PRÁCTICA Nº 3: 2 sesión

(del 10 al 19 de Abril de 2002)

Utilización del tipo abstracto de dato Pila. Evaluación de una expresión algebraica en notación infija.

0. OBJETIVO

El objetivo de esta práctica es la implementación del tipo abstracto de datos pila y su utilización en la evaluación de una expresión algebraica escrita en notación infija.

1. TIPO ABSTRACTO DE DATOS PILA

Las pilas son estructuras de datos lineales, denominadas así, porque consisten en una secuencia de elementos $a_0, a_1, ..., a_{n-1}$ dispuestos en una dimensión. Dependiendo del tipo de elementos que queramos almacenar, podremos tener distintos tipos de pilas. Así, tendremos pilas de enteros, de caracteres, etc, pudiendo tener también pilas de otros TAD's.

Dentro de las estructuras lineales, las pilas también se conocen como estructuras LIFO (Last In, First Out). El nombre LIFO hace referencia al modo en que se accede a los elementos, ya que todas las inserciones y supresiones tienen lugar en un extremo denominado tope.

2. IMPLEMENTACIÓN DE LA CLASE PILA (DE ENTEROS)

Deseamos crear una clase **`PilaEnt`** que represente al TAD Pila de enteros.

Realizaremos dos implementaciones distintas, tal y como se ha visto en clase: Una implementación estática y una implementación dinámica.

En la representación estática preveremos un máximo de 500 enteros.

En cualquier caso, las operaciones de acceso a la pila serán las definidas por el TAD y estarán presentes en la parte pública de la clase, siendo las mismas en el caso estático y en el caso dinámico:

`Pila (void);`
Constructor de la clase. Inicia la pila a vacía.

`bool PilaVacia (void);`
Comprobación de si hay o no elementos en la pila.

`bool Apilar (int x);`
Apilará, si se puede 'x' en la cima de la pila y devolverá false (no se ha producido ningún error.) Si no se puede apilar, devolverá true (se ha producido un error.)

`bool Desapilar (int & x);`
Si hay elementos en la pila, pondrá el valor de la cima en 'x', eliminará un elemento y devolverá false. Si no hay elementos en la pila devolverá true.

`bool CimaPila (int & x);`
Si hay elementos en la pila, pondrá en 'x' el valor que esté en la cima y devolverá true. Si no hay elementos en la pila devolverá false.

Si se considera necesario se implementará también el constructor de copia.

3. EXPRESIONES ALGEBRAICAS

3.1. Notaciones infija y postfija

Dada una cierta expresión algebraica, existen básicamente tres formas diferentes de escribirla, notación

prefija, notación infija y notación postfija, en función de la situación concreta en la que se pongan los operadores respecto de los operandos. Así la expresión algebraica que representa la suma entre un cierto valor A y otro B se podría poner de la siguiente forma:

+ A B	Notación prefija
A + B	Notación infija
A B +	Notación postfija

La notación utilizada habitualmente es la infija.

Si observamos un ejemplo adicional de notación infija en la que se utilicen más de un operador podemos observar que para realizar correctamente la operación tenemos que conocer una información adicional acerca de los operadores que aparezcan: La prioridad. Dependiendo de la prioridad del operador, la operación se realizará antes o después, dando como consecuencia un resultado distinto si variamos la prioridad de los operadores. Así, en A + B * C, se realizará primero la multiplicación y a continuación la suma. Si deseamos variar la prioridad, y en consecuencia el orden de evaluación de las operaciones, hay que añadir una información adicional que son los parentesis. Si en el ejemplo deseamos realizar primero la suma deberemos incluirla entre parentesis (A + B) * C.

Esta inclusión de parentesis no es necesaria en notación prefija o postfija.

En nuestro caso nos centraremos en las notaciones infija (la notación habitual) y la postfija (más conveniente para uso interno en el ordenador.)

Notación infija	Notación postfija
A + B * C	A B C * +
(A + B) * C	A B + C *

La conversión entre la expresión en notación infija y postfija se puede realizar de forma sencilla mediante la utilización de una pila y considerando las prioridades entre los operadores que aparecen en la expresión.

Por ejemplo, supongamos que queremos pasar la expresión infija A + B * C - D a postfija.

El proceso sería el siguiente:

Iniciamos la pila a vacía (pila = <>.)

Obtenemos el primer elemento de la expresión (A). Como es un operando, pasa automaticamente a la salida (Salida = "A".)

Miramos el siguiente elemento (+). Como es un operador, comparamos su prioridad con la del último operador de apilado en la pila. Como la pila está vacía, el operador se apila en la pila (pila = <+>.)

El siguiente elemento es la B, un operando. Como antes pasa directamente a la salida (Salida = "A B".)

*Ahora el elemento es *, un operador. Comparamos su prioridad con la del operador de la cima de la pila. Como el * es más prioritario que el +, lo apilamos, en espera de obtener la otra parte de la operación (pila = <+, *>.)*

El siguiente elemento es C. Va directamente a la salida (Salida = " A B C".)

