TIPO ABSTRACTO DE DATOS ***PILA***

Las Pilas son secuencias de elementos que pueden crecer y contraerse siguiendo la política : Ultimo en Entrar, Primero en Salir. A raíz de que entre los elementos de la Pila existe un *orden temporal*, a las pilas se las suele llamar Listas LIFO- Last In First Out, o Listas "último en entrar primero en salir". Otro nombre con el que se las suele conocer es Stack.

Vamos a construir el TAD Pila, que será utilizado para resolver problemas que requieren que se respete el orden mencionado. Dado que la gama de situaciones reales en las que puede usarse este TAD es muy amplia, las consideraciones sobre el tipo de cada uno de los elementos se realizarán en cada aplicación particular.

a) *Especificación*

Pila: Secuencia de cero o mas elementos de un tipo determinado que obedece a la política LIFO.

$P = (a_1, a_2, ..., a_n)$, $n>=0$

El orden temporal de inserciones en este conjunto determina completamente el orden en el cual los elementos serán recuperados. Dado que esta secuencia de elementos puede crecer y contraerse, debemos convenir por cual de los extremos de la secuencia se ingresarán o retirarán elemento (el extremo de la derecha por ejemplo).

P es una abstracción de una realidad que se rige según la política LIFO, por esto podríamos imaginar que P representa a una pila de bandejas de las cuales la última colocada es la primera en ser retirada; en este caso cada a_i representa una bandeja particular.

Si a la secuencia la representamos verticalmente, con a_n como último elemento ingresado, podríamos decir que a_n es el elemento que se encuentra en el tope o cima de la pila. Por ello vamos a llamar tope de la pila al extremo por el cual se ingresan o eliminan los elementos.

Las operaciones asociadas con TAD pila son:

- ***Insertar*** un elemento X en la pila P

Si $P = (a_1, a_2, ..., a_n)$, $n>=0$ y X es el elemento a insertar
Entonces
$P = (a_1, a_2, ..., a_n, , X)$

- ***Suprimir*** un elemento de la pila P

Si $P =(a_1, a_2, ..., a_n)$, $n>0$

Entonces $P = (a_1, a_2, ..., a_{n-1})$ y $X = a_n$

- ***Recorrer*** la pila P

Esta operación procesa todos los elementos de P. El proceso específico que pueda realizarse sobre cada elemento depende de la aplicación particular en la que se usará el TAD Pila. Podríamos considerar como proceso, por ejemplo, la presentación por pantalla de cada elemento.
Hasta este momento hemos definido a la pila como modelo matemático, pues es este modelo el que establece la esencia del TAD.
A partir de la descripción de cada operación abstracta, construimos su especificación.

Sean P una pila y X un elemento:

NOMBRE	*ENCABEZADO*	*FUNCION*	*ENTRADA*	*SALIDA*
Insertar	Insertar (P,X)	Ingresa el elemento X en la pila P	P y X	$P=(a_1,a_2,...,a_n,X)$
Suprimir	Suprimir(P,X)	Si P no está vacía, elimina el elemento que fue insertado mas recientemente	P	$P=(a_1,a_2,...a_{n-1})$ y $X= a_n$ si $n>0$; Error en caso contrario
Recorrer	Recorrer(P)	Procesa todos los elementos de P siguiendo la política LIFO	P	Está sujeta al proceso que se realiza sobre los elementos de P
Crear *	Crear(P)	Inicializa P	P	P=( )
Vacía *	Vacía(P)	Evalúa si P tiene elementos	P	Verdadero si P No tiene elementos, Falso en caso contrario.

*Las operaciones Crear y Vacía, son necesarias cuando se construye el TAD pila.

Un TAD puede ser usado en distintos programas de aplicación. Dado que la persona que diseña e implementa el programa mencionado puede no ser quien construye el TAD, esto no es un obstáculo, ya que para poder usarlo solo se necesita conocer su especificación.
Ahora proponemos un Ejemplo de aplicación del TAD Pila, desde la perspectiva de un programador de aplicaciones que dispone de las especificaciones del TAD.

- Se desea controlar la correspondencia de "**[" / "]" , "{" / "}"** y **"(" / ")"** en una expresión aritmética en la que los identificadores de los operandos están formados por un solo carácter.

