es primo, esto debido a que si el número es demasiado grande, puede que usar un lenguaje de programación declarativo genere problemas ya que la recursividad puede hacer que la pila de ejecución llegue a su límite, sin embargo con números no tan grandes puede ser muy efectivo, además de que su implementación es intuitiva y elegante.

También se pueden usar lenguajes de programación imperativa (ya sea estructurado o orientado a objetos), ya que por medio de iteraciones se puede resolver el problema, sin embargo puede que en algunos casos sea más lento que la recursión, sin embargo, si el número que queremos determinar si es primo es demasiado grande puede tener un desempeño más estable.

d) Dado un archivo con los primeros 10,000 dígitos de π, contar todas las apariciones de números primos presentes en él.

Debido al tamaño de la entrada, determinamos que la mejor opción es una busqueda con un lenguaje de programación imperativa, pues se puede controlar mejor la busqueda con iteraciones, además de que no tenemos que preocuparnos por la pila de ejecución. Para este caso puede ser mejor el estructurado sobre el orientado a objetos ya que no tenemos la necesidad de usar objetos (considerando que no tenemos que importar el archivo como objeto).

Pregunta 5

Investiga que significan los siguientes conceptos en un lenguaje de programación. Y elabora un pequeño ejemplo ocupando como base al lenguaje de programación HASKELL.

- Sintaxis.

  Se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

  En HASKELL, los nombres de las funciones sólo pueden contener caracteres normales, es decir, letras, dígitos, comillas y subrayados. El primer carácter de un identificador de función no puede ser una letra mayúscula ni un dígito. Normalmente se emplea la notación prefija, sin embargo, es posible usar notación infija al usar acentos franceses, ejemplo de esto es 8 ‘div‘ 3.

- Semántica.

  Descripción de los comportamientos que resultan de la ejecución de un programa o pieza de software en particular.

  En HASKELL, se tiene una evaluación perezosa que permite dar significados o valores a las funciones hasta que así se requiera.

- Idioms (Convenciones de programación).

  Son costumbres en la programación que son usadas por un grupo (comunidad) de desarrolladores con frecuencia.

  En HASKELL, tenemos que normalmente se usan x:xs para representar una lista, empleamos las letras a, b, c para valores numéricos (también x, y o n, m, k). Se suela usar $x' = x + y$ (donde basta con que y sea del tipo de x), etc.

- Bibliotecas.

  Son un conjunto de herramientas ya construidas y que muchas veces trae por omisión el lenguaje, en otros casos hay que darle la referencia a estas (cargarlos).

  En HASKELL, tenemos algunas como Math, Eq, etc.

Pregunta 6

A partir de la siguiente función, crea una firma indicando el tipo de entrada que recibirá la función y el tipo de salida que se obtendrá de la función. Asigna un nombre mnemotécnico (es decir que se autodescriba) para la misma. Justifica tu respuesta.

```
(define (foo a b)
(cond
[(> (car a) (car b)) (sub1 (foo (cdr a) (cdr b)))]
[else 0]))
```

Aqui pueden comenzar a poner sus respuestas.

Pregunta 7

Calcula el resultado de las siguientes funciones en el lenguaje de programación Racket y muéstralo. Posteriormente realiza tu propia implementación de cada función.

a) (second '(1 7 9 4 5 6))

Aqui pueden comenzar a poner sus respuestas.

b) (append '(Bue) '(n semestre))

Los resultados de la función (second '(1 7 9 4 5 6)) son los siguientes:

```
Welcome to DrRacket, version 8.6 [cs].
Language: plai, with debugging; memory limit: 128 MB.
> (second (list 1 7 9 4 5 6))
7
>
```

Los resultados de la función (append '(Bue) '(n semestre))

```
Welcome to DrRacket, version 8.6 [cs].
Language: plai, with debugging; memory limit: 128 MB.
> (append '(Bue) '(n semestre))
'(Bue n semestre)
>
```

Nuestra propia implementación de append es esta:

```
;; Second
(define (second-chido list)
  (first (rest list)))

;; otra forma
(define (second-chido2 list)
  (list-ref list 1))
```

Nuestra propia implementación de append es esta:

```
;; Append
(define (append-chido listal lista2)
  (cond
    [(empty? listal) lista2]
    [(empty? lista2) listal]
    [else (cons (first listal) (append-chido (rest listal) lista2))]))
```

Pregunta 8

Dibuja un mapa mental que muestre las fases de generación código ejecutable, sus principales características y elementos involucrados..

Aqui pueden comenzar a poner sus respuestas.