Práctica de Programación II

Cuadernillo del Alumno

Curso 2008-2009

PARA TODOS LOS ALUMNOS:

- La **asistencia** a las sesiones de prácticas es **obligatoria** (**sin excepciones**), por lo que debes informarte del calendario y procedimiento para asistir a esas sesiones en tu Centro Asociado.
- El plazo para entregar la documentación lo establece el **Tutor de Prácticas del Centro Asociado**. La calificación y las revisiones sobre la misma las realiza ese mismo tutor.
- Todos los **tutores** deben ponerse en contacto con el Equipo Docente a través del correo electrónico de los foros virtuales para obtener el acceso a la aplicación de entrega de calificaciones de prácticas.
- Todos los alumnos **deberán registrarse**, a través de su curso virtual (acceso desde Ciberuned), con el tutor o tutora con que hayan asistido a las sesiones presenciales obligatorias de prácticas, a fin de que su práctica pueda ser calificada. La aplicación de registro estará disponible en los cursos virtuales.
- Los alumnos matriculados en el **extranjero** deben ponerse en contacto con su tutor virtual en el **foro de alumnos en el extranjero**.
- **No se puede entregar** la documentación **si no se ha asistido** a las sesiones de prácticas.
- **No se puede aprobar** la asignatura sin haber aprobado la práctica.
- Las dudas sobre la práctica, pueden consultarse al Tutor, durante las sesiones de prácticas, o al Tutor Virtual en CiberUNED.
- **¡Debes consultar tu calendario de exámenes!**
- **Muy importante:** los exámenes de la asignatura podrán contener preguntas cuyo enunciado se extraiga de alguna de las partes o cuestiones de esta práctica transformándolas en un cierto grado. Será necesario contestar correctamente esas preguntas para acceder al beneficio en la calificación que pueda aportar la nota obtenida en la práctica.

Alumno:
D.N.I:
Centro Asociado:

Índice

Introducción

1. Cuestiones preliminares

1.1. Objetivos de la Práctica de Programación II

El objetivo es que el alumno ponga en práctica, de forma monitorizada, las técnicas de diseño y análisis de programas propias de la asignatura. En particular, se pretende:

- Habituar al estudiante a los modos de razonamiento analítico, inductivo y deductivo propios de la programación metódica, mediante el cálculo y verificación de algoritmos recursivos e iterativos.
- Enlazar el trabajo de diseño (propio de la asignatura) con la implementación en un lenguaje de programación
- Ilustrar el tipo de problemas que pueden ser objeto de examen, en un entorno controlado y extendido a lo largo del cuatrimestre, de forma que, a la vez que se aprenden las técnicas concretas, se vaya adquiriendo soltura con las herramientas de demostración y cálculo. Cabe notar que el detalle y la extensión con que se tratan estos problemas en la práctica es muy superior al que se requiere en las pruebas presenciales de la asignatura.

La práctica tiene un carácter formativo; por ello, es esencial que el alumno trabaje con ella antes de las sesiones presenciales; que acuda a esas sesiones y resuelva en ellas sus dudas iniciales; y, finalmente, que consulte a sus tutores (presenciales o virtuales) ante cualquier duda que le surja durante la realización de la misma.

1.2. Comentarios al enunciado

El trabajo que el Alumno debe realizar consta de dos tareas: la primera, consiste en el diseño y análisis teórico de los algoritmos a que dé lugar el enunciado, mientras que la segunda requiere la implementación y análisis empírico de dichos algoritmos.
En conjunto, se trata de:

Tarea 1: Diseño y Análisis Teórico:

1) **Especificar** una función que resuelva el problema planteado
2) **Diseñar** un algoritmo recursivo no final que satisfaga esa especificación
3) **Verificar** formalmente la corrección del algoritmo obtenido y hallar su **coste**
4) **Diseñar formalmente** un algoritmo iterativo que resuelva el problema

Tarea 2: Implementación y Análisis Empírico:

1) **Implementar** cada uno de los algoritmos obtenidos, así como un programa principal que los llame y que permita
2) **Comprobar** empíricamente la corrección del programa mediante juegos de pruebas

3) **Estimar** (empíricamente) el crecimiento del coste temporal respecto al del tamaño del problema

Por último, para facilitar el uso de este cuadernillo como guía de estudio a lo largo del curso, en cada apartado destacaremos, en un recuadro, los conocimientos previos que el alumno debería tener antes de abordarlo, en el bien entendido de que el uso del cuadernillo se presupone secuencial, por lo que dichos conocimientos no son exclusivos sino acumulados.

1.3. Aspectos importantes sobre las prácticas

Los siguientes puntos pretenden clarificar aspectos relevantes en cuanto al carácter de las prácticas y su organización.

1. Carácter de las prácticas
   - La realización de las prácticas es **obligatoria** e inexcusable. La obtención de un aprobado en las mismas es un requisito imprescindible para aprobar la asignatura
   - Las prácticas se circunscriben únicamente al curso en el que se realizan. Ello implica que **no se conservan notas de prácticas** entre cursos. Asímismo, es imprescindible que cualquier alumno que desee concurrir a la Convocatoria Extraordinaria de Fin de Carrera haya superado la práctica satisfactoriamente en el curso inmediatamente anterior al de la fecha del examen
   - La realización de las prácticas tiene carácter **individual**. Esta norma obliga al alumno a cumplimentar por sí mismo toda la documentación y a responsabilizarse de la misma
   - La documentación ha de entregarse en tiempo y forma a los profesores tutores o mediante el procedimiento que el Centro disponga para tal fin
2. Calificación de las prácticas
   - Superar las prácticas es imprescindible para aprobar la asignatura, si bien la nota obtenida en las mismas no aporta ningún porcentaje prefijado a la calificación final. Sin embargo, una buena calificación en las prácticas puede influir positivamente en la nota final de la asignatura.
   - La copia de toda o parte de la documentación de las prácticas a través de cualquier medio implicará el suspenso inmediato en la asignatura.
   - Salvo que el Centro disponga de una fecha de entrega adicional y extraordinaria, ésta será única, por lo que la calificación obtenida afectará a las dos convocatorias (ordinaria y extraordinaria, además de la Convocatoria Extraordinaria de Fin de Carrera, si tal fuese el caso) para las que habilita y a cuyo período se circunscribe
3. Recogida de la documentación
   - **El equipo docente no recogera prácticas en ningún caso**
   - La fecha de entrega, que será en principio **única** (para cada Tutor y Centro Asociado), la fijará el Tutor

