INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESTRUCTURAS DE DATOS

PRÁCTICA 4

LISTAS SIMPLEMENTE ENLAZADAS

HERNÁNDEZ CASTELLANOS CÉSAR URIEL
MARTINEZ ISLAS MAURICIO JOEL

1CV7

30/03/2017

LUNA BENOSO BENJAMIN

Indice

Resumen:

En el presente reporte se muestra la documentación de la práctica, cuya función principal es la siguiente:

La obtención de distintos resultados matemáticos, los cuales son:

- Obtener la derivada de un polinomio.
- Obtener la integral de un polinomio.
- Suma de dos polinomios.
- Producto de dos polinomios.

Introducción:

Las listas enlazadas fueron desarrolladas en 1956 por Cliff Shaw y Herbert Simon en RAND Corporation como una estructura de datos de su lenguaje de procesamiento de la información (IPL), las listas fueron usados por los autores para el desarrollo de distintos programas de inteligencia artificial

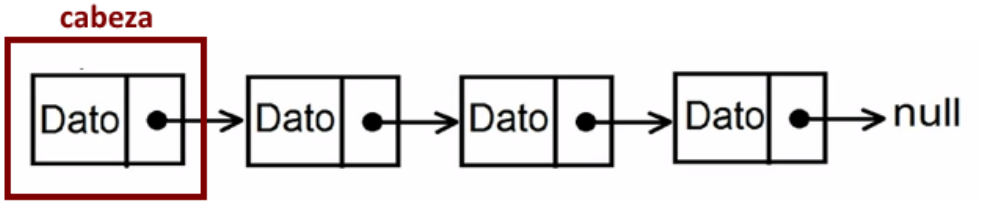
Las listas enlazadas se tratan de estructuras de datos similares a los array, sin embargo el acceso en las listas enlazadas se realiza con un puntero y no con un índice, la asignación de la memoria se realiza durante la ejecución del programa.

En cambio, mientras que en un array los elementos están de manera secuencial en la memoria en una lista se encuentran dispersos, el enlace entre los nodos se realiza por un puntero.

El puntero siguiente del último elemento apunte hacia NULL, lo que indica el final de la lista.

Tipos de listas enlazadas:

- Listas simplemente enlazadas.
- Listas doblemente enlazadas.
- Listas enlazadas circulares.
- Listas enlazadas simples circulares.
- Listas enlazadas doblemente circulares.



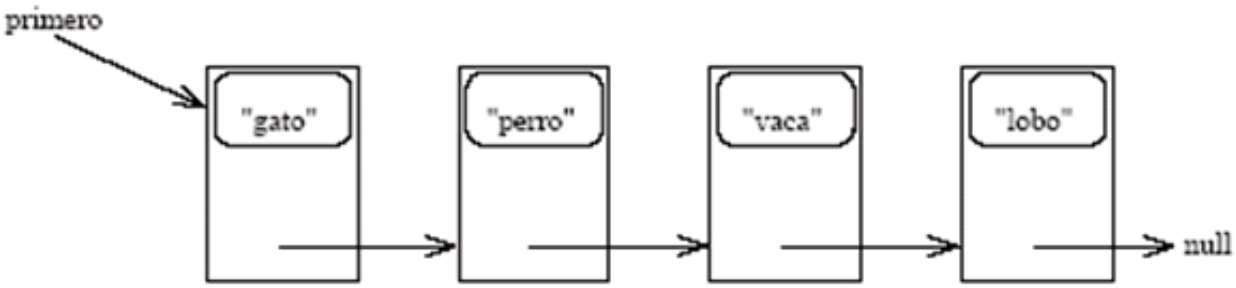
Para poder acceder a un elemento especifico de la lista, la lista es recorrrida comenzando por el inicio, el puntero siguiente permite el cambio hacia el próximo elemento.

El desplazamiento únicamente se realiza en una dirección del primero al último elemento, esto para las LSE, en cambio si queremos recorrer la lista hacia delante y hacia atrás tendremos que recurrir a las listas doblemente enlazadas.

Aplicaciones de las listas enlazadas.

Las listas enlazadas se usan como módulos para otras estructuras de datos, como lo son las pilas, colas y sus variaciones.

En algunas ocasiones, las listas enlazadas se usan para implementarse con árboles binarios de búsqueda equilibrados, Sin embargo, algunas veces ka kista enlazada es dinámicamente creada fuera de un subconjunto propio de nodos semejante a un árbol, y son más eficientes para recorrer diversos datos.



Ventajas de las listas enlazadas respecto a los arrays.

1- Tamaño dinámico: Lo que implica optimización de la memoria.

Desventajas de las listas enlazadas respecto a los arrays.

