Procesadores de lenguajes

Ingeniería Informática
Especialidad de Computación
Tercer curso, segundo cuatrimestre

Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

Curso académico: 2024 - 2025

TRABAJO DE PRÁCTICAS

1. Introducción

- **Competencias**
  - El presente trabajo de prácticas pretende desarrollar las siguientes "competencias de la asignatura":
    - CU1. Acreditar el uso y dominio de una lengua extranjera.
    - CTEC2. Capacidad para **conocer** los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento **léxico**, **sintáctico** y **semántico** asociadas, y saber **aplicarlas para la creación, diseño y procesamiento de lenguajes.**
- **Objetivo**
  - Se debe utilizar **flex** y **bison** para elaborar un intérprete de código
    - *interpreter.exe*

- Descripción de los apartados:
  2) Elaboración y entrega del trabajo
  3) Características del lenguaje de pseudocódigo
  4) Control de errores
  5) Modos de ejecución del intérprete
  6) Documentación del trabajo
  7) Criterios de evaluación

2. Elaboración y entrega

- **Modo de realización del trabajo**
  - El trabajo se podrá realizar de forma individual o por parejas.
- **Plazo de entrega**
  - Hasta las 9:00 horas del lunes 23 de junio de 2025.
- **Modo de entrega**
  - Un fichero comprimido deberá ser "subido" a la tarea de la plataforma de "*moodle*".
  - Dicho fichero comprimido deberá contener:
    - Documentación del trabajo (véase el apartado nº 6)

- Fichero de flex
- Fichero de bison
- Ficheros de C++ (“.cpp”, “.hpp”)
- Fichero makefile
- Ficheros de ejemplo de pseudocódigo con la extensión “.p”

3. Características de lenguaje de pseudocódigo

a) Componentes léxicos o *tokens*

- **Palabras reservadas**
  - **Sentencias de control**
    - ✓ *read, read_string*
    - ✓ *print*
    - ✓ *if, then, else, end_if*
    - ✓ *while, do, end_while*
    - ✓ *repeat, until, for, end_for, from, step, to*
    - ✓ *switch, case, default, end_switch*
  - **Manejo de la pantalla**
    - ✓ *clear_screen, place*
  - **Funciones predefinidas**
    - ✓ *sin, cos, sqrt, log, log10, exp, sqrt, integer, abs.*
  - **Constantes predefinidas**
    - ✓ *pi, e, gamma, phi, deg*
  - **Operador matemático**
    - ✓ mod
  - **Constantes y operadores lógicos**
    - ✓ *true, false*
    - ✓ *or, and, not*
  - **Observaciones**
    - ✓ No se distinguirá entre mayúsculas ni minúsculas.
    - ✓ Las palabras reservadas no se podrán utilizar como identificadores.
- **Identificadores**
  - Características
    - ✓ Estarán compuestos por una serie de letras, dígitos y el símbolo de subrayado “_”.
    - ✓ Deben comenzar por una letra
    - ✓ No podrán acabar con el símbolo de subrayado, ni tener dos subrayados seguidos.
  - Identificadores válidos:
    - ✓ *dato, dato_1, dato_1_a*
  - Identificadores **no** válidos:
    - ✓ *_dato, dato_, dato__1*
  - No se distinguirá entre mayúsculas ni minúsculas.
- **Número**
  - Se utilizarán números enteros, reales de punto fijo y reales con notación científica.
  - Todos ellos serán tratados conjuntamente como números.

- **Cadena**
  - Estará compuesta por una serie de caracteres delimitados por comillas simples:
    - ✓ *'Ejemplo de* cadena'
    - ✓ 'Ejemplo de cadena con *salto de línea* \n *y tabulador* \t'
  - Deberá permitir la inclusión de la comilla simple utilizando la barra (\):
    - ✓ *'Ejemplo de* cadena *con* \' *comillas*\' *simples'*.
  - **Notas**:
    - ✓ Las comillas exteriores no se almacenarán como parte de la cadena.
    - ✓ Por tanto, al escribir una cadena,
      - ➢ no se deberán imprimir las comillas exteriores
      - ➢ pero sí se deberán interpretar los comandos \n, \t y \'.
- **Operador de asignación**
  - asignación: :=
- **Operadores aritméticos**
  - suma: +
    - ✓ Unario: + 2
    - ✓ Binario: 2 + 3
  - resta: -
    - ✓ Unario: - 2
    - ✓ Binario: 2 - 3
  - producto: *
  - división: /
  - división entera: //
  - módulo: ***mod***
  - potencia: ^
- **Operador alfanumérico:**
  - concatenación: ||
- **Operadores relacionales de números y cadenas:**
  - menor que: <
  - menor o igual que: <=
  - mayor que: >
  - mayor o igual: >=
  - igual que: =
  - distinto que: <>
  - Por ejemplo:
    - ✓ si ***A*** es una variable numérica y ***control*** una variable alfanumérica, se pueden generar las siguientes expresiones relacionales:

```
(A >= 0)
(control <> 'stop')

valor := (a > b)
if (valor = true) then … end_if
if (valor <> false) then … end_if
```