- El Centro dispondrá un procedimiento adecuado a su contexto que permita la recogida de la documentación que aporten los alumnos y será responsable de su custodia hasta su entrega al Tutor para su corrección, así como su almacenamiento posterior por el período que marque la legislación vigente y su destrucción efectiva cuando dicho plazo prescriba

4. **El Alumno será responsable de**

   - Informarse sobre las fechas y procedimiento de las sesiones de prácticas y la entrega de la documentación
   - Darse de alta en la aplicación de calificación de prácticas adscrito a su Tutor presencial
   - Informarse sobre su nota de prácticas a través de la aplicación de consulta que pone a su disposición el Equipo Docente, así como sobre el plazo de revisión sobre la misma, que fijará el Tutor

5. El Tutor de Prácticas será responsable de

   - Ponerse en contacto con el Equipo Docente, a través del correo de los cursos virtuales, para darse de alta como Tutor en la aplicación de gestión de calificaciones de prácticas
   - Corregir las Prácticas conforme a las indicaciones del Equipo Docente y remitir a éste los informes sobre las mismas con la debida diligencia, de manera que no se retrase el proceso general de publicación de notas de la asignatura
   - Publicar las calificaciones obtenidas por sus alumnos en las prácticas, a través de la aplicación que el Equipo Docente pone a su disposición a tal efecto, con suficiente antelación al comienzo de los exámenes
   - Disponer un plazo y fijar un procedimiento para permitir las revisiones de las prácticas
   - Custodiar la documentación que le entreguen los alumnos en tanto dure el período de corrección y revisión de la misma y devolverla al Centro para su almacenamiento durante el período reglamentario

6. El Centro Asociado será responsable de

   - Disponer, fijar y publicar la fecha de las sesiones de prácticas
   - Recoger y custodiar la documentación que entreguen los alumnos y procurar que todas las actividades relacionadas con la celebración, corrección y calificación de las prácticas se desarrollen con la máxima diligencia posible

Enunciado

2. Enunciado de la práctica

Por favor, lea ATENTAMENTE el siguiente enunciado, ya que aporta los datos esenciales sobre el diseño que ha de realizar.

Juegos Matemáticos: Juego de la Vida

EL JUEGO DE LA VIDA (o simplemente "Vida") fue diseñado en 1.970 por el matemático británico JOHN HORTON CONWAY, que a la sazón trabajaba en la Universidad de Cambridge y que actualmente pertenece a la de Princeton. Se trata de un juego de *cero* jugadores, es decir, que se desarrolla a partir de una situación inicial mediante un conjunto de reglas y sin intervención humana posterior (el *jugador* se limita a establecer unas condiciones iniciales y a observar su *evolución* siguiendo las reglas).
La primera publicación de este juego se debe a Martin Gardner, quien la explicó en su sección "Juegos Matemáticos" de la revista *Scientific American*. John Conway obtuvo la forma canónica de las reglas del juego tras estudiar el trabajo de John von Neumann en la década de 1.940 para tratar de simplificar el complejo conjunto de reglas que von Neumann utilizó para tratar de diseñar una máquina teórica que pudiese autocopiarse. Después de muchos intentos infructuosos, Conway propuso un modelo matemático para esa máquina basado en una parrilla bidimensional de casillas cuadradas, cada una de las cuales puede o no estar *viva* en un momento determinado del juego.
La importancia del Juego de la Vida estriba en el hecho de haber abierto un nuevo campo de las Matemáticas, el de los Autómatas Celulares, que interesan a diversos campos de la Ciencia por sus propiedades *emergentes* y su capacidad de simular procesos similares a otros de interés en estas ramas del saber (como Informática, Biología, Física, etc.)
Un autómata celular es una máquina de cómputo constituida por una parrilla de casillas (también llamadas *celdas* o *células*) dispuestas en cualquier número de dimensiones, cada una de las cuales puede encontrarse en un número finito de estados tales que el actual (que notaremos t) depende del anterior ($t-1$, ya que el *tiempo* se considera discreto en los autómatas celulares) mediante un conjunto de reglas fijo e idéntico para todas las células de la parrilla tales que el estado de cada una de ellas dependerá del de sus vecinas. A cada paso del juego, se aplican las reglas para todas las células que componen la parrilla y se *actualizan* sus valores, dando lugar a lo que se conoce como una nueva *generación*.

Reglas del Juego de la Vida

El Juego de la Vida se desarrolla en una parrilla bidimensional de células cuadradas, cada una de las cuales puede encontrarse en uno de dos estados (*viva* o *muerta*). Los vecinos de una célula dada son todas sus casillas adyacentes en cualquier dirección (horizontal, vertical o diagonal). En esta versión, consideraremos que el tablero tiene límites que no están conectados entre sí (o sea, los cuatro lados externos de la parrilla representan el final del universo de la Vida).

Para cada generación:

1. Una celda *viva* con menos de dos vecinos vivos muere de soledad
2. Una celda *viva* con más de tres vecinos vivos muere por sobrepoblación
3. Una celda *viva* con dos o tres vecinos vivos sobrevive y permanece viva en la siguiente generación
4. Una celda *no viva* con (exactamente) tres vecinos vivos nacerá y estará viva en la siguiente generación

La posición inicial se establece arbitrariamente (por el jugador). Después y para cada generación, las reglas se aplican "en paralelo" (es decir, sobre el estado de una celda en la generación t no tienen influencia los de otras celdas en la propia generación t). Las reglas se pueden aplicar tantas veces como se desee para obtener nuevas generaciones.