*Luego tenemos el –. Comparamos su prioridad con la cima de la pila. El '–' tiene menor prioridad que el *, luego desapilamos el asterisco, que va a la salida (pila = <+>, Salida = "A B C *"). Comparamos la prioridad del '–' ahora con la de la cima de la pila (que es el '+'.) Como la prioridad es la misma, desapilamos el + y va a la salida (Salida = "A B C * +".) La pila esta vacía con lo que no podemos seguir comparando y apilamos el '–' (pila = <–>.)*

*El siguiente elemento de la expresión es la D. Va directamente a la salida (Salida = "A B C * + D")*

*Como ya no hay más elementos en la expresión, nos limitamos a desapilar todos los elementos restantes de la pila y los ponemos en la salida (pila = <>, Salida = "A B C * + D –".)*

El proceso con paréntesis sería similar, pero teniendo en cuenta que un parentesis abierto nunca tiene precedencia sobre ningún elemento, de manera que siempre se apila, y el parentesis cerrado desapila todos los símbolos hasta encontrar el parentesis abierto, que también se desapila. Los paréntesis, no deben salir ya en la expresión postfija.

3.2. Evaluación de una expresión algebraica en notación postfija

Una vez transformada la expresión a notación postfija se realizará un algoritmo que la evalue y dé su resultado. La idea básica del algoritmo es ir apilando los operandos y resultados parciales en una pila e ir deapilándolos a medida que van siendo necesarios (cuando encontremos en la expresión un cierto operador.)

Supongamos la expresión "13 7 3 * + 5 –" (en infija sería "13 + 7 * 3 – 5" que tiene como resultado 29.)

La manera de evaluarla será:

Iniciamos la pila a vacía (pila = <>)

Obtenemos el primer valor.

Como es un entero, lo apilamos (pila = <13>)

Obtenemos el segundo elemento y el tercero, que también son enteros y los apilamos (pila = <13, 7, 3>)

Obtenemos el siguiente elemento que es un ''. Como es un operador, necesitamos obtener los dos operandos involucrados que son los dos últimos de la pila, el 3 y el 7 que desapilamos. Los operamos con el * y obtenemos 21. El valor 21 lo apilamos (pila = <13, 21>)*

El siguiente elemento es el +. Necesita de nuevo dos operandos que serán los dos últimos de la pila, el 21 y el 13, que desapilamos. Los operamos con + y apilamos el resultado (pila = <34>)

Después obtenemos el 5 en la expresión. Lo apilamos (pila = <34, 5>)

Finalmente obtenemos de la expresión el '–'. Desapilamos el 5 y el 34, restamos (34-5=29) y apilamos el resultado (pila = <29>)

Como ya hemos llegado al final de la expresión, el resultado de la operación es lo que esté en la cima de la pila.

Si en algún momento nos quedamos sin operandos en la pila cuando es necesario desapilar dos elementos, se ha producido un error.

Si al finalizar el proceso, queda más de un elemento en la pila también se ha producido un error.

En ambos casos la expresión estaba mal escrita.

4. TAREAS A REALIZAR

En esta práctica tenemos que realizar **un solo programa** que pida al usuario una expresión algebraica compuesta por números enteros (de uno o más dígitos) y los operadores '`+`', '`-`', '`*`' y '`/`', la pase a notación postfija y la evalue ayudándose de una pila de enteros, mostrando por pantalla el paso intermedio (expresión en notación postfija) y el resultado final.

Este programa deberá funcionar, indistintamente, con la clase Pila de Enteros Estática (incluida en **`PilaEntE.h`** y **`PilaEntE.cpp`**) y con la clase Pila de Enteros Dinámica (incluida en **`PilaEntD.h`** y **`PilaEntD.cpp`**.)

5. REQUISITOS

Recordar que para poder acceder a la sesión de prácticas correspondiente es necesario entregar un

pequeño esquema (documentación del programa) de lo que se va a realizar en la práctica. Si no se presenta la documentación ANTES de acceder al laboratorio, no se permitirá la entrada a la sesión de prácticas.

En este caso, tanto el algoritmo de paso de notación infija a postfija, como el algoritmo de evaluación de la expresión son especialmente complejos, de manera que se requiere que se realice claramente la especificación (entradas, salidas y tareas que realiza) y el organigrama de ambas funciones.

6. ENTREGA

La entrega de esta tercera práctica debe incluir los siguientes archivos:

1.- **`analiz##.cpp:`** con la función principal y todas las funciones que éste requiera.

2.- **`PilaEntE#.cpp`**, **`PilaEntE##.h`**: Con las implementaciones de la clase 'Pila' estática, y

3.- **`PilaEntD#.cpp`**, **`PilaEntD##.h`**: Con la implementación de la clase 'Pila' dinámica.

La entrega puede realizarse por e-mail a la dirección del profesor encargado del grupo o en disquette al iniciar la siguiente sesión de prácticas, pero siempre ANTES de empezar la siguiente sesión de prácticas.

Cualquier práctica entregada fuera de plazo no será admitida para su corrección.

7. BIBLIOGRAFÍA:

Para la realización de esta práctica podeis consultar la siguiente bibliografía:

- "Estructura de Datos en Pascal"
  A.M. Tenenbaum, M.J. Augenstein. Ed. Prentice Hall.
  (*Explicación del problema y programa resuelto en Pascal)*
- "Estructuras de datos. Libro de problemas"
  L. Joyanes, I. Zahonero, M. Fernández, L. Sánchez. Ed. McGrawHill
  (*Explicación del problema y programa resuelto en Pascal)*
- "Como programar en C/C++"
  H.M. Deitel, P.J. Deitel. Ed. Prentice Hall
  (*Algoritmo de resolución)*
- "Data Structures & Algorithm Analysis in C++"
  M. Allen. Ed. Benjamin/Cummings Publishing Company Inc.
  (*Explicación del problema*)
- "Estructuras de datos en C/C++"
  Y. Langsam, M.J. Augenstein, A.M. Tenenbaum.
  (*Explicación del problema y resolución en C*)

Fecha límite de entrega de prácticas: Sesiones del 24 al 26 de Abril