LENGUAJES DE PROGRAMACIÓN

Semestre Enero-Mayo de 2018

TAREA 1:

Conceptos sobre Lenguajes de Programación

Forma de trabajo:

Equipo de 2 integrantes

Forma de entrega:

Examen individual en clase y documento del equipo en Blackboard.

Fecha de entrega:

1a parte (durante la clase del martes 16 de enero de 2018) y 2a parte (martes 23 de enero de 2018)

Integrantes:

Sergio Lopez Madriz A01064725

Hector Hernandez Morales A00816446

PRIMERA PARTE:

Repaso y evaluación de conceptos previos (Individual - 30 puntos)

Repasa los conceptos sobre teoría de lenguajes vistos en tu clase de Matemáticas Computacionales, incluyendo los temas de: Autómatas Finitos Deterministas y No deterministas, Expresiones Regulares, Gramáticas Regulares, Gramáticas Libres de Contexto, Autómatas de Pila, Analizadores Sintácticos Descendentes y Ascendentes, y Máquinas de Turing. Asegúrate de tener claras las ventajas y desventajas de cada una de estas herramientas para la representación de lenguajes. Este punto de la tarea será evaluado a través de un examen rápido de diagnóstico en la fecha establecida.

SEGUNDA PARTE:

Lectura y reflexión sobre conceptos básicos (En equipo) Crear el documento Word M\_Tarea1.docx (donde sustituyan la M con sus matrículas) con los resúmenes solicitados abajo en páginas con márgenes normales y Font Arial de tamaño 12. Resúmenes que no cubran todo lo leído o no cumplan con la extensión solicitada (+/- un cuarto de página) serán penalizados. El documento debe ser subido a Blackboard por uno solo de los integrantes del equipo.

PROBLEMA 1 (20 puntos)

Leer las secciones

1.1 What is a Programming Language?

Un lenguaje de programación es definido como “una notación para comunicar a una computadora lo que queremos que realice”. Pero desde antes se programaba por conexiones directas a las cuales no les podemos llamar lenguajes.

John von Neumann fue quien propuso que las computadoras no se “programaran” por conexiones directas para realizar solo una cosa. Sino que se dieran códigos con instrucciones para determinar las acciones por una unidad de procesamiento central. Aqui nace ensamblador, pero el ser dependIente de maquina, tener poca abstracción y dificultad de escribir y entender propicio al desarrollo al desarrollo de lenguajes con mayor abstracción, lo que genera código más entendible y sencillo de escribir.

Muchas estructuras son usadas repetidas veces, como asignaciones, repeticiones, selecciones, y en ningún momento han tenido que ver mucho con la maquina, sino más bien con las instrucciones, estas se hicieron más comprensibles y sencillas.

Actualmente los programas son relativamente independientes, cada vez hay más abstracción y nuevas arquitecturas, entre otros cambios. Los lenguajes no necesitan basarse en un modelo solo deben describir el proceso de computo en general. Está realizacion conlleva al nuevo significado de lenguajes de programación: Sistema notacional para describir cálculos en una forma legible para la máquina y humano.

La tesis de Church dice que no es posible crear una máquina más poderosa que una maquina de Turing. Los cálculos no son solo operaciones matemáticas. Hay lenguajes de uso general y especifico.

Para que sea legible por máquina el lenguaje debe ser sencillo y estructurado. No depende de la máquina. Debe haber un algoritmo de traducción y este algoritmo debe ser eficiente y sencillo. Usualmente se restringe la estructura a la de un lenguaje libre de contexto.

Para que sea legible por humano debe proveer abstracciones de las acciones que ejecutara la maquina. Por lo que pueden llegar a parecer lenguajes naturales. Debido a los trabajos grandes que se hacen ahora y los equipos de trabajo, el mismo lenguaje de programación se vuelve parte del entorno de desarrollo de software impulsando una metodología.

1.2 Abstractions in Programming Languages

Las abstracciones que dan los lenguajes de programación son dos: de datos (permite la abstracción de las propiedades de los datos)y de control (permite la abstracción de propiedades que transfieren control).

Además de los tipos de abstracción hay niveles: básico (se da en la información más localizada ), estructurado (,as global que la básica) y unidad (sobre piezas completas del programa).

Abstracción por datos.