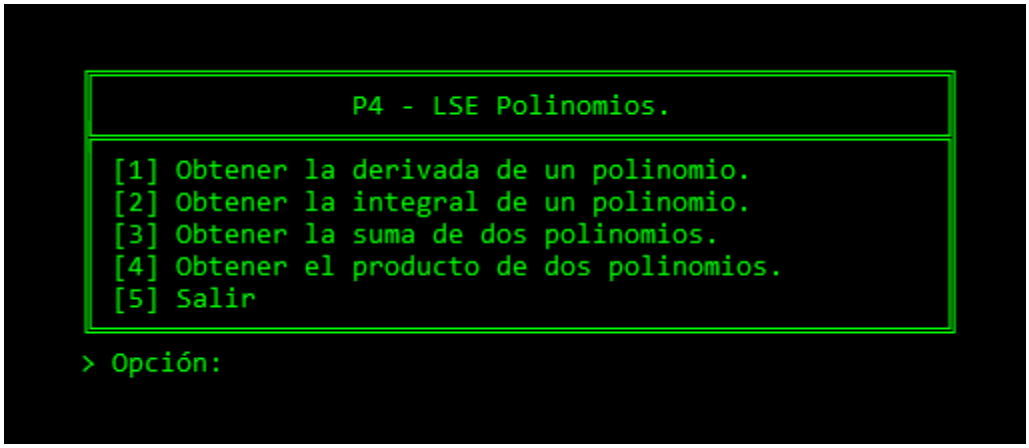
1- El acceso es secuencial (para llegar a una posición deberemos pasar por todas las anteriores). Esto significa lentitud. Imaginad por un momento el tinglado que tendríamos que montar para ordenar una lista enlazada. Si buscamos un elemento de una lista ordenada también tardaremos más, no vale la búsqueda dicotómica que vimos en la Ampliación 1 de C (métodos de clasificación en memoria dinámica).
2- El código se complica.

Experimentación y resultados.

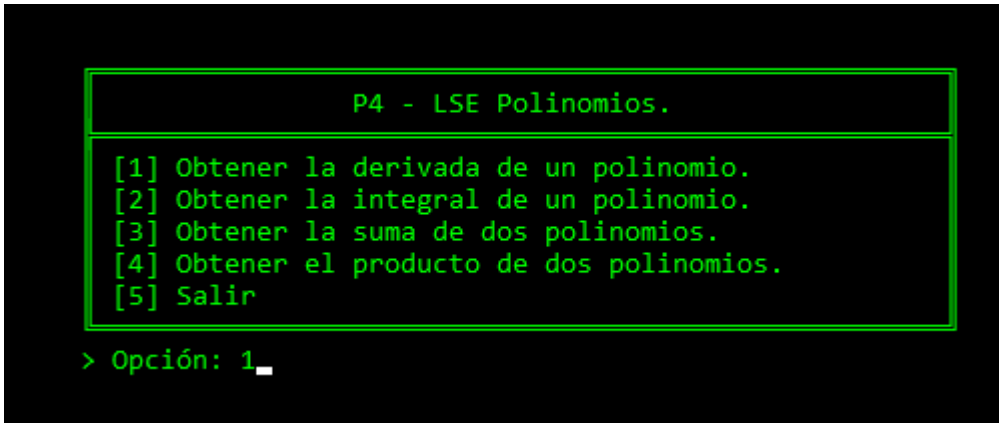
La práctica consiste en llevar a cabo una aplicación en el que se haga uso de listas simplemente enlazadas, la práctica consiste en solicitar al usuario un cierto número de polinomios para realizar las siguientes operaciones:

- Obtener la derivada de un polinomio.
- Obtener la integral de un polinomio.
- Suma de dos polinomios.
- Producto de dos polinomios.

Derivada de un polinomio.



Solicitud de opción en el menú principal.