- **Operadores lógicos**
  - disyunción lógica: *or*
  - conjunción lógica: *and*
  - negación lógica: *not*
    - ✓ Por ejemplo:
      (A >= 0) and not (control <> 'stop')
- **Comentarios**
  - De varias líneas: delimitados por el símbolos (* y *)

    ```
    (* ejemplo
       de comentario
       de varias líneas
    *)
    ```
  - De una línea
    - ✓ Todo lo que siga al carácter # hasta el final de la línea.
      # ejemplo de comentario de una línea
    - ✓ Nota:
      - ➢ Se debe eliminar la opción de terminar un programa cuando el símbolo "#" aparece al principio de una línea.
- **Punto y coma: ;**
  - Se utilizará para indicar el fin de una sentencia.

b) Sentencias

- **Asignación**
  - ***identificador*** *:= expresión numérica*
    - ✓ Declara a ***identificador*** como una variable numérica y le asigna el valor de la expresión numérica.
    - ✓ Las expresiones numéricas se formarán con números, variables numéricas y operadores numéricos.
  - ***identificador*** *:= expresión alfanumérica*
    - ✓ Declara a ***identificador*** como una variable alfanumérica y le asigna el valor de la expresión alfanumérica.
    - ✓ Las expresiones alfanuméricas se formarán con cadenas, variables alfanuméricas y el operador alfanumérico de concatenación (||).
- **Lectura**
  - ***read*** *(identificador)*
    - ✓ Declara a ***identificador*** como variable numérica y le asigna el número leído.
  - ***read_string*** *(identificador)*
    - ✓ Declara a ***identificador*** como variable alfanumérica y le asigna la cadena leída (sin comillas).
- **Escritura**
  - ***print*** *(expresión numérica)*
    - ✓ El valor de la expresión numérica es escrito en la pantalla.
  - ***print*** *(expresión lógica)*

- ✓ El valor de la expresión lógica es escrito en la pantalla.

- ***print*** *(expresión alfanumérica)*
  - ✓ La cadena (sin comillas exteriores) es escrita en la pantalla.
  - ✓ Se debe permitir la interpretación de comandos de saltos de línea (\n) y tabuladores (\t) que puedan aparecer en la expresión alfanumérica.

    print*('\t Introduzca el dato \n');*

    Introduzca el dato

- **Sentencias de control**[1]
  - Sentencia *condicional* simple

    if *condición*
    then *lista de sentencias*
    end_if
  - Sentencia *condicional* compuesta

    if *condición*
    then *lista de sentencias*
    else *lista de sentencias*
    end_if

  - Bucle "***while***"

    while *condición* ***do***
    lista de sentencias
    end_while

  - Bucle "***repeat***"

    repeat
    lista de sentencias
    until *condición*

  - Bucle "***for***"

    for *identificador*
    from *expresión numérica 1*
    to *expresión numérica 2*
    [step *expresión numérica 3]*
    do
    lista de sentencias
    end_for
    - ✓ Notas
      - ➢ El ***paso*** (**step**) es opcional; en su defecto, tomará el valor 1
      - ➢ Se debe controlar que el bucle esté bien definido, es decir, que el intervalo de valores del identificador no sea nulo o infinito.

[1] Una *condición* será una *expresión relacional* o una *expresión lógica compuesta*.

- Sentencia "***switch***"

switch *(expresión)*
case *expresión 1: lista de sentencias*
case *expresión 2: lista de sentencias*
…
[default*: lista de sentencias]*
end_switch

- **Observación**
  - ✓ La anterior sentencia de bloque, delimitada por llaves "{" y "}", ya no es necesaria y se puede suprimir.

- **Comandos especiales**
  - ***clear_screen***
    - ✓ borra la *pantalla*
  - ***place****(expresión numérica1, expresión numérica2)*
    - ✓ Coloca el cursor de la pantalla en las coordenadas indicadas por los valores de las expresiones numéricas.