> Se desea diseñar una función `gen` que, dado un tablero del Juego de la Vida que represente el estado de la parrilla para una generación, calcule la siguiente.

El diseño de la función que constituye la parte teórica de la práctica debe ser genérico, es decir, debe tratar todos los factores (por ejemplo, el tablero, su tamaño, los posibles estados de las celdas y el cálculo del estado de cada una de ellas para la nueva generación en función de sus vecinos, entre otros) como parámetros, sin importar cómo se instancien para resolver el problema (lo que se hará en la implementación de la práctica).

2.1. Cuestiones sobre el enunciado

1. Piense intuitivamente cómo resolvería *a mano* el problema y exponga claramente sus conclusiones
2. ¿Cuáles son los casos extremos del cálculo de vecinos? Clasifique las celdas en categorías atendiendo a sus número de vecinos.
3. (**IMPORTANTE Y RECOMENDADA**) Simplifique las reglas del enunciado de manera que necesite menos reglas para decir lo mismo. Pista: necesitará realizar alguna abstracción.
4. ¿Qué efecto tiene sobre las reglas la definición de los *vecinos*?¿Cuántos vecinos tendría una celda en una parrilla de tres dimensiones? Justifíquelo (Pista: ¿qué forma debería tener una celda en una parrilla tridimensional?)
5. ¿Qué sucedería si se considerase el tablero cerrado (o sea, si el borde superior se imaginase unido con el inferior y el izquierdo con el derecho?¿Qué efecto tendría esta consideración sobre cada generación para su cálculo?

Tarea 1: Diseño y Análisis Teórico

3. Especificación

3.1. Consideraciones sobre la especificación

Para la construcción de la especificación del problema han de tenerse en cuenta los siguientes aspectos:

- El primer paso debe ser declarar la función, esto es, darle un nombre y declarar los parámetros que recibe y el resultado que devuelve, nombrándolos y decidiendo a qué tipos de datos pertenecen.

  En nuestro caso, la *signatura* o perfil[†] de la función será la siguiente (nótese que los tipos predefinidos están en **negrita**).

$$\underline{\textbf{fun}}\ \texttt{gen} : \underline{\textbf{vector de bool}} \longrightarrow \underline{\textbf{vector de bool}} \tag{1}$$

[†]La *signatura* o *perfil* de una función consiste en el nombre de la misma y los tipos de datos de sus parámetros (pero no sus nombres). En la función `gen`, cuya signatura se da más arriba, el parámetro de entrada es un vector unidimensional de booleanos que representa la parrilla del Juego de la Vida. Si dicha parrilla tiene, por ejemplo, m posiciones horizontales por n verticales, el vector deberá contar con $m \times n$ posiciones en total. Cada posición del vector se corresponderá con una celda y tendrá valor **cierto** si está *viva* y **falso** en caso contrario. Este vector de entrada constituye la parrilla del juego en una generación mientras que el parámetro de salida (otro vector del mismo tipo que el de entrada) deberá representar la parrilla en la *siguiente* generación.

- La postcondición expresa, en forma de predicado que define un conjunto posible de estados (valores de las variables libres del programa) el resultado que se desea alcanzar, por lo que éste será el siguiente paso. Se trata de expresarla de forma precisa (recomendamos se estudien las cuestiones sobre el enunciado antes de proceder a realizar esta postcondición, en especial, la destacada como **importante**).

- La precondición define, en forma de predicado, el conjunto de datos de entrada que se consideran válidos para la función

- Ténganse en cuenta, al expresar la postcondición, todos los casos posibles, con especial atención a los *rangos vacíos* (si se usan cuantificadores) y a los casos extremos, ya que la sintaxis, usada imprecisamente, puede dar lugar a equívocos y a definiciones incorrectas. Recuérdese que la postcondición debe expresar la relación que **debe** existir entre las variables de salida y las de entrada

- Una vez decidida la postcondición, la precondición debe restringir los casos potencialmente erróneos, limitando el dominio de aplicación de los datos de entrada

- En muchos casos, debido a la naturaleza del problema (en otros, se trata tan solo de conveniencia o sencillez) la función que debemos diseñar **no nos permite** realizar directamente un diseño recursivo, por lo cual deberemos especificar otra función que sí nos lo permita (es importante convencerse de este hecho que puede presentarse en multitud de ocasiones. ¿Es éste el caso?)

3.2. Cuestiones sobre la especificación

1. Si en la postcondición de una función no se hace referencia alguna a las variables de entrada, ¿qué podemos decir de dicha función?

2. Expónganse formas alternativas de codificar el estado de las celdas. Sería positivo valorar sus pros y contras

3. Dibújese un esquema que permita discernir los vecinos de una casilla según su categoría (véase la segunda pregunta del apartado 2.1 en la página 8)

3.3. Especificación de `gen` (trabajo del alumno)

Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 1. Lógica y Especificación** especialmente, el apartado *Especificación Pre-Post. Ejemplos y Ejercicios*

En este apartado, vamos a construir la especificación de la función `gen` cuyo perfil será:

fun `gen` : **vector de bool** $\longrightarrow$ **vector de bool**

Como se enunciaba en (1) en la página 9, `gen` debe calcular la parrilla correspondiente a la siguiente generación del Juego de la Vida a partir de la actual, que recibe com entrada. Para ello, deberá aplicar las reglas descritas en el enunciado (se recomienda, una vez más, simplificarlas).
Recuérdese que la representación que estamos proponiendo es la de un **vector de una sola dimensión**, cuyas filas se almacenarían de forma consecutiva, es decir, si hubiese n columnas, las primeras n posiciones (que recomendamos se indexen de 0 a n-1) de este vector corresponderían a la primera fila, las siguientes n a la segunda fila y así sucesivamente.

Actividad 3.3. Especificación de `gen`

3.4. Más cuestiones y algunos ejercicios sobre la especificción

1. ¿Por qué en el perfil de `gen` no aparecen parámetros para expresar las filas o columnas? ¿Qué ventajas y desventajas tendría incluir como parámetros de entrada las dimensiones del tablero o parrilla (su número de filas y columnas)? Justifíquese.

