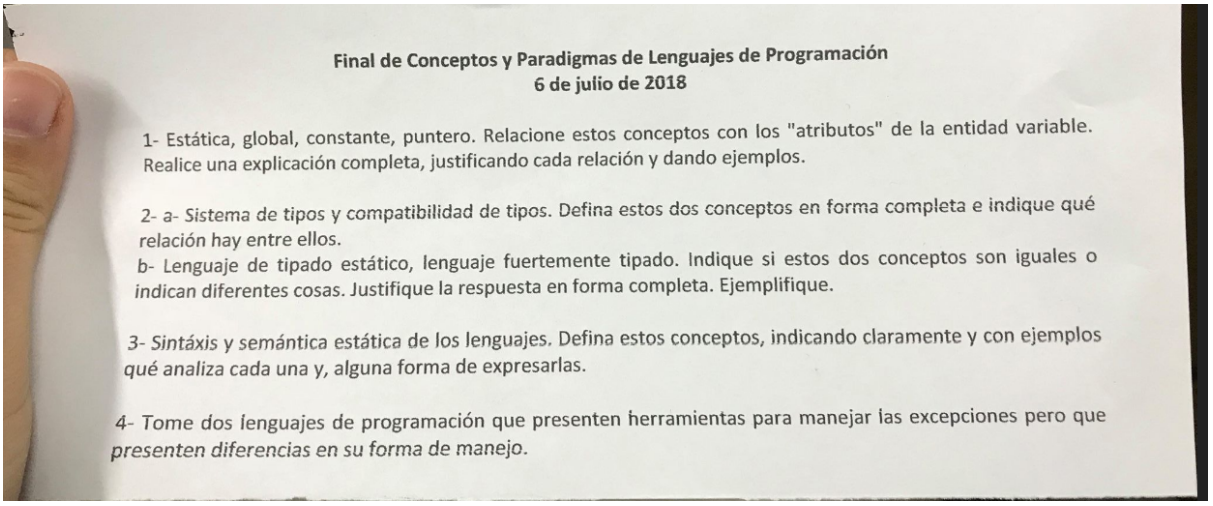
Preguntas de finales

2018-07



Final de Conceptos y Paradigmas de Lenguajes de Programación
6 de julio de 2018

1- Estática, global, constante, puntero. Relacione estos conceptos con los "atributos" de la entidad variable. Realice una explicación completa, justificando cada relación y dando ejemplos.

2- a- Sistema de tipos y compatibilidad de tipos. Defina estos dos conceptos en forma completa e indique qué relación hay entre ellos.
b- Lenguaje de tipado estático, lenguaje fuertemente tipado. Indique si estos dos conceptos son iguales o indican diferentes cosas. Justifique la respuesta en forma completa. Ejemplifique.

3- Sintáxis y semántica estática de los lenguajes. Defina estos conceptos, indicando claramente y con ejemplos qué analiza cada una y, alguna forma de expresarlas.

4- Tome dos lenguajes de programación que presenten herramientas para manejar las excepciones pero que presenten diferencias en su forma de manejo.

1 -
Variable estática: Variable que tiene un tiempo de vida (L-Value) igual a todo el programa
Variable global: variable en la cual el alcance es todo el programa
Constante: variable en la cual el R-value no cambia.
Puntero: variable en la cual su R-value apunta al L-Value de otra variable

2-
a-
Sistema de tipos: establece el tipo para cada valor manipulado. Provee mecanismos de expresión:

- Expresar tipos intrínsecos o definir nuevos tipos
- Asociar los tipos definidos con construcciones del lenguaje

Define reglas de resolución:

- Equivalencia de tipos - ¿dos valores tienen el mismo tipo?
- Compatibilidad de tipos: ¿puedo usar el tipo en este contexto?
- Inferencia de tipos: ¿Cuál tipo se deduce del contexto?

Mientras más flexible el lenguaje más complejo el sistema.

Sistema de tipos: conjunto de reglas que estructuran y organizan una colección de tipos. Tiene como objetivo que los programas sean tan seguros como sea posible.
b-
Tipado estático: ligaduras en compilación
Tipado dinámico: ligaduras en ejecución
Fuertemente tipado: cuando el sistema de tipos especifica restricciones sobre como las operaciones que involucran valores de diferentes tipos pueden operarse. De lo contrario se dice que es **Débilmente tipado**.