- **Observación**
  - Se debe permitir que una variable pueda cambiar de tipo durante la ejecución del intérprete.

    - ✓ Ejemplo

```
# La variable dato es numérica
dato := 10;
print(dato);
…
# La variable dato se convierte en alfanumérica
read_string(dato);
print(dato);
```

- **Se valorará la ampliación del lenguaje de pseudocódigo**
  - **Ejemplos**
    - ✓ Sentencia ***do … while***
    - ✓ Operador alternativa "?"
      - ➢ a := (b>=0)? b , -b;
    - ✓ Operadores unarios: ++, --, ! (factorial), etc.
    - ✓ Nuevas funciones matemáticas: factorial, etc.
    - ✓ Operadores aritméticos y de asignación: +:=, -:=, etc.
    - ✓ Manejo de pantalla: negrita, color, etc.
    - ✓ Etc.

4. Control de errores

El intérprete deberá controlar toda clase de errores:

- **Léxicos**:
  - Identificador mal escrito.
  - Un número mal escrito.
  - Operadores mal escritos.
  - Utilización de símbolos no permitidos.
  - Etc.

- **Sintácticos:**
  - Sentencias de control más escritas.
  - Sentencias con argumentos incompatibles.
  - Etc.
  - **Observación**
    - Se valorará la utilización de "reglas de producción de control de errores" que no generen conflictos.

- **Semánticos**
  - Argumentos u operandos incompatibles.

- **De ejecución**
  - Sentencia "***para***" que pueda generar un bucle infinito.
  - Fichero de entrada inexistente o con una extensión incorrecta.
  - Etc.

5. Modos de ejecución del intérprete

El intérprete se podrá ejecutar de dos formas diferentes:

- **Modo interactivo**
  - Se ejecutarán las instrucciones tecleadas desde un terminal de texto

    interpreter.exe

    > …
  - Se utilizará el carácter de fin de fichero para terminar la ejecución:
    - Control + D

- **Ejecución desde un fichero**
  - Se interpretarán las sentencias de un fichero pasado como argumento desde la línea de comandos
  - El fichero deberá tener la extensión ".p"

    interpreter.exe example.p

- **Observaciones**
  - La gramática **no** deberá tener ningún conflicto.
  - El intérprete deberá funcionar correctamente en "ThinStation" de la Universidad de Córdoba.
  - En particular, deberán interpretar correctamente los ejemplos proporcionados por el profesor
    - menu.p
    - binario.p
    - conversion.p
    - test_switch.p

6. Documentación del trabajo

Se deberá elaborar un documento con formato "pdf" con las siguientes características:

- **Portada**
  - Título del trabajo desarrollado: Intérprete de pseudocódigo en español: interpreter
  - Nombre y apellidos de los autores
  - Nombre de la asignatura: Procesadores de lenguaje
  - Nombre de la Titulación: Grado de Ingeniería informática
  - Especialidad: Computación
  - Tercer curso
  - Segundo cuatrimestre
  - Curso académico: 2024- 2025
  - Escuela Politécnica Superior de Córdoba
  - Universidad de Córdoba
  - Lugar y fecha

- **Índice**
  - Las páginas deberán estar numeradas.

- **Introducción**
  - Breve descripción del trabajo realizado y de las partes del documento.

- **Lenguaje de pseudocódigo**
  - Se corresponde con el apartado nº 3 de este documento:
    - Componentes léxicos
    - Sentencias
  - **Observaciones**
    - Se valorará la inclusión de **ejemplos** de los componentes léxicos y las sentencias.
    - Si se ha ampliado el lenguaje de pseudocódigo con nuevas sentencias u operadores entonces se deberá indicar "**expresamente**" en este apartado.

- **Tabla de símbolos**
  - Resumen y descripción de las clases utilizadas.
  - Se pueden utilizar las figuras generadas por doxygen.

- **Análisis léxico**
  - Descripción del fichero de **flex** utilizado para definir y reconocer los componentes léxicos.

- **Análisis sintáctico:**
  - Descripción del fichero de **bison** utilizado para describir la gramática de contexto libre:
    - Símbolos de la gramática
      - ✓ Símbolos terminales (componentes léxicos)
      - ✓ Símbolos no terminales
    - Reglas de producción de la gramática
    - Acciones semánticas:
      - ✓ Se deberán describir las acciones semánticas de las producciones que generan las sentencias de control y especialmente las diseñadas para los bucles "**repeat**" y "**for**" y para sentencia "**switch**".
      - ✓ Se valorará la inclusión de gráficos explicativos.