2. Propóngase otra forma diferente (en lugar de un solo vector de booleanos) de representar la parrilla o *tablero* del Juego de la Vida. Razónense las ventajas e inconvenientes respecto a la representación propuesta en esta práctica

3. **ejercicio recomendado**. Especifíquese `gen` de forma modular. Para ello, deberá proponerse una o más funciones auxiliares y especificarlas completamente declarándose, además, la relación entre `gen` y sus funciones auxiliares. Sugerimos que se utilice una función `vecinos`, que calcule el número de celdas adyacentes *vivas*. Nótese que se debe definir el concepto de ADYACENCIA de manera que resulte consistente con las reglas enunciadas en 2, en la página 7.

4. Supóngase un tablero tridimensional para el Juego de la Vida. Especifíquense `gen` y `vecinos` para adaptarse a esta situación. Compare las especificaciones de `gen` y `vecinos` para dos y tres dimensiones.

5. Modifíquese la función `vecinos` para que calcule los de una casilla dada de tal manera que se consideren no sólo los adyacentes, sino todos los que estén a menos de un cierto número de casillas de distancia? (Pista: defínase una medida de distancia, si se desea, como una función auxiliar). Especifíquese `gen` utilizando `vecinos`. Especifíquese nuevamente `vecinos` para que responda a las reglas del enunciado.

3.5. Nueva especificación de `gen`

En este apartado se trata de modificar la especificación que se realizó en 3.3 en la página 10 para adaptarla de manera que utilice la función `vecinos`.
En el siguiente especio reservado, escríbase la nueva especificación para `gen`:

Actividad 3.5. Nueva especificación de `gen`

4. Diseño recursivo mediante una inmersión no final de `gen`

4.1. Diseño recursivo de `gen`

Para diseñar una función recursiva necesitamos contar con algún parámetro de entrada sobre el que se pueda establecer un subproblema, en función de cuya solución podamos resolver el problema original. El tamaño del subproblema **debe ser menor que el del problema** para que la cadena de llamadas recursivas termine. Esto significa que deberemos encontrar algún parámetro de entrada que podamos hacer decrecer a cada nueva llamada recursiva. Un natural puede ser un buen candidato, pero puede lograrse con cualquier tipo enumerado o, incluso, con combinaciones de parámetros.
La función que vamos a diseñar, `gen`, tiene como entrada un vector de booleanos, que siempre debe permanecer igual a lo largo de la cadena de llamadas recursivas, ya que sólo actúa como parámetro de lectura. Por lo tanto, necesitaremos otro parámetro que nos permita ir reduciendo paulatinamente el tamaño del problema de entrada.

4.2. Cálculo de la inmersión y llamada inicial (trabajo del alumno)

> Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 2. Recursividad** especialmente, el apartado *Técnicas de inmersión*

Hállese una inmersión (`igen`) para la función recursiva especificada en el apartado 3.5 (en la página 11). Se trata, como se sugiere en la explicación anterior, de incluir en la cabecera de la función inmersora (`igen`) un parámetro adicional a los que ya aparecían en `gen` y que permita abordar un diseño recursivo (cálculo de una recurrencia respecto a los parámetros de entrada, haciendo que alguno(s) varíen para llegar a un caso final).
Escríbase en el espacio reservado a continuación la especificación de la función inmersora `igen` así como la llamada inicial que consigue que ésta calcule lo mismo que `gen`. Explíquese la elección del parámetro de inmersión y detállese su rango admisible.

Actividad 4.2. Especificación de la función inmersora `igen` y llamada inicial

4.3. Análisis por casos

El análisis por casos de una función recursiva consiste en obtener una clasificación, dependiente de los parámetros de entrada, que decida para qué valores de éstos proporcionar una solución inmediata (que ya no requiera descomposición recursiva), y para cuáles apoyar la

solución en una nueva invocación recursiva de la propia función. En esta clasificación, el parámetro *inmersor* que hayamos elegido va a realizar la función de discriminador.
En cada caso, se trata de obtener una *protección* (o condición booleana que se debe cumplir para que se proceda a devolver la parte derecha de dicha expresión) y el resultado si ésta se abre. Ha de tenerse en cuenta, además, que todos los casos posibles para los datos de entrada estén cubiertos (o sea, que la disyunción de *todas* las alternativas debe evaluarse a **cierto**), para evitar que haya casos sin tratar.
En la práctica que nos ocupa, el caso *trivial* (es decir, el que no es recursivo, que también se conoce como caso *final*) puede tratarse del análisis de un cierto valor del parámetro natural de entrada. El caso recursivo requerirá la evaluación de una expresión sobre ese mismo parámetro en función de lo obtenido en una llamada recursiva.
Pista: puede aclarar las cosas bastante "desarrollar" o "desplegar" algunos de los términos de la expresión de la postcondición, separándolos del cuantificador, lo que, en abstracto sería tanto como decir que, para una postcondición como $\{\mathrm{R} \equiv \bigotimes \alpha \in \{a \cdot\cdot b\} \,.\, P(\alpha)\}$, se correspondería un desarrollo como $P(a) \otimes P(suc(a)) \otimes P(suc(suc(a))) \otimes P(suc(suc(suc(a)))) \otimes \cdots \otimes P(pre(b)) \otimes P(b)$, donde $\otimes$ es una operación, cuyo cuantificador asociado es $\bigotimes$, $suc(i)$ es el *sucesor* o siguiente elemento enumerado a partir de i y $pre(i)$ es un elemento cuyo sucesor es i.