Abstracción basica: Se encarga de abstraer la representación interna de los valores comunes de de valores de datos. Las ubicaciones en memoria de valores son abstraídas con el nombre de variables, y el tipo de valor con tipo de dato y tienen nombre basado en su representación matemática. Ejem: int X;

Abstracción estructurada: la estructura de dato es el principal método para abstraer grupos de valores de datos relacionados, esto es logrado con arreglos de tipos de datos o conglomerados que representan un dato llamados tipos estructurados. Ejem: int a[10];

Abstracción por unidad aquí juntamos pedazos de código relacionado a partes específicas. Esto se llama encapsulación de datos u ocultamiento de información. Los podemos ver en módulos, *packages* y clases. Permite reusar. Se incluyen componentes (objetos ya funcionales) o contenedores (estructuras que contienen info de datos dados por el usuario). Son la base para librerias.

Abstracción de Control.

Abstracción básica: Combinan algunas instrucciones para hacer una instrucción más entendible, Una estructura básica de control es el goto que abstrae la operación de brinco. Ejem: x=x+3;

Abstracción estructurada: Se divide el programa a grupos de instrucciones que están anidadas. Ejem: *If, Case, Switch*. Se pueden usar *brackets* e indentación para agrupar también. Permite anidar más estructuras de control dentro. Aquí también se incluyen los: *while, for, loop* y el *repeat.* También existen las subrutinas, subprogramas y procedimientos. Permiten que se ejecuten secuencias de acciones como una acción. Al uso se le llama invocación o activación.

Abstracción de unidad: Se agrupa una colección de procedimientos para formar una unidad. Es similar a la abstracción unitaria de datos pero se le da énfasis a las funciones más que a los datos.

Casi todos los mecanismos de abstracción se dan para ayudar a la lectura de humanos. Un lenguaje que tiene los mecanismos para describir todos los cálculos que hace una máquina de Turing se le llama Turing completo, debe tener variables enteros, y ejecutar estatutos aritméticos y secuenciales.

1.3 Computational Paradigms

Al inicio los lenguajes imitaban las operaciones de la computadora. Y6Y normalmente era la de von Neumann. Que un lenguaje sea Turing completo da las funciones a un lenguaje del modelos de von Neumann. Hay varios tipos de paradigmas:

Imperativos (Procedurales)

La importancia es en ejecución secuencial, uso de variables y asignaciones. El tener las instrucciones en secuencia se le llama el von Neumann *bottleneck* ya que no permite el paralelismo. Un ejemplo de este lenguaje es ADA.

Funcional (aplicativo)

Se basa en la notación abstracta. Se evalúan funciones o le aplicamos funciones a valores conocidos. Tiene mecanismos de evaluacion de funcion. Es opuesto a Orientado a objetos en el sentido que se concentra en valores y funciones. Son más independientes y podemos obtener conclusiones más precisas. LISP es un ejemplo de un lenguaje funcional.

Logico (declarativo)

Se basa en simbolismo lógico. Se dan estatutos para describir los que es verdad acerca el resultado esperado. No necesita abstracciones de control ya que este se da por el mismo sistema. Solo se necesita dar las propiedades del cálculo. No se especifica una secuencia. Prolog es un lenguaje asi, requiere que los valores se den por variables. .

Orientado a objetos

Permiten el reuso y la extension de codigo. Se puede tomar como una extensión del paradigma imperativo. Una pieza grande se da como unión de fragmentos. Se tratan objetos como colecciones de memoria. Los objetos se agrupan en clases que representan objetos similares. Las clases se instancian para traer objetos. *Constructor*: asigna memoria y da valores iniciales. Los procesos son llamados métodos.

Es común que un lenguaje no se adhiera a un solo paradigma, sino que contienen características de varios paradigmas a la vez. El estilo de programación es seguir uno (o mas) paradigma(s) en un lenguaje que lo permita.

PROBLEMA 2 (20 puntos)

La siguiente página es una interesante referencia que muestra la popularidad y la tendencia de los lenguajes de programación: thttp://www.tiobe.com/tpci.htm. Después de revisar esta página, investiguen en internet, clasifiquen y reporten los primeros 50 lenguajes de programación que ahí aparecen dentro de las 4 clasificaciones vistas en clase (por generación, por nivel de abstracción, por la forma de expresar la solución a un problema y por paradigma). Identifiquen SOLO la categoría principal en la que cae cada lenguaje en cada clasificación, por ejemplo, aunque Scheme es multi-paradigma, su paradigma principal es el funcional.