Tipado estático Vs fuertemente tipado: Python es un lenguaje de tipado dinámico (no estático) pero fuertemente tipado dado que asegura que no se producirán errores de tipo en

ejecución. Se deben realizar casteo de tipos (no se puede tratar un entero como si fuera un string)

3 -

- Sintaxis: Conjunto de reglas que definen como componer letras, dígitos y otros caracteres para formar los programas
- Semántica: describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático
    - Semántica estática: no está relacionado con el significado del programa,está relacionado con las formas válidas. Se las llama así porque el análisis para el chequeo puede hacerse en compilación. Para describir la sintaxis y la semántica estática formalmente sirven las denominadas gramáticas de atributos, inventadas por Knuth en 1968.Generalmente las gramáticas sensibles al contexto resuelven los aspectos de la semántica estática

```
Procedure Main;
      var y: integer;
      Procedure A;
      begin
            x:=x+1;
      end;
begin
            read (y);
            if (y>0) then A;
            write (y);
end;
end;
```

```
-#include <stdio.h>
main()
{
      int a, b;
      printf("\n Ingrese dos números:");
      scanf("%d%d", a, b);
      if (a=b) then
                  printf("\n los números  ingresados son iguales");
     else
     printf("\n los números ingresados son distintos");
}
```

Para expresar la sintaxis se puede utilizar:

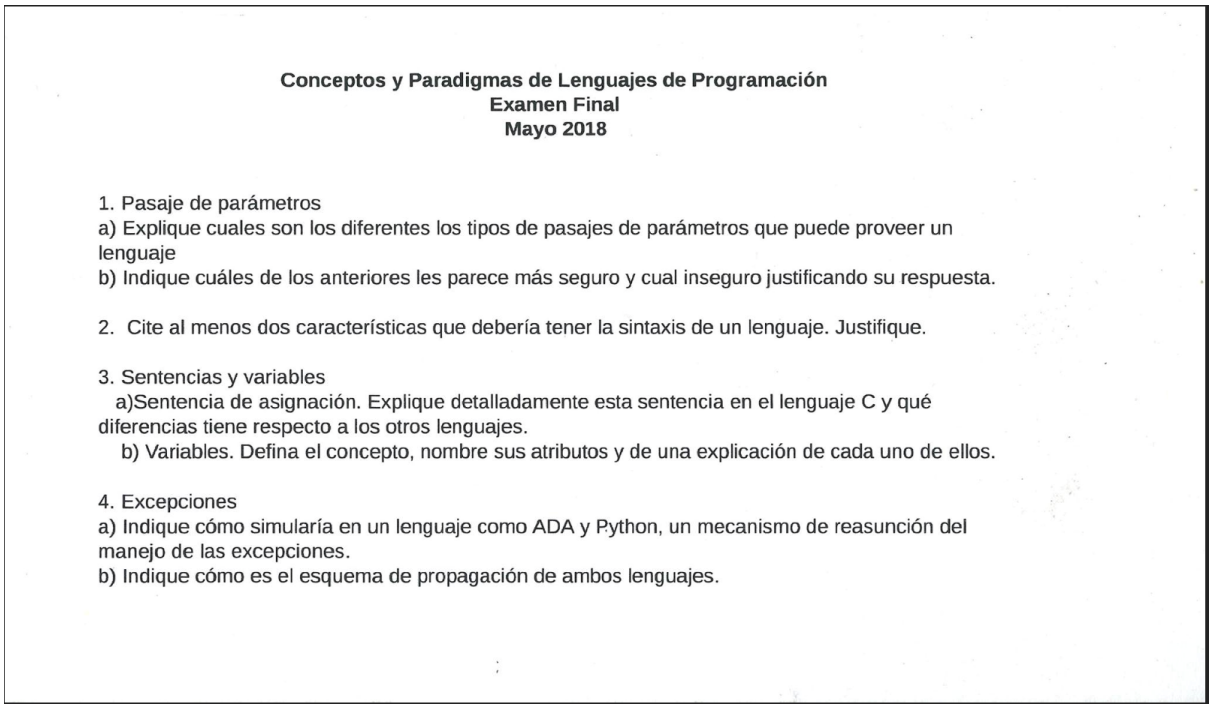
- Lenguaje Natural: Ej.: Fortran

- Gramática libre de contexto (BNF o EBNF): es un metalenguaje, utiliza meta-símbolos, define las reglas por medio de producciones. EBNF es su versión extendida.
- Diagramas sintácticos: equivalentes a BNF pero más intuitivos.