4.4. Análisis por casos de la función `igen` (trabajo del alumno)

Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 2. Recursividad** especialmente, el apartado *Derivación formal de programas recursivos*

Escríbase en el siguiente espacio reservado el análisis por casos de la función `igen`, junto con el proceso de su obtención (esto es, debe justificarse cada elemento del análisis por casos: protecciones de las alternativas y resultados que se devuelven para cada una de ellas):

Actividad 4.4. Análisis por casos para `igen`

4.5. Composición algorítmica de `igen` (trabajo del alumno)

La composición algorítmica no es más que la expresión, utilizando una sintaxis algorítmica, del trabajo que hemos realizado hasta el momento, es decir, la especificación formal y el análisis por casos. En ella deben incluirse la precondición, la declaración o cabecera de la

función, la alternativa de las protecciones con sus resultados dada por el análisis por casos, y la postcondición. Para ello, utilizaremos la sintaxis descrita el texto de referencia y cuyos elementos principales para una función recursiva lineal se resumen en la figura 3.2 ([Peña93, página 53], [Peña97, pag. 61] o [Peña05, pag. 68]), con la que el alumno debería estar familiarizado antes de abordar la realización de este apartado. De hecho, sería muy conveniente que el alumno identificase y escribiese todos los componentes del código y su documentación, según aparecen en (refs a las figuras que describen el código de un aloritmo recursivo lineal), es decir $\overline{x}$, $\overline{y}$, $Q(\overline{x})$, $R(\overline{x},\overline{y})$, $s(\overline{x})$, $B_t(\overline{x})$, $B_{nt}(\overline{x})$, $triv(\overline{x})$ y $c(\overline{y},s(\overline{x}))$ (véase también el siguiente apartado). Para simplificar las explicaciones reproducimos, a continuación, la forma general de una función recursiva lineal, tal como se describe en las referencias citadas anteriormente:

$$
\begin{array}{l}
\{Q(\bar{x})\} \\
\underline{\textbf{fun}}\ f\ (\bar{x}\ :\ T_1)\ \underline{\textbf{dev}}\ (\bar{y}\ :\ T_2) \equiv \\
\quad \underline{\textbf{caso}}\ B_t(\bar{x}) \rightarrow triv(\bar{x}) \\
\quad\quad [\!]\ \ B_{nt}(\bar{x}) \rightarrow c(f(s(\bar{x})),\bar{x}) \\
\quad \underline{\textbf{fcaso}} \\
\underline{\textbf{ffun}} \\
\{R(\bar{x},\bar{y})\}
\end{array}
$$

Figura 1: Forma abstracta de una función recursiva lineal, *op cit.*

La composición algorítmica, que estamos a punto de escribir, constituye el código de la función.
En el espacio reservado que aparece a continuación, debe transcribirse la composición algorítmica de la función `igen`, que deberá incluir también la documentación (especificación pre-post) de dicha función:

Actividad 4.5. Composición algorítmica para `igen`

4.6. Verificación formal de la corrección de `igen`

Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 2. Recursividad** especialmente, el apartado *Verificación formal de programas recursivos*

Una vez obtenido el código de la función `igen`, vamos a verificar, formalmente, su corrección. Nótese que, si se hubiese derivado el análisis por casos formalmente (es decir, incluyendo la demostración de la validez de cada caso y de su mutua exclusión), sólo restaría demostrar la terminación de este algoritmo, que sería correcto por construcción.

4.6.1. Completitud de la alternativa (trabajo del alumno)

Como la función ha de estar bien definida en su dominio, hay que demostrar que el conjunto de protecciones que hemos obtenido en el análisis (véase 4.4) cubre *todos* los casos posibles para los datos de entrada recogidos en la precondición. Es decir, y siguiendo la notación de la figura 1, hay que demostrar que $Q(\bar{x}) \Rightarrow B_t(\bar{x}) \vee B_{nt}(\bar{x})$, que es el siguiente trabajo que hay que llevar a cabo. En el siguiente espacio, verifíquese la completitud de la alternativa:

Actividad 4.6.1. Verificación de la completitud de la alternativa para `igen`

4.6.2. Satisfacción de la precondición para la llamada interna (trabajo del alumno)

Para asegurar que se cumple la postcondición, la función ha de invocarse siempre en estados que satisfagan su precondición. Esto es, la propiedad que hay que demostrar se escribe formalmente como $Q(\bar{x}) \wedge B_{nt}(\bar{x}) \Rightarrow Q(s(\bar{x}))$. Hágase en el siguiente espacio:

Actividad 4.6.2. Verificación de la satisfacción de la precondición para la llamada interna de `igen`

4.6.3. Base de la inducción (trabajo del alumno)

Ha de demostrarse la satisfacción de la postcondición para los casos triviales, o sea, $Q(\bar{x}) \wedge B_t(\bar{x}) \Rightarrow R(\bar{x}, triv(\bar{x}))$.
La importancia de este punto radica en que sin la base de una inducción, ésta no supondría una demostración. El o los casos triviales son aquellos en que ya no se necesita más descomposición recursiva. Por tanto, se tienen que apoyar en propiedades o cálculos básicos, que aparecerán reflejados en esta demostración.
En el espacio siguiente, demuéstrese que se alcanza la postcondición mediante los casos triviales:

Actividad 4.6.3. Verificación de la base de la inducción para `igen`

4.6.4. Paso de inducción (trabajo del alumno)

Hay que probar que se satisface la postcondición para los casos recursivos, a partir de la hipótesis de inducción, esto es, de la suposición de la satisfacción de dicha postcondición para los datos de la llamada interna. Formalmente, hay que demostrar que $Q(\bar{x}) \wedge B_{nt}(\bar{x}) \wedge R(s(\bar{x}), \bar{y}') \Rightarrow R(\bar{x}, c(\bar{y}', \bar{x}))$, lo que debe realizarse en el espacio reservado a continuación:

Actividad 4.6.4. Demostración del paso de inducción para `igen`

4.6.5. Elección de una estructura de preorden bien fundado (trabajo del alumno).

Hay que encontrar $t : \mathcal{D}_{T_1} \longrightarrow \mathbb{Z}$ tal que $Q(\bar{x}) \Rightarrow t(\bar{x}) \geq 0$