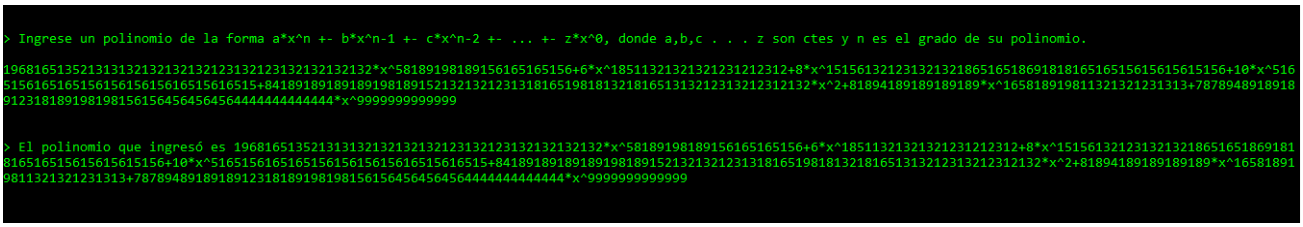


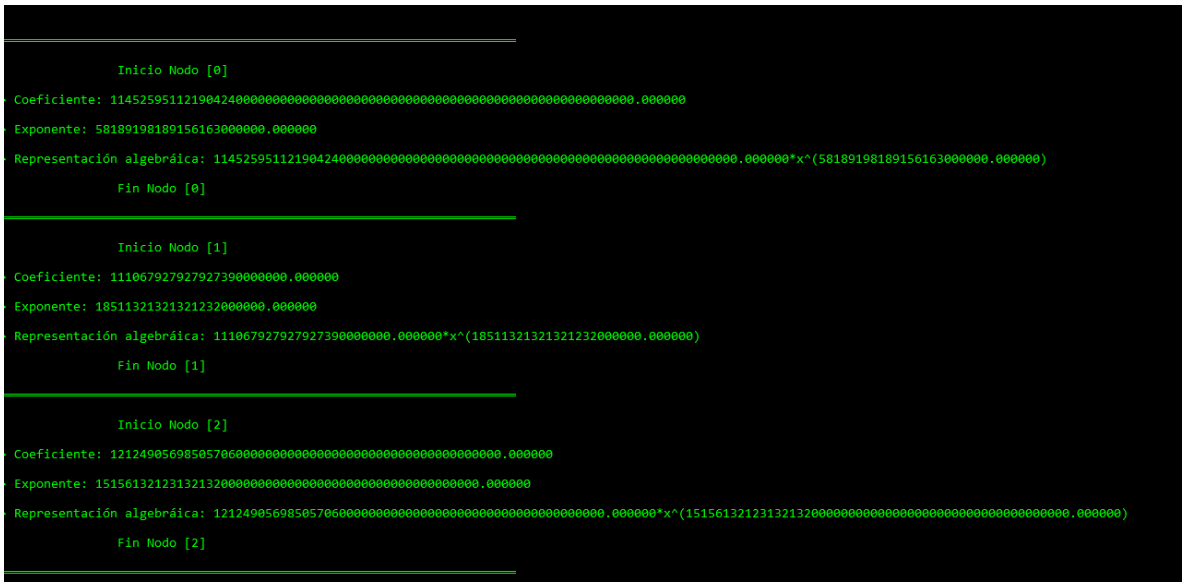
Ingreso a la opción uno.

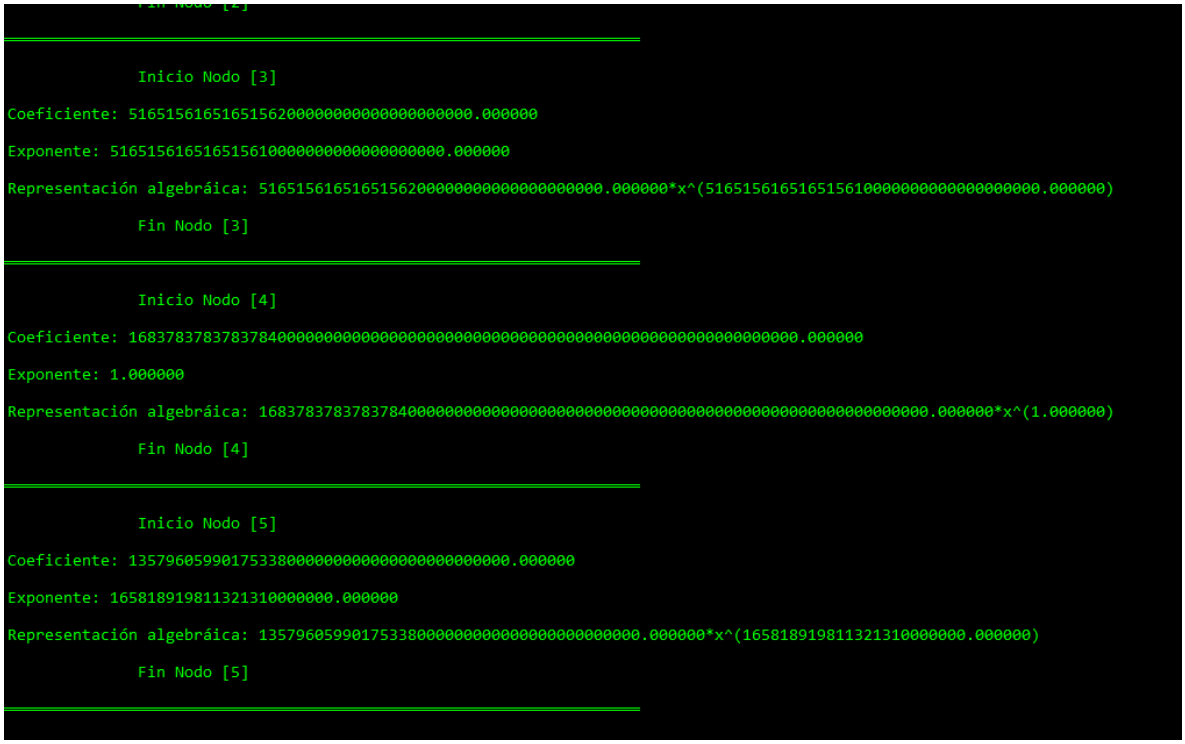
Solicitud e ingreso del polinomio:

1968165135213131321321321321232132132132132132*x^581891981891561
65165156+6*x^185113213213212312312+8*x^151561321231321865165186918181651651561561515615+10*x^516
5156165165156156156516515616515+84189189189189189189152132132123131816519818132181651313212313212312132*x^2+81894189189189189*x^16581891981132132123131+7878948918918
912318189198198156156456456456444444444444444*x^9999999999999