4 -

- PL1
  - Criterio de reasunción
  - Las excepciones son llamadas conditions
  - Se pueden declarar manejadores con **ON CONDITION(Nombre-manejador) Manejador**
  - El manejador puede ser una instrucción o un bloque
  - Las excepciones se alcanzan explícitamente con la palabra clave **Signal condition (Nombre-Excepción)**
- C++
  - Criterio de terminación
  - Excepciones predefinidas
  - Manejadores con bloques **try - catch**
  - Las rutinas en su interfaz pueden listas las excepciones que ella puede alcanzar
  - Se pueden capturar excepciones específicas catch (MyException) {}

2018-06



Conceptos y Paradigmas de Lenguajes de Programación
Examen Final
Mayo 2018

1. Pasaje de parámetros
a) Explique cuales son los diferentes los tipos de pasajes de parámetros que puede proveer un lenguaje
b) Indique cuáles de los anteriores les parece más seguro y cual inseguro justificando su respuesta.

2. Cite al menos dos características que debería tener la sintaxis de un lenguaje. Justifique.

3. Sentencias y variables
a)Sentencia de asignación. Explique detalladamente esta sentencia en el lenguaje C y qué diferencias tiene respecto a los otros lenguajes.
b) Variables. Defina el concepto, nombre sus atributos y de una explicación de cada uno de ellos.

4. Excepciones
a) Indique cómo simularía en un lenguaje como ADA y Python, un mecanismo de reasunción del manejo de las excepciones.
b) Indique cómo es el esquema de propagación de ambos lenguajes.

1 -

a.

- **Modo In:** El parámetro formal recibe el dato desde el parámetro real
  - Por valor:

        - se utiliza el valor del parámetro real para inicializar el local al momento de invocar a la unidad
        - El parámetro formal actúa como una una variable local de la unidad llamadora
        - Consume el tiempo para hacer la copia y el almacenamiento para duplicar el dato
        - Protege los datos de la unidad llamadora, el parámetro real no se modifica
    - Por valor constante:
        - No indica si se realiza o no la copia lo que establece es que la implementación debe verificar que el parámetro real no sea modificado
        - Requiere realizar más trabajo para realizar los controles
        - Protege los datos de la unidad llamadora, el parámetro real no se modifica
- **Modo Out:** el parámetro formal envía el dato al parámetro real
    - Por resultado:
        - el valor del parámetro formal se copia al parámetro real al terminar de ejecutarse la unidad llamada.
        - el parámetro formal es una variable local, sin valor inicial
        - consume tiempo y espacio
        - si se repiten los parámetros reales los resultados pueden ser diferentes.
        - Se debe tener en cuenta el momento en el que se evalúa el parámetro real
        - Protege los datos de la unidad llamadora. el parámetro real no se modifica en la ejecución de la unidad llamada
    - Por resultado de funciones
- **Modo In/Out:** el parámetro formal recibe el dato del parámetro real y el parámetro formal le envía el dato al parámetro real
    - Por valor/resultado:
        - Copia a la entrada y a la salida de la activación de la unidad llamadora
        - El parámetro formal es una variable local que se recibe una copia a la entrada del contenido del parámetro real y a la salida el parámetro real recibe una copia de lo que tiene el parámetro formal.
        - Cada referencia al parámetro formal es una referencia local
        - Tiene las desventajas y las ventajas de ambos.
    - Por referencia:
        - Se transfiere la dirección del parámetro real al parámetro formal.
        - El parámetro formal será una variable local a la unidad llamadora que contiene la dirección en el ambiente no local
        - Cada referencia al parámetro formal será a un ambiente no local. Esto significa que cualquier cambio que se realice en el

parámetro formal dentro del cuerpo del subprograma quedará registrado en el parámetro real.
        - El parámetro real es compartido por la unidad llamada.
        - El acceso al dato es más lento por la indirección
        - Se pueden modificar el parámetro real inadvertidamente
        - Se pueden generar alias y estos afectan la legibilidad y por lo tanto la confiabilidad, hacen muy difícil la verificación de programas.
        - Eficiente en tiempo y espacio. No se realizan copias del dato
    - Por nombre:
        - El parámetro formal es sustituido textualmente por el parámetro real. Es decir se establece la ligadura entre parámetro formal y parámetro real en el momento de la invocación pero **la ligadura del valor se difiere hasta el momento en el que se lo utiliza**
        - El objetivo es otorgar mayor flexibilidad a través de esta evolución de valor diferida.
        - Si el dato a compartir es:
            - Un único valor: igual que pasaje por referencia
            - Constante: igual que por valor
            - Si es un elemento de un arreglo puede cambiar el suscripto entre las distintas referencias.
            - Si es una expresión se evalúa cada vez