Se trata de establecer una aplicación entre el tipo de datos de la entrada y los enteros (los naturales, de hecho, las más de las veces, pero es admisible utilizar los enteros siempre y cuando se limite el decrecimiento en este conjunto hasta un cierto mínimo que no se vaya a poder rebasar, ya que, en ese caso, serían isomorfos a los naturales) de manera que se pueda demostrar que es un preorden *bien fundado*, es decir, que no existen cadenas decrecientes infinitas.
En el espacio siguiente, propóngase una estructura de preorden bien fundado a partir de los datos de las llamadas recursivas y demuéstrese que no pueden existir cadenas decrecientes infinitas en este p.b.f.:

Actividad 4.6.5. Elección de un p.b.f.: decrecimiento de los datos en las llamadas recursivas a `igen`

4.6.6. Demostración del decrecimiento de los datos (trabajo del alumno).

En ese preorden, para las sucesivas llamadas recursivas. Esto equivale a demostrar que $Q(\bar{x}) \wedge B_{nt}(\bar{x}) \Rightarrow t(s(\bar{x})) < t(\bar{x})$. La importancia de este punto estriba en que, si el preorden es bien fundado no pueden existir *cadenas descendentes infinitas* en él, por lo que, si los datos decrecen sobre el mismo, su tamaño llegará a un caso trivial (que se corresponderá con un minimal en el preorden bien fundado, de los que podría haber varios) o, en términos prácticos y dada la estructura elegida *sobre los datos de las llamadas recursivas*, la función terminará. En el espacio siguiente, demuéstrese que los datos de cada llamada recursiva decrecen en el preorden bien fundado elegido en 4.6.5:

Actividad 4.6.6. Demostración de que los datos decrecen en el pbf elegido (punto 4.6.5) para `igen`

4.7. Estudio del coste de `igen` (trabajo del alumno)

> Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 2. Recursividad** especialmente, el apartado *Cálculo de la eficiencia en programas recursivos: tamaño del problema, medidas asintóticas, resolución de recurrencias*

Una vez que hemos demostrado que el algoritmo es correcto y termina queremos determinar la cantidad de recursos que consume para su funcionamiento. De hecho, nos interesa realizar un estudio comparativo, esto es, averiguar cuánto crece el consumo de recursos en relación con el crecimiento del tamaño de los datos de entrada. Como se va a estudiar el crecimiento, nos conviene utilizar medidas asintóticas, cuya naturaleza nos permite establecer un nivel de abstracción adecuado a la comparación que se plantea. Por último, el estudio se basará en

el reconocimiento del *caso peor*, esto es, tratamos de calcular una cota superior al crecimiento del coste. Para ello, y dado que se trata de un algoritmo recursivo, hay que plantear una recurrencia a partir del tamaño del problema y la forma de decrecimiento de éste en el *p.b.f.*, identificar los parámetros y resolverla.
En el espacio siguiente, debe calcularse el coste de la función `igen` mediante la recurrencia adecuada. Debe justificarse el valor de cada uno de los parámetros, a, b y k.:

Actividad 4.7. Cálculo del coste asintótico temporal en el caso peor de `igen`

4.8. Cuestiones sobre el diseño y verificación recursivos

1. ¿Que sucedería si la alternativa no fuese completa? ¿Y si hubiese intersección en las protecciones que definen la alternativa?
2. Proponga un caso trivial alternativo. ¿Qué consecuencias traería su uso?
3. ¿Puede optimizarse `igen`?¿De ser así, cómo se haría?

5. Transformación a recursivo final: Desplegado-plegado

5.1. Árbol sintáctico de la función recursiva (trabajo del alumno)

En este apartado vamos a transformar la función `igen`, recursiva no final, en otra recursiva final `iigen`, que será inmersión de la anterior (es decir, incluirá algún parámetro adicional y sus especificaciones se relacionarán por el valor de este parámetro en la inmersora). Para hacerlo, emplearemos la técnica del *desplegado-plegado*.

> Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 3.Diseño y verificación de algoritmos recursivos** especialmente, el apartado *Técnica de desplegado y plegado*

En el siguiente espacio, dibújese el árbol sintáctico de la llamada recursiva interior de la función no final `igen` (rama del caso no trivial):

Actividad 5.1. Árbol sintáctico de la llamada recursiva de `igen`

5.2. Substituciones aplicadas (trabajo del alumno)

Aplíquense las substituciones pertinentes según la heurística de Kodratoff, explicada y aplicada en los ejemplos dados en [Peña93, Técnica de desplegado y plegado, pag(s) 81 a 87], [Peña97, Técnica de desplegado y plegado, pag(s) 94 a 99] o [Peña05, Técnica de desplegado y plegado, pag(s) 102 a 108]. Debe reescribirse el árbol de `igen` incluyendo las substituciones realizadas en el espacio que se reserva a continuación:

Actividad 5.2. Aplicación de la heurística de Kodratoff, substituciones

5.3. Desplegado y plegado. Llamada inicial (trabajo del alumno)

En este apartado han de listarse todas las transformaciones que permiten desplegar y posteriormente plegar la función `igen` en la nueva función, recursiva final, `iigen` (inmersión de `igen`). Ha de darse, además, la llamada inicial que hace que $\texttt{iigen}(?) = \texttt{igen}(?)$.
Es necesario recalcar la importancia de modificar la precondición y la postcondición, ya que al realizar el desplegado y posterior plegado se añadirá a la función un nuevo parámetro, por lo cual la precondición y la postcondición deberán reflejarlo de forma adecuada. Asímismo, es necesario demostrar que la función que se va a desplegar y plegar cumple las propiedades que permiten hacerlo, así como no utilizar propiedades que no sean necesarias.

Actividad 5.3. Transformación por desplegado y plegado de `igen` en `iigen`

5.4. Código de la función recursiva final (trabajo del alumno)

En el espacio reservado a continuación, se debe transcribir el código de la función recursiva final `iigen` completo con la precondición y postcondición.

Actividad 5.4. Código de `iigen`

5.5. Cuestiones sobre la transformación final

1. Explique la especificación de la función recursiva final `iigen`. ¿Qué otras alternativas se pueden tomar?
2. ¿Qué quiere decir realizar la transformación *con postcondición constante*?(¿Respecto a qué parámetros es constante la postcondición?).¿Qué ventajas tiene?