Proponemos una solución al problema planteado a través del uso del TAD Pila, en el que el objeto de datos deberá soportar caracteres - `[`, `{` y `(`.
El algoritmo presentado a continuación analiza una expresión recibida como entrada y reporta el mensaje "ERROR DE CORRESPONDENCIA" o "CORRESPONDENCIA", según corresponda.

Sean:

P : Pila de `[`, `{` y `(` ; ***X, aux*** : caracter y ***Expresión*** : expresión aritmética

```
Recuperar ( Expresión, X )
MIENTRAS  (No fin de Expresión)  y  (No Error)
    SI  ( X = " [ " ó  X = " { "  ó   X = "(" )
                ENTONCES
                        Insertar ( P , X)
    FIN SI
    SI ( X = "]"  ó X = " }"  ó   X = ")" )
                ENTONCES
                        Suprimir (P, aux)
                        SI (No Error)
                              ENTONCES
                                   SI (X = " ] "  y   aux ≠ "[" )  ó
                                      (X = " } "  y   aux ≠ "{ " )  ó
                                      (X = " ) "  y   aux ≠ " (" )
                                       ENTONCES
                                            Error
                                   FIN SI
                         FIN SI
    FIN SI
    Recuperar ( Expresión, X )
FIN MIENTRAS
SI ( No (Vacía (P)) ó (Error)
                        ENTONCES
                                " ERROR DE CORRESPONDENCIA"
                        SINO
                                " CORRESPONDENCIA"
FIN SI
```

En este algoritmo la operación Recuperar (Expresión, X), recupera en X cada caracter contenido en Expresión

Expresión de Entrada	Mensaje de Salida
- {[(A-B)*C]^D}	CORRESPONDENCIA
- {([A-B)*C]^D}	ERROR DE CORRESPONDENCIA
- {([A-B)*C]^D	ERROR DE CORRESPONDENCIA
-{[A-B)*C]^D}	ERROR DE CORRESPONDENCIA

Actividades:

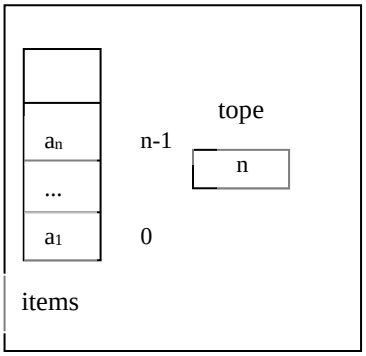
1- ¿Puede resolverse el mismo problema sin usa una pila?, En caso de respuesta afirmativa diseñe el nuevo algoritmo y realice un análisis comparativo entre ambos.

2 - a) Diseñe un algoritmo recursivo que evalúe **a*b** por medio de sumas sucesivas.
b) Diseñe un algoritmo que, apoyado en el TAD Pila, simule el comportamiento del Stack de Recursión para la operación del inciso a)

b) *<u>Representación</u>*

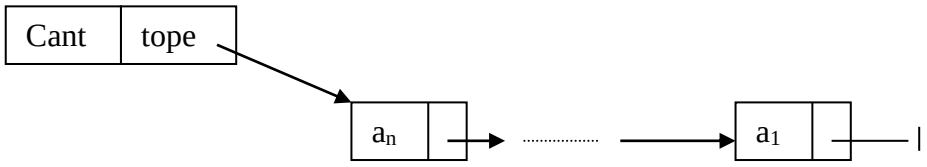
Dado que el objeto de datos de nuestro TAD Pila está constituido como un agregado de componentes, es decir nuestro objeto de datos es una estructura de datos, entonces trabajamos con las representaciones secuencial y encadenada.

- Representación secuencial: en este caso los elementos de la pila podrían ser almacenados en una estructura de arreglo de componentes, en la que cada componente soporte un solo elemento de la pila.
  Para acceder con mas facilidad al componente del arreglo que contiene al elemento que se encuentra en la cima de la pila, adicionaremos un campo, al que identificaremos como *tope*. Por ello, la estructura de datos propuesta para este objeto de datos es un registro compuesto por un arreglo y un campo numérico.



Actividad:
Analice las ventajas y desventajas del uso del campo tope.