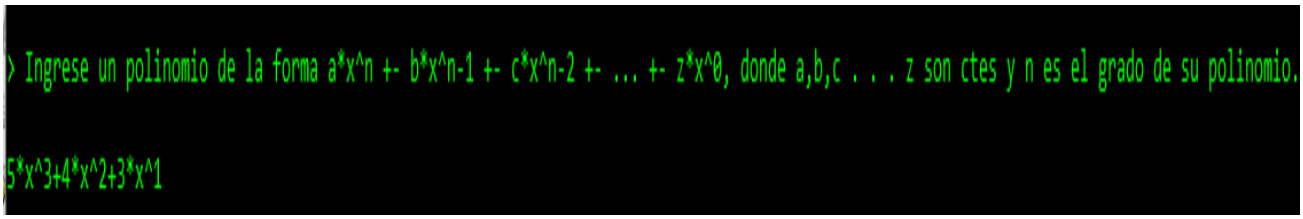
890189123189189198198156156456456456444444444444444*x^9999999999999



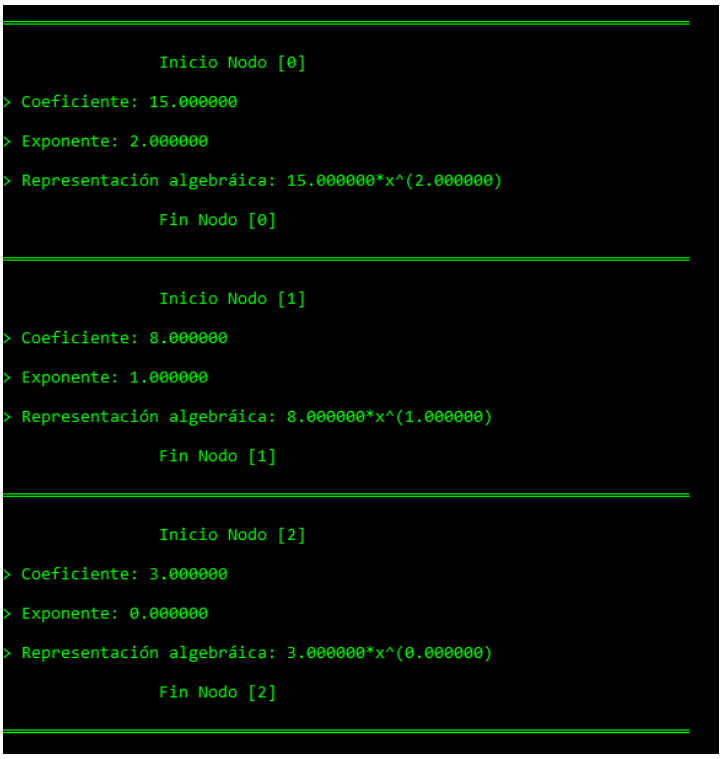




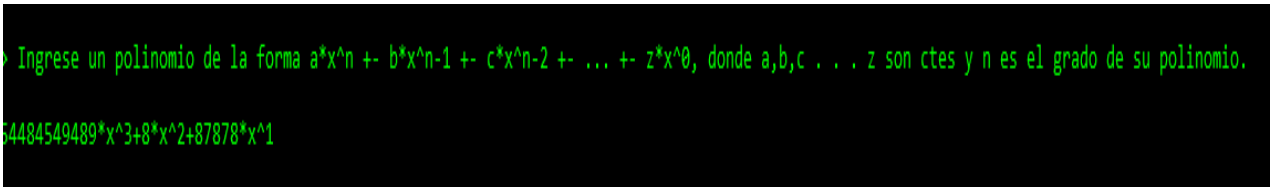
Mostrar la lista enlazada con el polinomio ya derivado.



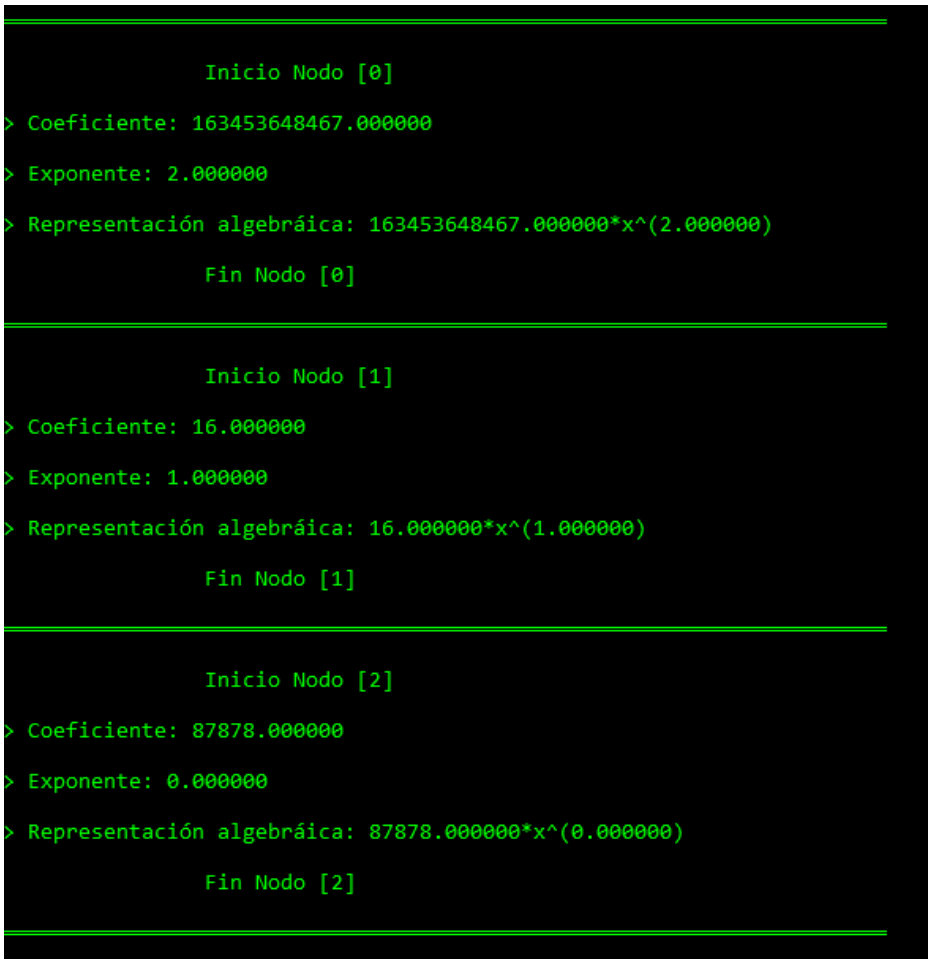
Solicitud del polinomio e ingreso del polinomio (5*x^3+4*x^2+3*x^1).