6. Diseño iterativo de `genIt`

Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 3. Iteración** especialmente, el apartado *Semántica iterativa y Esquema general y verificación formal de bucles*

6.1. Cuestiones preliminares sobre el diseño iterativo

A partir de la función `igen`, cuya inmersión, `iigen`, hemos diseñado y verificado en la anterior sección, vamos a derivar un algoritmo iterativo que la utilice para obtener el resultado esperado para `gen`. El objetivo es practicar la construcción de programas imperativos mediante iteración.

6.2. Derivación del invariante desde de la postcondición (trabajo del alumno)

Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 3. Iteración** especialmente, el apartado *Bucles y concepto de invariante*

Partiremos de la postcondición de la función especificada en 3.5, y derivaremos el invariante del bucle iterativo (consúltense [Peña93, derivación del invariante a partir de la postcondición, pag(s) 117 y 118], [Peña97, derivación del invariante a partir de la postcondición, pag(s) 133 y 134], [Peña05, derivación del invariante a partir de la postcondición, pag(s) 146 y 147] y [Balc93, derivación directa de bucles, pag(s) 227 y ss.]).
Escríbase aquí el invariante del bucle, junto con las explicaciones necesarias para explicar su derivación formal, así como la protección de dicho bucle (su condición de terminación) y las explicaciones pertinentes sobre la misma.
(**Pista**: una opción que habitualmente es razonable probar consiste en debilitar la postcondición, puesta previamente en forma conjuntiva, para lograr un invariante y una condición de terminación).

Actividad 6.2. Invariante y protección del bucle para `genIt`

6.3. Inicialización del bucle (trabajo del alumno)

El invariante que acabamos de derivar (apartado 6.2), debe cumplirse al comienzo del bucle (así como, por supuesto, al final de cada iteración del mismo). Para garantizar esta propiedad, vamos a derivar una inicialización para el bucle. Normalmente, se trata de obtener un conjunto de instrucciones de asignación que proporcionen valores adecuados a las variables que intervienen en el cuerpo del bucle y sobre las que, por tanto, se define el invariante. Ha de utilizarse la regla de la asignación como herramienta para obtener esos valores.

Derívese, a continuación, la inicialización del bucle y explíquense los argumentos pertinentes para comprobar la invarianza al inicio del mismo.
En el espacio que se deja a continuación, debe calcularse la inicilaización del bucle:

Actividad 6.3. Inicialización del bucle para `genIt`

6.4. Instrucción avanzar del bucle (trabajo del alumno)

Una vez obtenido el invariante del bucle y la inicialización procederemos a derivar su cuerpo, comenzando por una instrucción *avanzar* que haga progresar el bucle hacia su terminación. Asímismo, se propondrá una función de cota para dicho bucle (consúltense [Peña93, derivación de bucles a partir de invariantes, pag(s) 116], [Peña97, derivación de bucles a partir de invariantes, pag(s) 132], [Peña05, derivación de bucles a partir de invariantes, pag(s) 145 y 146] y [Balc93, derivación directa de bucles, pag(s) 227 y ss.]).
En el espacio reservado a continuación, deberán escribirse la instrucción avanzar así como la cota del bucle junto con las explicaciones pertinentes para ambas.

Actividad 6.4. Instrucción avanzar y cota para el bucle de `genIt`

6.5. Restablecimiento del invariante (trabajo del alumno)

Esta instrucción de avanzar hará que el invariante que habíamos calculado pierda su propiedad de invarianza, por ello será necesario incluir otra instrucción (o secuencia de instrucciones) que nos permita restablecer dicha propiedad (consúltense [Peña93, derivación de bucles a partir de invariantes, pag(s) 116] o [Peña97, derivación de bucles a partir de invariantes, pag(s) 132], [Peña05, derivación de bucles a partir de invariantes, pag(s) 145 y 146] y [Balc93, derivación directa de bucles, pag(s) 227 y ss.]).

Derívese aquí la instrucción restablecer, indicando los argumentos necesarios por los que restablece la invarianza.

Actividad 6.5. Instrucción para restablecer la invarianza a cada vuelta del bucle en `genIt`

6.6. Código del algoritmo iterativo de `genIt` (trabajo del alumno)

A continuación se deberá escribir el código completo del algoritmo iterativo para `genIt` junto con su precondición y postcondición, así como su invariante y cota (véanse los anteriores apartados, (6.4, en la página 22) y (6.5, en la página 22) para completar esta tarea).
En el siguiente espacio disponible para ello (en la página siguiente), escríbase el código completo de `genIt` con los predicados que la documentan.

Actividad 6.6. Código de `genIt`

6.7. Estudio del coste (trabajo del alumno)

> Para realizar este apartado, el alumno ha debido estudiar ya el **Tema 3. Iteración** especialmente, el apartado *Cálculo de la eficiencia en programas iterativos, operaciones entre órdenes de complejidad*

En este apartado se deberá escribir (a partir del invariante) la función de cota del bucle y el coste del algoritmo iterativo obtenido en el apartado anterior (esto último, según aparece en [Peña93, Reglas prácticas para el cálculo de la eficiencia, pag(s) 11 y ss.], [Peña97, Reglas prácticas para el cálculo de la eficiencia, pag(s) 12 y ss.]) o [Peña05, reglas prácticas para el

cálculo de la eficiencia, pag(s) 13 y ss.]. Se deberá razonar y justificar dicho coste según las anteriores reglas.