2 - La sintaxis debe ayudar al programador a escribir programas correctos sintácticamente.Establece reglas que sirven para que el programador se comunique con el procesador. Debe contemplar soluciones características tales como:

a. Legibilidad
b. Verificabilidad
c. Traducción
d. Falta de ambigüedad

Dichas características son necesarias para ayudar al programador a escribir programas correctos sintácticamente.

3 -

a - En cualquier lenguaje convencional, ej Ada, existe diferencia entre sentencia de asignación y expresión. C define la sentencia de asignación como una expresión con efectos laterales.
La mayoría de los lenguajes de programación requieren que sobre el lado izquierdo de la asignación aparezca un l-valor. C permite cualquier expresión que denote un l-valor. Ej: `++p = *q;`
b - Una variable es una abstracción de una celda elemental de memoria principal identificada por una dirección. El contenido de una celda es la representación codificada de un valor. Sus atributos son:

- Nombre: string de caracteres que se utiliza para referenciar a la variable. (identificador)
- Alcance: es el rango de instrucciones en el que se conoce el nombre.

- Tipo: define los posibles valores y operaciones a realizar
- L-Value: es el lugar de memoria asociado con la variable (tiempo de vida)
- R-Value: es el valor codificado almacenado en la ubicación de la variable

4-

a) En ADA para simular el esquema de reasunción se debe generar un bloque más interno donde puede ocurrir una excepción:

```
declare
begin -- del bloque interno
      instrucciones que pueden fallar
      exception
            manejadores
end;  --del bloque interno
…
Instrucciones a ejecutar aunque se haya levantado una exception
…
```

En Python se puede simular el esquema de reasunción utilizando bloques **try: except:**

b)

ADA:

- Se termina la unidad, bloque, paquete o tarea donde se alcanza la unidad.
- Si el manejador se encuentra en ese ámbito se ejecuta
- Si el manejador no se encuentra en ese lugar la **excepción se propaga dinámicamente. Se vuelve a levantar en otro ámbito**
- Siempre tener en cuenta el alcance, puede convertirse en anónima.

Python: (si la excepción no encuentra un manejador en su bloque “try except”

- Busca estáticamente: analiza si el try está contenido dentro de otro y si ese otro tiene un manejador. Sino:
  - Busca dinámicamente: analiza quien lo llamó y busca allí
- Si no encuentra un manejador se corta el proceso y larga el mensaje standard error

2018-02

1) Sean las siguientes dos clasificaciones de variables:
I) Estática, automática, semiautomática y dinámica.
II) Global, local, no local.

a) Indique bajo qué conceptos relacionado con variable se realizaron las clasificaciones I y II.

Estática, automática, semiautomática y dinámica son clasificaciones según el tiempo de vida (L-Value).

Global, local, no local son clasificaciones según el alcance.

b) Explique detalladamente características de cada uno de los tipos de variables enunciados en el ejemplo.

Global: Todas las referencias creadas en el programa principal.
Ejemplo: String definido en el programa principal.

Local: Todas las referencias que se han creado dentro del programa o subprograma.
Ejemplo: Integer definido en un procedimiento.

No-local: Son todas las referencias que se utilizan dentro del subprograma pero que no han sido creadas en él.
Ejemplo: Dentro de un módulo, la utilización del String que fue defino de manera global. En ese caso esa variable global, es no-local al módulo donde está siendo utilizada.

Estática: Sensible a la historia, se reserva la memoria en tiempo de compilación.
Ejemplo: Una variable global.

Dinámica: Cuando aparece la declaración y se asigna memoria dinámicamente.
• Automática: Son aquellas variable dinámicas que son locales solo a un grupo de sentencias.
Ejemplo:: Las declaradas en una función que es invocada dinámicamente.