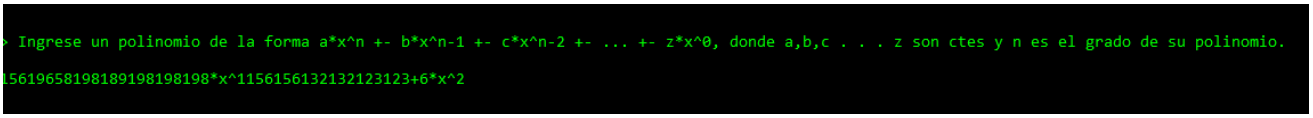
Mostrar la lista enlazada con el polinomio ya derivado.



Ingreso del polinomio 54484549489*x^3+8*x^2+87878*x^1



Mostrar la lista enlazada con el polinomio ya derivado.



Ingreso del polinomio

1561965819819198189191981 98*x^11561561321321312 3+6*x^2

```
Inicio Nodo [0]
> Coeficiente: 18058763607644230000000000000000000000000000.000000
> Exponente: 1156156132132123100.000000
> Representación algebráica: 18058763607644230000000000000000000000000000.000000*x^(1156156132132123100.000000)
Fin Nodo [0]

Inicio Nodo [1]
> Coeficiente: 12.000000
> Exponente: 1.000000
> Representación algebráica: 12.000000*x^(1.000000)
Fin Nodo [1]
```

Mostrar la lista enlazada con el polinomio ya derivado.

```
> Ingrese un polinomio de la forma a*x^n +- b*x^n-1 +- c*x^n-2 +- ... +- z*x^0, donde a,b,c . . . z son ctes y n es el grado de su polinomio.
894981891981563132168519819815165165156165156561651656165*x^1851891121+6*x^3+58*x^2+158181814818*x^1

> El polinomio que ingresó es 894981891981563132168519819815165165156165156561651656165*x^1851891121+6*x^3+58*x^2+158181814818*x^1
```

Ingreso del polinomio

894981891981563132116851198198115165165156165156561651656165*x^1861891121+6*x^3+58*x^2+158181814818*x^1

```
Inicio Nodo [0]
> Coeficiente: 165740901921643790000000000000000000000000000000000000000000000000.000000
> Exponente: 1851891120.000000
> Representación algebráica: 165740901921643790000000000000000000000000000000000000000000000000.000000*x^(1851891120.000000)
Fin Nodo [0]

Inicio Nodo [1]
> Coeficiente: 18.000000
> Exponente: 2.000000
> Representación algebráica: 18.000000*x^(2.000000)
Fin Nodo [1]

Inicio Nodo [2]
> Coeficiente: 116.000000
> Exponente: 1.000000
> Representación algebráica: 116.000000*x^(1.000000)
Fin Nodo [2]
```

```
                Inicio Nodo [3]
> Coeficiente: 158181814818.000000
> Exponente: 0.000000
> Representación algebráica: 158181814818.000000*x^(0.000000)
                Fin Nodo [3]
================================================================

--------------------------------
```

