

Posible solución a la Práctica 9 de C: 24-03-2009

Objetivos:

- Revisar todos los conceptos de análisis y diseño de soluciones algorítmicas vistas hasta el momento.
- Ver en un ejemplo cómo afecta la decisión “Programas = Algoritmos + EE.DD.”
- Asentar los conocimientos sobre definición, manejo y paso de tipos estructurados a funciones en C.

Enunciado:

Queremos disponer de un programa que maneje polinomios de una variable, por ejemplo:

$$P(x) = 3 \cdot x^5 - 2 \cdot x^2 + 6 \cdot x - 2$$

Sobre los polinomios queremos hacer tres operaciones:

Algoritmo lee (p) es

P: polinomio; {parámetro resultado}

{Debe pedir al usuario los datos (coeficientes y grados) del polinomio, que deben quedar almacenados en una única variable, p, para su posterior uso}

Algoritmo muestra (p) es

P: polinomio; {parámetro dato}

{Debe mostrar por pantalla el polinomio, de forma similar al ejemplo que hemos puesto arriba}

Algoritmo evalúa (p, x0, res) es

P: polinomio; {parámetro dato}

X0: numérico; {real; parámetro dato}

Res: numérico; {real; parámetro resultado}

{Debe proporcionar en res el resultado de evaluar P(x) en el punto x0}

1. Analiza qué es lo que hay que hacer en cada uno de los algoritmos, en términos de repeticiones y las estructuras de datos necesarias. Propón dos formas distintas de definir el tipo *polinomio* y analiza qué ventajas e inconvenientes tiene una frente a la otra.
2. Para cada uno de los dos tipos de datos realiza el análisis de los algoritmos anteriores (análisis de la secuencia, tipo de esquema repetitivo).

3. Elije uno de los dos tipos e implementa en C un programa que realice los tres algoritmos (leer, mostrar y evaluar) hasta que el usuario decida terminar.
- Debes entregar un único fichero de texto (WORD, ODT, TXT, o similar) con la solución a todos los apartados, en el orden en el que se piden.
 - El fichero debe llevar la identificación del número de práctica y vuestro nombre y primer apellido.
 - El código C que generes debe ir al final de la memoria. Opcionalmente, puedes entregarlo en un fichero .C junto con la memoria. En ese caso debe llamarse TU_NOMBRE_practica_9.c
 - Aquellas soluciones que no respeten todas estas indicaciones no serán revisadas.

Fecha límite de entrega: 20 de abril de 2009 (ampliado a 27 de abril de 2009)

1. Analiza qué es lo que hay que hacer en cada uno de los algoritmos, en términos de repeticiones y las estructuras de datos necesarias. Propón dos formas distintas de definir el tipo *polinomio* y analiza qué ventajas e inconvenientes tiene una frente a la otra.

Algoritmo lee (p) es

P: polinomio; {parámetro resultado}

{Debe pedir al usuario los datos (coeficientes y grados) del polinomio, que deben quedar almacenados en una única variable, p, para su posterior uso}

Debemos definir un tipo Polinomio que almacene los coeficientes y los grados del polinomio, indicando cuál es el grado máximo del polinomio.

Supondremos que el polinomio no está vacío (al menos tendrá un término), para facilitar la lectura.

Debemos recorrer todos los posibles términos del polinomio y guardar los coeficientes de aquellos que no sean nulos. Por ejemplo en el caso $P(x) = 3x^5 - 2x^2 + 6x - 2$, deberíamos guardar (3,5), (-2,2), (6,1) y (-2,0) siendo los pares (coeficiente, grado).

Si se supone conocido el grado máximo, deben recorrerse los términos de 1 al grado máximo (o preguntarle directamente al usuario) buscando los coeficientes no nulos (ESQUEMA 2, TRATAMIENTO EN CURSO = TRATAMIENTO FINAL = LEER TERMINO; Número de iteraciones conocidas → esquema repetitivo para).

Si sólo limitamos el número de pares (coeficiente, grado) del polinomio se deben solicitar al usuario hasta que no quiera introducir más o nos quedemos sin memoria.

En ambos casos será necesario un esquema repetitivo sobre la secuencia (i, (coeficiente, grado_i)) y el tratamiento será el mismo (LEER TÉRMINO → ESQUEMA 2; número de iteraciones desconocido (pero al menos hay una) → esquema repetir).