Por Niveles de Abstracción

* Bajo Nivel
* Alto Nivel
  + C
  + C++
* Muy Alto Nivel
  + Java

PROBLEMA 3 (30 puntos)

Un concepto muy relevante en la teoría e implementación de lenguajes de programación es el que tiene que ver con su sistema de tipos. Investiguen y contesten las siguientes preguntas incluyendo el resultado en el documento de la tarea. Las descripciones de los ejemplos deben explicarse para ilustrar los conceptos:

1. ¿Qué es y para que se utiliza un tipo de dato? Describir un ejemplo de tipo de dato.

Es una clasificacion de datos que le dice al compilador o interpretador como se planea usar el dato.

int x; //Es un ejemplo de una variable con el tipo de dato que lo identifica como una variable que guardara numeros enteros.

2. ¿Qué es y para que se utiliza un sistema de tipos? Describir un ejemplo de uso de un sistema de tipos.

Define como el lenguaje de programación debe clasifica los valores y las expresiones en tipos asi como su utilizacion, manipulacion e interacciones.

La expresion a + 3 sera encontrada como un error debido a que los tipos no son coherentes.

3. ¿Cuáles son los lenguajes de programación fuertemente tipados? Mencionar dos lenguajes de programación fuertemente tipados (nombre).

Se llaman asi los lenguajes que no permiten violaciones a los tipos de datos.

C++

Java

4. ¿Cuáles son los lenguajes de programación débilmente tipados? Mencionar dos lenguajes de programación débilmente tipados (nombre).

Son los que permiten cambios a los tipos de datos.

PHP

Python

5. ¿En qué consiste el chequeo estático de tipos? Describir un ejemplo de cómo trabaja este tipo de chequeo.

Cuando la comprobacion de tipificacion se ejecuta durante la compilacion y no la traduccion.

Un ejemplo es C++ que verifica los tipos en la compilacion y detecta los errores de tipos antes de correr el programa.

6. ¿En qué consiste el chequeo dinámico de tipos? Describir un ejemplo de cómo trabaja este tipo de chequeo.

Está comprobacion se realiza durante la traduccion y no la compilacion.

Un ejemplo es Python que verifica el tipo de datos al momento de ser utilizados.

7. ¿En qué consiste el tipado seguro en lenguajes de programación? Describir un ejemplo de tipado seguro en un lenguaje de programación.

Es equivalente a tipado fuerte. Ya que no permitira violaciones al tipo de dato que se este manejando.

C++ que verifica el tipo de datos para que no se cometan errores que afecten al programa.

8. ¿Cuál es la diferencia entre equivalencia de tipos, compatibilidad de tipos, e inferencia de tipos en un sistema de tipos?

Equivalencia de tipos: Son iguales para el sistema.

Compatibilidad de tipos: Es la capacidad que hay entre dos tipos diferentes de interactuar juntos.

Inferencia de tipos: El tipo se determina por la estructura del dato usado.

9. ¿Qué significa coerción de tipos? Describir un ejemplo donde se aplique.

Se llama asi a la conversión de datos de un tipo a otro, debe ser soportado por el compilador.

void subrutina(float a);

int x;

subrutina((float) x);

La subrutina que utiliza float recibe un parametro X de tipo int, que convertimos a float de manera explícita.

10.¿Qué es un sistema de tipos nominal? Describir un ejemplo de cómo trabaja un lenguaje con este tipo de sistema de tipos (que no sea el de Wikipedia)

Al definir un tipo se le asigna un nombre que lo hace único, si dos tipos son iguales en todo excepto el nombre no son iguales.

Typedef Lalo int edad;

Typedef Juanita int edad;

Ambos datos son diferentes debido al nombre que tienen asignado, por lo cual a pesar de ser estructuralmente idénticos son diferentes.

11.¿Qué es un sistema de tipos estructural? Describir un ejemplo de cómo trabaja un lenguaje con este tipo de sistema de tipos (que no sea el de Wikipedia)

El tipo se identifica por las características que tiene.

Typedef Lalo int edad;

Typedef Juanita int edad;

Ambos datos se ven igual en memoria y por ser estructuralmente iguales son compatibles y se pueden intercambiar libremente.