Actividad 6.7. Cálculo del coste para `genIt`

6.8. Cuestiones sobre la función iterativa

1. ¿Qué sucedería si no se inicializasen las variables antes de comenzar el bucle?
2. ¿Qué relación tiene la función limitadora de un algoritmo iterativo con la estructura de preorden bien fundado necesaria para demostrar la finalización de uno recursivo?
3. Se recomienda al alumno que realice la conversión de la función recursiva final `iigen` (véase 5.4 en la página 20) a iterativo
4. Si el alumno ha realizado la conversión de recursivo final a iterativo como se proponía en las cuestión anterior, ahora podrá responder a las siguientes cuestiones:
   - ¿Cuál es la relación entre los distintos predicados que documentan las dos versiones (recursiva final e iterativa) de `gen`?. Realícese una tabla que explique las correspondencias.
   - Compárese la eficiencia (coste asintótico temporal en el caso peor) de `iigen` y `genIt`

Tarea 2: Implementación y Análisis Empírico

7. Implementación

7.1. Código y juegos de pruebas (trabajo del alumno)

A partir de este punto, debe listarse el código según las indicaciones que se dan en a continuación. No obstante, debe entregarse el listado de todos los ficheros fuente de la práctica así como los ejecutables.
Recordamos al alumno que es su responsabilidad conocer el lenguaje de progamación así como algún entorno de edición y compilación adecuado para el mismo sobre el sistema operativo y plataforma en la que trabaje. A este respecto, el Equipo Docente no se hará cargo sobre dudas de código o compilación en Modula-2.

7.2. Módulos de apoyo a la implementación

El Equipo Docente, a través del área de material de los cursos virtuales proporcionará módulos de apoyo a la implementación, de uso obligatorio, para facilitar la codificación y prueba de la práctica. Las instrucciones de uso y los propios módulos aparecerán a su debido tiempo, ya que, en primer lugar, debería realizarse el diseño y verificación de los algoritmos, al que dicho código **deberá responder** para considerarse correcto.

7.3. Módulos que implementará el Alumno

El alumno deberá implementar, probar y documentar los módulos que el equipo docente indique como trabajo del alumno, lo que se comunicará oportunamente a través de los foros del entorno virtual de la asignatura.

7.4. Formato de entrada y salida

El formato de entrada/salida se describirá en un documento que se publicará junto con los juegos de pruebas.

7.5. Juegos de pruebas

Un juego de pruebas consiste en un conjunto de datos de entrada junto con la salida esperada para esos datos. Ésta se contrasta con la salida que realmente produce el programa. Los juegos de pruebas deberían estar lo más cercanos posibles a la exhaustividad, probando cualesquiera condiciones potencialmente productoras de errores (como por ejemplo extremos de los intervalos de inmersión, intervalos vacíos, puntos de corte o condiciones no habituales).
Se publicará un documento anexo sobre el significado y diseño de los Juegos de Pruebas que estará disponible a través de los cursos virtuales. Además, en dicho entorno, estará disponible un Juego de Pruebas Público. Que el programa del alumno supere este Juego de Pruebas

es condición necesaria para obtener calificación en la Práctica (y, como consecuencia, en la Prueba Presencial de la Asignatura). Existirá también un Juego de Pruebas Privado, (que, por tanto, no le será proporcionado al alumno) contra el que también se ejecutarán los programas de los alumnos, que deberán, asimismo, superarlo, por lo que recomendamos que se prueben exhaustivamente dichos programas antes de su entrega.

7.6. Estudio empírico del coste

Con los datos sobre los tiempos invertidos en los cálculos, obtenidos a partir de las diferentes ejecuciones del programa (para diferentes ejemplares y tamaños del mismo), se pide que se construya una gráfica que estudie el coste *empírico* de los algoritmos programados. Como decimos, deben realizarse diversas ejecuciones con problemas de diferentes tamaños, ya que, lo que se pretende estudiar con la gráfica es el crecimiento del coste respecto al tamaño del problema.
Esta gráfica representará el tiempo empleado en la ejecución del algoritmo en función de la dimensión de la matriz de entrada. Al contar con varios algoritmos diseñados e implementados que participan en la resolución del problema general, se deberá realizar una gráfica por algoritmo, para así poder realizar un estudio comparativo del coste real de los algoritmos implementados.

7.7. Cuestiones sobre la implementación

1. ¿Cómo se podría mejorar (optimizar) el coste de `gen`?

Documentación

8. Documentación que hay que entregar

A modo de resumen, listamos a continuación la documentación que se debe entregar **al tutor, antes de la fecha que éste disponga**, para que la práctica se considere completa:

1. **Cuadernillo del alumno cumplimentado** con el diseño. Las cuestiones adicionales, para las que no hay espacio reservado en este cuadernillo, son voluntarias y pueden contestarse en tantas hojas añadidas como sea necesario
2. **Implementación:**
   - El programa se codificará en y constará de varios módulos, algunos de los cuales deberá implementarlos el alumno (como se explica en la sección 7.3). En todos los casos, el código ha de estar lo mejor comentado posible
   - El ejecutable funcionará bajo GNU/Linux o DOS/Windows
   - El soporte en que se entregue la implementación de la práctica debe estar libre de virus. La presencia de estos se considerará un error grave
3. **Juegos de pruebas:** según se especifica en la sección 7.5 (Consúltese la documentación adicional)
4. **Estudio empírico del coste:** según se especifica en la sección 7.6

Referencias

[CCM93] Jorge Castro Felipe Cucker Xavier Messeguer Albert Rubio Luis Solano Borja Valles. *Curso de Programación*. McGraw-Hill, 1993.

[Cerr93] José Antonio Cerrada Manuel Collado. *Programación I*. UNED, 1993.

[CCEG2K] José Antonio Cerrada Manuel Collado José Félix Estívariz Rubén Gómez. *Fundamentos de Programación con Modula 2*. CEURA, 2000.

[ColP2] José Ignacio Mayorga Toledano Miguel Rodríguez Artacho Fernando López Ostenero. *Colección de problemas de Programación II*. Edición interna del Equipo Docente, 2002.

[Balc93] José Luís Balcázar. *Programación Metódica*. McGraw-Hill, 1993.

[Peña93] Ricardo Peña. *Diseño de Programas: formalismo y abstracción*. Prentice Hall, 1993.

[Peña97] Ricardo Peña. *Diseño de Programas: formalismo y abstracción*, 2ª Edición. Prentice Hall, 1997.

[Peña05] Ricardo Peña. *Diseño de Programas: formalismo y abstracción*, 3ª Edición. Prentice Hall/Pearson, 2005.