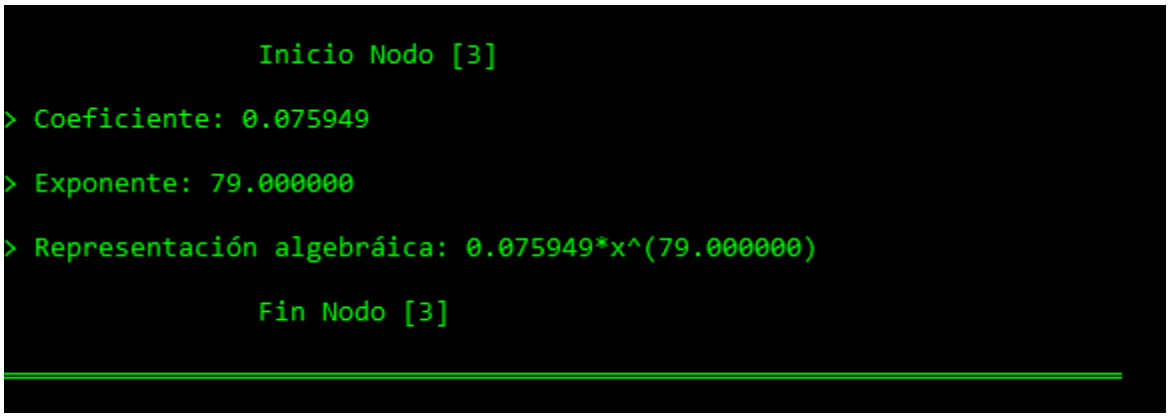
Integral de un polinomio.

Solicitud e ingreso del polinomio

9*x^9+6*x^4+5*x^2+6*x^78

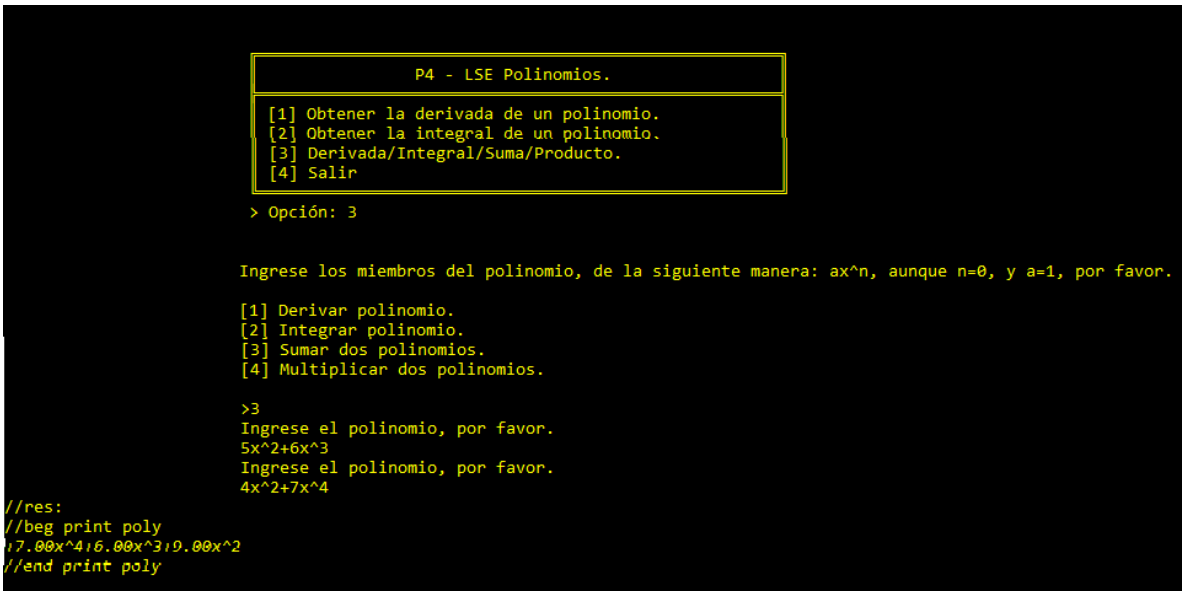
```
> Ingrese un polinomio de la forma a*x^n +- b*x^n-1 +- c*x^n-2 +- ... +- z*x^0, donde a,b,c . . . z son ctes y n es el grado de su polinomio.
9*x^9+6*x^4+5*x^2+6*x^78

> El polinomio que ingresó es 9*x^9+6*x^4+5*x^2+6*x^78
```

```
================================================================

                Inicio Nodo [0]
> Coeficiente: 0.900000
> Exponente: 10.000000
> Representación algebráica: 0.900000*x^(10.000000)
                Fin Nodo [0]
================================================================

                Inicio Nodo [1]
> Coeficiente: 1.200000
> Exponente: 5.000000
> Representación algebráica: 1.200000*x^(5.000000)
                Fin Nodo [1]
================================================================

                Inicio Nodo [2]
> Coeficiente: 1.666667
> Exponente: 3.000000
> Representación algebráica: 1.666667*x^(3.000000)
                Fin Nodo [2]
================================================================
```



Mostrar la lista enlazada con el polinomio ya integrado.