LEER TERMINO no será una operación elemental, ya que debemos garantizar que los términos del polinomio están ordenados. Debemos leer los datos (coeficiente, termino) del término i-ésimo y después comprobar que el grado es mayor que el término i-1-ésimo. Esta operación se puede hacer para los términos entre 2 y grado_máximo.

Algoritmo muestra (p) es

P: polinomio; {parámetro dato}

{Debe mostrar por pantalla el polinomio, de forma similar al ejemplo que hemos puesto arriba}

Debemos recorrer todos los pares no nulos del polinomio ((coeficiente, grado)) y mostrarlos por pantalla. Se puede asumir que la variable es X y que se muestre en una única línea de texto.

Será necesario utilizar la misma secuencia del programa anterior: (i, (coeficiente_i, grado_i))

Debe recorrerse toda la secuencia y para todos los elementos se hará el mismo tratamiento (esquema 2).

Algoritmo evalúa (p, x0, res) es

P: polinomio; {parámetro dato}

X0: numérico; {real; parámetro dato}

Res: numérico; {real; parámetro resultado}

{Debe proporcionar en res el resultado de evaluar P(x) en el punto x0}

Partiremos de un polinomio almacenado. Debemos recorrer nuevamente la secuencia de los términos del polinomio, pero ahora añadiendo el valor de la potencia para el grado i: (i, (coeficiente_i, grado_i), potencia_i). Los valores coeficiente_i * potencia_i deben acumularse en res.

Se hará el mismo tratamiento para todos los elementos de la secuencia, que serán conocidos (esquema 2 con para).

Existen dos formas sencillas de almacenar un polinomio, siempre partiendo del hecho de que los términos del polinomio se almacenan ordenados por su grado:

{Definición 1} Se permite almacenar un número máximo de coeficientes, estando los grados limitados entre 1 y TAM. Si un coeficiente no se utiliza, se pondrá a cero.

Polinomio = registro de

Grado_max: numérico; {máximo grado válido del polinomio}

Coeficiente: vector[1..TAM] de numérico; {reales}

Fin registro

{Definición 2} Se permite almacenar un número máximo de TAM coeficientes no nulos. Cada elemento del polinomio estará formado por (coeficiente, grado). Grado_max nos indicará el grado del último elemento del vector. También podríamos añadir un contador del número de elementos del vector realmente ocupados.

Polinomio = registro de

Grado_max: numérico; {máximo grado válido del polinomio}

Ocupados: numérico; {entero}

Coeficiente, grado: vector[1..TAM] de numérico; {reales y entero, respectivamente}

Fin registro

Este último tipo también permitiría definir un tipo básico TERMINO y declarar el polinomio como un vector de términos. Contendría la misma información y las soluciones serían equivalentes:

Termino = registro de

Coeficiente, grado: numérico; {reales y entero, respectivamente}

Fin registro

Polinomio = registro de

Grado_max: numérico; {máximo grado válido del polinomio}

Ocupados: numérico; {entero}

terminos: vector[1..TAM] de Termino;

fin registro

2. Para cada uno de los dos tipos de datos realiza el análisis de los algoritmos anteriores (análisis de la secuencia, tipo de esquema repetitivo).

Los algoritmos que utilicen la definición 1 sólo tienen que pedirle al usuario los grados con coeficientes no nulos. Entonces la relación (grado-1) y posición del vector es bidireccional (en su implementación en C la relación bidireccional es entre el grado y la posición).

Esto supone lectura del vector rápida, pero debe recorrerse todo el vector (hasta grado_max) para mostrarlo o evaluarlo. Existe una limitación del grado máximo del polinomio (TAM).

Los algoritmos que utilicen la definición 2 no tienen la limitación del grado máximo, si no del número de términos válidos (TAM). Los algoritmos de lectura y escritura no tienen que recorrer TAM elementos, sino sólo de 1 a ocupados.

*A la hora de evaluar el polinomio con la definición 1 es bastante sencillo generar las potencias asociadas a cada posición (por cada posición que visitemos, potencia_actual := potencia_actual * x0).*

Para evaluar el polinomio con la definición 2 tenemos que ir recorriendo las potencias de 1 a grado_max y detenernos cada vez que coincida con un término no nulo del vector.