Semidinámicas: Son variables que se alocan o desalocan por decisión del programador.
Por ej: Un arreglo dinámico.

c) Los conceptos variable estática y constante son lo mismo? Justifique.

No son lo mismo, una constante no cambia nunca su R-Value, en cambio una variable estática si puede cambiar su valor.

2) Enuncie las características de sintaxis de lenguaje
b) A que se refiere sintaxis abstracta, concreta y pragmática? Ejemplifique cada uno.
Rta:
La sintaxis debe ayudar al programador a escribir programas sintácticamente válidos.
Establece reglas que sirven para que el programador se comunique con el procesador
Características

Tipos de sintaxis:

- Abstracta: se refiere básicamente a la estructura
- Concreta: se refiere básicamente a la parte léxica
- Pragmática:se refiere básicamente al uso práctico

Ejemplo Abstracta y concreta:

```
while (x != y)  while x <> y do
{               begin
}               end
```

Ejemplo sintaxis pragmática:

- <> más legible que !=
- en C y Pascal {} o begin-end pueden omitirse si el bloque está compuesto por una única sentencia. Pragmáticamente puede concluirse que si se necesitara agregar una sentencia debe agregarse el begin-end o las {}

3) Determine las características que debe de tener toda excepción.

- ¿Cómo se maneja una excepción y cuál es su ámbito?
- ¿Cómo se alcanza una excepción?
- ¿Cómo especificar la unidades (manejadores de excepciones) que se han de ejecutar cuando se alcanza las excepciones?
- ¿A dónde se cede el control cuando se termina de atender las excepciones?
- ¿Cómo se propagan las excepciones?
- ¿Hay excepciones predefinidas?

b) Definir las excepciones en ADA o Python.

ADA: Utiliza el criterio de Terminación. Cada vez que se produce una excepción se termina el bloque, procedimiento, paquete o tarea donde se levantó y se ejecuta el manejador asociado.

Las excepciones se declaran en la zona de definición de variables, colocando la palabra Exception, e: exception. El alcance de una excepción es el mismo alcance de las variables. Las excepciones se alcanzan explícitamente con la palabra clave raise.

```
RAISE (NombreExcepcionALanzar)
```

Los manejadores se encuentran en el final de cuatro diferentes unidades de programa: Bloque, Procedimiento, Paquete o Tarea. Forma de definirlos:

```
…
Begin
….
exception
      when nomExcep1 => Manejador1
      when  nomExcep2 => Manejador2
      …..
      when OTHERS  => Manejador (opcional);
End;
```

Ada ya tiene cuatro excepciones predefinidas con su manejador asociado. Estas pueden ser:

- **Constraint_Error:** Falla un chequeo en tiempo de ejecución sobre alguna limitación (por ejemplo: fuera del límite en un arreglo.)

- **Program_Error:** Falla un chequeo en tiempo de ejecución sobre alguna regla del lenguaje.
- **Storage_Error:** Falla un chequeo de tiempo de ejecución sobre la disponibilidad de memoria (por ejemplo por alocación dinámica.).
- **Tasking_Error:** Falla un chequeo de tiempo de ejecución en el sistema de task (por ejemplo en el manejo y la comunicación de tareas)

En algunas ocasiones, se necesita levantar de nuevo la excepción colocando solamente la palabra raise SIN NOMBRAR la excepción.

4) Diferencie las características de un lenguaje con paradigma Funcional e Imperativo.

Imperativo: compuesto de sentencias + secuencias de comandos
Funcional: los programas se componen de funciones.

- Vista uniforme de programa y función
- Tratamiento de funciones como datos
- Liberación de efectos colaterales
- Manejo automático de memoria
- Ineficiencia de ejecución
- Define un conjunto de datos
- Provee un conjunto de funciones primitivas
- Provee un conjunto de formas funcionales
- Requiere de un operador de aplicación

Característica	Enfoque imperativo	Enfoque funcional
Enfoque del programador	Cómo realizar tareas (algoritmos) y cómo realizar el seguimiento de cambios de estado.	Información deseada y transformaciones necesarias
Cambios de estado	Importante	Inexistente
Orden de ejecución	Importante	Baja importancia
Control del flujo primario	Bucles, elementos condicionales y llamadas a funciones (métodos).	Llamadas a funciones, incluyendo la recursividad.
Unidad de manipulación primaria	Instancias de estructuras o clases	Funciones como recopilaciones de datos y objetos de primera clase