- **Código de AST**
  - Resumen y breve descripción de las clases utilizadas.
  - Se pueden utilizar las figuras generadas por doxygen.

- **Funciones auxiliares**
  - Resumen de las funciones auxiliares que se hayan codificado.
    - Funciones matemáticas
    - Funciones alfanuméricas
    - Etc.

- **Modo de obtención del intérprete**
  - Nombre y descripción de cada directorio.
  - Nombre y descripción de cada fichero utilizado de cada directorio.

- **Modo de ejecución del intérprete**
  - Interactiva
  - A partir de un fichero

- **Ejemplos**
  - **IMPORTANTE**
    - Se deberán proponer ejemplos que muestren el uso de las sentencias y operadores del lenguaje de pseudocódigo:
      - *read, print,*
      - *read_string*
      - *if, while, repeat, for,*

        - *switch*,
        - ...
      - Se valorará la cantidad, originalidad y complejidad de los ejemplos propuestos.
    - También se deben incluir los ejemplos propuestos por el profesor:
      - menu.p
      - binario.p
      - conversion.p
      - test_switch.p

- **Conclusiones**:
  - Puntos fuertes y puntos débiles del intérprete desarrollado.
  - Reflexión final sobre el trabajo realizado.

- **Bibliografía o referencias web**
  - Se recomienda consultar el documento elaborado por el personal de la biblioteca de la Universidad de Córdoba

- **Anexos**
  - En su caso, se podrían incluir aquellos anexos que se consideren oportunos para mejorar la calidad de la documentación.

7. Criterios de evaluación

- Se utilizará la siguiente **Tabla de Evaluación**

	Necesita mejorar	Puede mejorar	Aceptable	Bien	Muy bien
Documento (40%)					
Portada					
Título					
Nombre y apellidos					
Asignatura					
Titulación y especialidad					
Curso y cuatrimestre					
Curso académico					
Escuela Politécnica Superior de Córdoba					
Universidad de Córdoba					
Ciudad y fecha					
Índice					
Páginas numeradas					
Introducción					
Breve descripción del trabajo					
Partes del documento					
Lenguaje de pseudocódigo					
Componentes léxicos					
Sentencias					
Tabla de símbolos					
Descripción					
Análisis léxico					
Descripción					
Análisis sintáctico					
Símbolos terminales					
Símbolos no terminales					
Reglas de producción					
Acciones semánticas					
AST					
Descripción					
Funciones auxiliares					
Descripción					
Modo de obtención del intérprete					
Descripción de los directorios					

	Necesita mejorar	Puede mejorar	Aceptable	Bien	Muy bien
Descripción de los ficheros					
Modo de ejecución					
Interactiva					
A partir de un fichero					
Ejemplos					
Cantidad					
Originalidad					
Complejidad					
Conclusiones					
Reflexión sobre el trabajo realizado					
Puntos fuertes y puntos débiles del intérprete					
Bibliografía o referencias web					
Corrección ortográfica					
Calidad en la redacción					
Intérprete (60 %)					
Gramática sin conflictos					
Ejecución del intérprete					
Completitud del lenguaje					
Ejemplos correctos					
Código documentado					
Control de errores					
Ampliación del lenguaje					
Asistencia a clase de prácticas					

- **Documentación:** 40 %
  - Se tendrá en cuenta lo indicado en el apartado nº 6.
  - Se valorará la inclusión de gráficos o figuras.
- **Funcionamiento del intérprete:** 60 %
  - La gramática diseñada **no** podrá tener conflictos.
    - Esta condición es **imprescindible** para aprobar el trabajo de prácticas.
  - El intérprete deberá funcionar correctamente en el entorno de **ThinStation** tanto de forma interactiva como ejecutando las instrucciones de los ficheros de ejemplo.
    - En particular, deberá ejecutar correctamente
      - los ejemplos propuestos por el profesor
      - y los ejemplos propuestos por los autores del trabajo.
  - **Se valorará**
    - La completitud del lenguaje de pseudocódigo.

- La calidad en el diseño del lenguaje y la gramática.
- El control de errores.
- La ampliación del lenguaje de pseudocódigo.
- El código elaborado deberá estar documentado con doxygen

- **Observación**:
  - Además, se valorará la asistencia a clase de prácticas y la capacidad de resolución de dificultades encontradas durante la elaboración del trabajo.