- *Representación encadenada:* En esta alternativa cada elemento de la secuencia es almacenado en una celda, que además del dato correspondiente a cada elemento de la pila, tiene un enlace a la celda que contiene al siguiente elemento de la secuencia. Para acceder a los elementos de la secuencia, debe tenerse la dirección del elemento que se encuentra en el tope de la pila. A la vez se mantendrá en el objeto de datos la cantidad de elementos de la secuencia.



La cantidad de elementos que la secuencia puede contener habitualmente, provee una valiosa ayuda al momento de elegir entre las dos representaciones mencionadas.

Actividad
Si bien en la representación encadenada propuesta, el campo *tope* contiene la dirección del elemento a_n. ¿Considera ud. que podría enlazar, indistintamente, al elemento a_1? Justifique su respuesta.

c) *Construcción de operaciones abstractas*

El diseño de los algoritmos correspondientes a las operaciones está sujeto a la representación de almacenamiento. Por ejemplo para la representación secuencial, se incrementa o decrementa el campo *tope* cuando se ingresa o retira un elemento, mientras que para la representación vinculada, hay que disponer de un nuevo espacio en memoria para una inserción, o liberarlo cuando se trata de la supresión de un elemento .

A continuación se presentan dos propuestas de TAD Pila de números enteros:

- Representación secuencial:

```
class pila
{
  int *items;
  int tope;
  int cant;
  public:
  pila(int xcant=0):
  cant(xcant)
  {  tope=-1;
     items=new int[cant];
   }

  int vacia(void)
  {
  return (tope==-1);
   }

  int insertar(int x)
 {
  if (tope<cant-1)
    { items[++tope]=x;
      return (x);
    }
    else return (0);
  }

  int suprimir(void)
 {
   int x;
   if (vacia())
   {
   printf("%s","Pila vacia");
   return(0);
   }
 else
   {
     x=items[tope--];
      return(x);
    }
 }

     void mostrar(void)
 { int i;
   if (!vacia())
   {  for (i=tope; i>=0; i--)
```

```
        {
          cout<<items[i]<<endl;
        }
      }
    }
  };
```

Actividad

Analice la implementación propuesta y a partir de ello:

a. ¿Qué información provee el valor 0-cero- retornado por las operaciones insertar, suprimir y vacía?
b. Identifique cada uno de los tres componentes de un TAD.

- Representación encadenada

Para esta representación, proponemos la construcción del TAD celda que es usado desde el TAD pila construido para la representación encadenada.

```
class celda
{
  int item;
  celda *sig;
  public:
  int obteneritem(void)
  {
   return(item);
  }
  void cargaritem(int xitem)
  {
   item=xitem;
  }
  void cargarsig(celda* xtope)
  {
   sig=xtope;
  }
  celda* obtenersig(void)
  {
   return(sig);
  }
};

class pila
{
  int cant;
  celda *tope;
  public:
  pila(celda* xtope=NULL,int xcant=0):
  tope(xtope),cant(xcant)
  { }

  int vacia(void)
  {
  return (cant==0);
   }
```

```
 int insertar(int x)
 {
  celda *ps1;
  ps1=new(celda);
  ps1->cargaritem(x);
  ps1->cargarsig(tope);
  tope=ps1;
  cant++;
  return(ps1->obteneritem());
  }

int suprimir(void)
 { celda *aux;
 int x;
 if (vacia())
   {
   printf("%s","Pila vacia");
   return(0);
   }
 else
 {
 aux=tope;
  x=tope->obteneritem();
  tope=tope->obtenersig();
  cant--;
  free(aux);
  return(x);
  }
 }
 int muestratope(void)
 {
   return(tope->obteneritem());
 }

 celda* recuperatope(void)
 {
  return(tope);
 }

 void recorrer(celda *aux)
 {
   if (aux!=NULL)
   {  cout<<aux->obteneritem()<<endl;
      recorrer(aux->obtenersig());
   }
 }
 };
```

Actividad

Analice detalladamente los TADs celda y pila, con representación encadenada, presentados en este documento, y proponga:

a. Las modificaciones que a su criterio sean una mejora. Justifique.
b. Una aplicación concreta, y el algoritmo que la materialice implementado en C++.