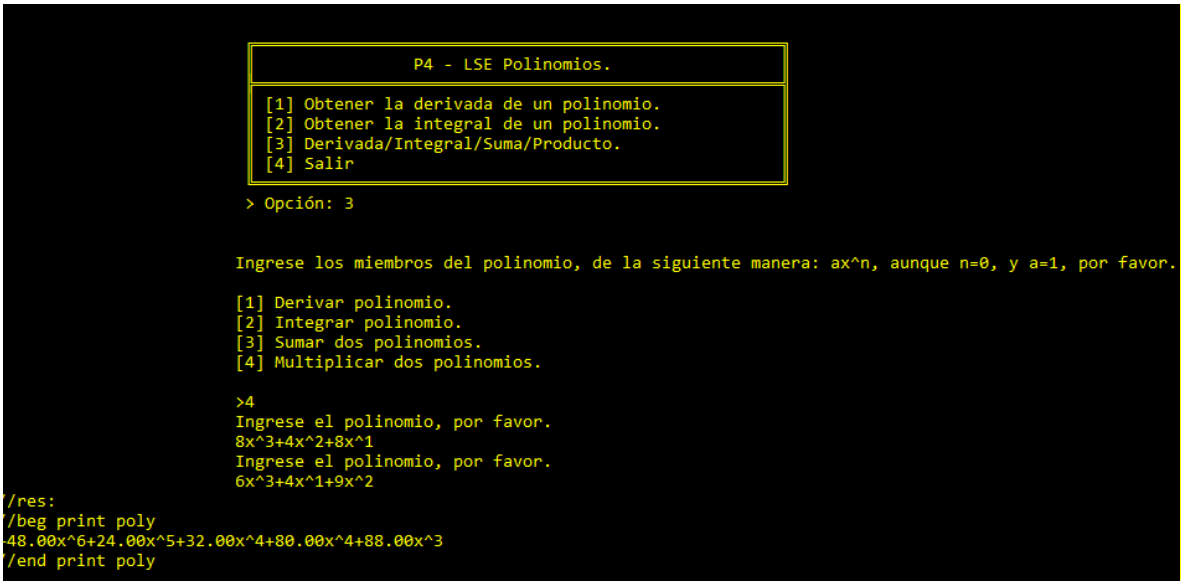
Suma de dos polinomios.



Ingreso de los polinomios e impresión del resultado:

5x^2+6x^3+ 4x^2+7x^4 = 9x^2+6x^3+7x^4

Producto de dos polinomios.



Ingreso de los polinomios e impresión del resultado:

8x^3+4x^2+8x^1 * 6x^3+4x^1+9x^2 =

48.00x^6+24.00x^5+32.00x^4+80.00x^4+88.00x^3

Pseudocódigo

// Se presentan en pseudocódigo de las funciones principales a continuación:

Funcion NODO* newcab <- multiPoly (NODO* cab1, NODO* cab2)

NODO* aux1, aux2

item auxItem, memItem, memItem2

Para aux1<-cab1 Hasta aux1=NULL Con Paso aux1=aux1->siguiente Hacer

Para aux2<-cab2 Hasta aux2=NULL Con Paso aux2=aux2->siguiente Hacer

memItem1 = aux1->dato

```
                memItem2 = aux2->dato
                reservar memoria de auxItem
                auxItem->coef = (memItem1->coef)*(memItem2->coef)
                auxItem->exp = (memItem1->exp)+(memItem2->exp)
                insertar auxItem en newcab
            Fin Para
        Fin Para
Fin Funcion

Funcion NODO* sumPoly <- addPoly ( NODO* cab1, NODO* cab2 )
        NODO* cCab1 = copia de cab1
        NODO* cCab2 = copia de cab2
        NODO* aux1, aux2
        item auxItem, memItem, auxItem2
        Para aux1<-cab1 Hasta aux1=NULL Con Paso aux1=aux1->siguiente Hacer
            reservar memoria de auxItem
            memItem = aux->dato
            auxItem->exp = memItem->exp
            auxItem->coef = memItem->coef
            Para aux2<-cab2 Hasta aux2=NULL Con Paso aux2=aux2->siguiente
Hacer
                auxItem = aux2->dato
                si auxItem2->exp == auxItem->exp entonces
                    (auxItem->coef)+=(auxItem2->coef)
```

```
                    eliminar a auxItem2 de cCab2
                FinSi
                insertar auxItem en sumPoly
            Fin Para
        Fin Para
        si cCab2 no está vacía entonces
            Para aux2<-cab2 Hasta aux2=NULL Con Paso aux2=aux2->siguiente
Hacer
                auxItem2 = aux2->dato
                insertar auxItem2 en sumPoly
            FinPara
        FinSi
Fin Funcion

Funcion NODO* integ <- polyInt ( NODO* cab )
        NODO* aux=cab
        NODO* newcab=NULL
        item auxItem, memItem
        Para aux<-cab Hasta aux=NULL Con Paso aux=aux->siguiente Hacer
            memItem = aux->dato
            reservar memoria para auxItem
            auxItem->exp = memItem->exp
            auxItem->coef = memItem->coef
            auxItem->exp++
```

```
            auxItem->coef/=auxItem->exp
            insertar auxItem en newcab
        Fin Para
Fin Funcion

Funcion NODO* deriv <- polyDeriv ( NODO* cab )
        NODO* aux=cab
        NODO* newcab=NULL
        item auxItem, memItem
        Para aux<-cab Hasta aux=NULL Con Paso aux=aux->siguiente Hacer
            memItem = aux->dato
            si auxItem->exp>0 entonces
                reservar memoria para auxItem
                auxItem->exp = memItem->exp
                auxItem->coef = memItem->coef
                auxItem->coef *= auxItem->exp
                auxItem->exp--
                insertar auxItem en newcab
            FinSi
        Fin Para
Fin Funcion
Funcion doublé*retornarCoeficientes(char *polinomio)
Entero i->0, c->0, d->0, r->0, h->0
```

Carácter *temporal, *polinomio1 -> polinomio, *mas;

Mas -> calloc.