Debemos elegir la definición 1 si nuestros polinomios tienen grados pequeños (inferior a TAM). Debemos elegir la definición 2 si nuestros polinomios tienen términos (máximo TAM) y grados arbitrariamente grandes.

3. Elige uno de los dos tipos e implementa en C un programa que realice los tres algoritmos (leer, mostrar y evaluar) hasta que el usuario decida terminar.

La solución 2 es más flexible, pero también más complicada de implementar. Escojo esta para mostrar estos conceptos en código C, pero también sería válido escoger la solución más sencilla (pero con menor capacidad de almacenamiento).

```

Termino = registro de
    Coeficiente, grado: numérico; {reales y entero, respectivamente}
Fin registro

Polinomio = registro de
    Grado_max: numérico; {máximo grado válido del polinomio}
    Ocupados: numérico; {entero}
    terminos: vector[1..TAM] de Termino;
fin registro

Algoritmo lee (p) es
    P: polinomio; {parámetro resultado}
    {Debe pedir al usuario los datos (coeficientes y grados) del polinomio, que deben quedar
    almacenados en una única variable, p, para su posterior uso}
    {Necesitamos además leer datos, calculando cuántos hay y cuál es el grado máximo}
    {Debemos leerlos comprobando que el usuario los introduce ordenadamente}
    Coeficiente, grado: numérico; {datos proporcionados por el usuario}
    Continuar: carácter; {indica si el usuario quiere continuar o no; puede querer crear un polinomio
    nulo}
    Inicio
        Continuar:= 's';
        p.ocupados := 0; {datos leídos}
        escribir "Introduce los datos del polinomio en orden. No podrás dar marcha atrás";
        repetir
            {leer termino válido}
            escribir "Dame el coeficiente y el grado, deben estar ordenados: ";
            leer coeficiente, grado;
            {comprobar si está ordenado}
            Si p.ocupados >= 1 entonces
                Mientras grado <= p.terminos(p.ocupados-1).grado hacer
                    escribir "Dame el coeficiente y el grado, deben estar ordenados: ";
                    leer coeficiente, grado;
                Fin mientras;
            {grado > p.terminos(p.ocupados).grado}
            Fin si;
            {guardar término}
            p.ocupados := p.ocupados+1;
            p.terminos(p.ocupados).coeficiente := coeficiente;
            p.terminos(p.ocupados).grado := grado;
            escribir "¿Quieres continuar introduciendo datos?";

```

leer continuar;
hasta que continuar <> 's' or p.ocupados = TAM;
p.grado_max := p.terminos(p.ocupados).grado;

Fin

Algoritmo muestra (p) es

P: polinomio; {parámetro dato}

{Debe mostrar por pantalla el polinomio, de forma similar al ejemplo que hemos puesto arriba}

I: numérico; {lo utilizamos para recorrer los términos no nulos. Estarán entre la posición 1 y p.ocupados → esquema 2}

Inicio

Escribir "El polinomio tiene un grado máximo ", p.grado_max, " y ", p.cuantos, " términos:";

Para i desde 1 hasta p.ocupados hacer

*Escribir p.terminos(i).coeficiente, " x**", p.terminos(i).grado;*

Fin para

Fin

Algoritmo evalúa (p, x0, res) es

P: polinomio; {parámetro dato}

X0: numérico; {real; parámetro dato}

Res: numérico; {real; parámetro resultado}

{Debe proporcionar en res el resultado de evaluar P(x) en el punto x0}

Potencia: numérico; {acumulará x0 elevado al grado actual}

I, J: numérico; {término del polinomio y posible grado del término, respectivamente}

*{En lugar de calcular x0**grado en cada caso, lo hacemos de forma incremental, avanzando hasta el siguiente grado; vamos contando en j el grado posible x0**j, y hacemos que j vaya avanzando}*

{Debe repetirse para cada término de los p.ocupados → esquema 2 con iteraciones conocidas → para}

Inicio

Res := 0;

Potencia := 1; {corresponde a j = 0}

J = 0;

Para i desde 1 hasta p.ocupados hacer

Mientras j < p.terminos(i).grado hacer

J := j + 1;

*Potencia := potencia * x0;*

*Fin mientras; {al terminar, j = p.terminos(i).grado y potencia es x0**j}*

*Res := res + p.terminos(i).coeficiente * potencia;*

Fin para

fin