Mas[0] = ‘+’

Si (polinomio[0] es diferente de ‘+’) entonces

    Si(polinomio[0] es diferente de ‘-’) entonces

        Concatenar mas con polinomio1

Fin Si

Fin Si

Temporal -> calloc

double *coeficientes, coeficientes2;

coeficientes -> calloc

double val -> 0

Para(i->0 i<longitud de polinomio i->i+1)

    Si polinomio[i] -> ‘+’ o polinomio[i] ->’-’ entonces

        c->i;

        Si(polinomio[i]->’-’) entonces

            r-> r+;

            temporal[r-1] = polinomio[i]

        Fin Si

        Mientras(polinomio[c] sea diferente de ‘*’) entonces

        c->c+1

    Si(polinomio[c] es diferente de ‘*’ y polinomio[c] es diferente de ‘-’ y polinomio[c] es diferente de ‘+’)

        r->r+1

```
                temporal[r-1] -> polinomio[c]
                Fin Si
                Fin Mientras
                Val -> atof(temporal)
                Coeficientes[h] -> val;
                h->h+1
                temporal-> calloc
                fin si
                fin para
                regresar coeficientes
                fin función
Funcion doublé*retornarExponentes (char *polinomio)
Entero i->0, c->0, d->0, r->0, h->0
Carácter *temporal;
Temporal->calloc
Double *coeficientes;
Coeficientes -> calloc
Double val -> 0
Para(i->0 i< longitud de polinomio i->i+1)
        Si(polinomio[i] es igual a '^') entonces
                c->i
                Si(polinomio[i+1]  es igual a '-') entonces
                        r->r+1
```

temporal[r-1] -> polinomio[i+1]

mientras(polinomio[c] es diferente de ‘+’ y c es diferente de la longitud de la cadena disminuido en uno) entonces

c->c+1

Si(polinomio[c] es diferente de ‘*’ y polinomio[c] es diferente de ‘-’ y polinomio[c] es diferente de ‘+’) entonces

r->r+1

temporal[r-1] -> polinomio[c]

Fin Si

Si(polinomio[c+1] es igual a ‘-’) entonces

Rompe el ciclo

Fin si

Fin mientras

Val->atof(temporal)

Coeficientes[h]->val

h->h+1

r->0

temporal->calloc

Fin Si

Fin para

Regresar coeficientes

Fin función

Conclusiones.

Hernández Castellanos César Uriel.

El tener pleno conocimiento de la implementación de las listas nos favorece la representación eficiente de los datos alojados en la memoria de la computadora, cuando la cantidad de elementos que se vayan a ingresar sea desconocida, por lo que el uso de variables de tipo puntero nos permite crear y destruir de manera dinámica.

Con respecto a la práctica, considero que se tuvo un buen desarrollo, aunque se tienen ciertos detalles, como el ingreso estricto de un cierto formato y la incomodidad que le puede generar al usuario ingresar una constante de la siguiente manera:

5*x^0

De ahí en fuera considero que fue una buena práctica.

Martínez Islas Mauricio Joel.

La manera en la que se resolvió no estuvo tan mal, ahora sí se pudo concentrarse en buenas prácticas y ser constantes con el diseño inicial de lo que se propuso. La verdad es que la función de simplificar polinomio debió de haber quedado mejor (en el sentido de que se derivaba en otra función, que era la de buscar un término). Algo que si se debe de notar es que no se le puso mucha atención al input del programa (no es flexible, pues) y tiene que aceptar formas que se apeguen estrictamente a ax^n, también otro detalle que se tiene que notar es el hecho de que el ”menú” no es precisamente un menú, y que estrictamente hablando, quedó mal esa función porque se debió de haber separado en la capa de vista para el usuario y otra lógica, haciendo que la de vista simplemente llame funciones y que no empiece a operar dentro de ella, como se hizo. De ahí en fuera, el código sigue teniendo detalles, pero no son tan graves como los mencionados.

Referencias

[1]Y. Langsam, M. Augenstein and A. Tenenbaum, *Estructuras de datos con C y C++*, 1st ed. Pearson Prentice Hall, 2000.

[2]S. Villalobos, *Introducción a las Estructuras de Datos*, 1st ed. Pearson Educación de México, S.A. de C.V., 2011.

[3]"Data Structures - GeeksforGeeks", *GeeksforGeeks*, 2017. [Online]. Available: http://www.geeksforgeeks.org/data-structures/. [Accessed: 05- Apr- 2017].

[4]"Data Structures - University of California, San Diego, Higher School of Economics | Coursera", *Coursera*, 2017. [Online]. Available: https://www.coursera.org/learn/data-structures. [Accessed: 05- Apr- 2017].

[5]"Estructuras de datos: listas enlazadas, pilas y colas.", *Calcifer.org*, 2017. [Online]. Available: http://www.calcifer.org/documentos/librognome/glib-lists-queues.html. [Accessed: 05- Apr- 2017].