Tema: TDA Pila

Apellido y Nombre: ..... Fecha:...../...../.....

EJEMPLOS

Ejemplo 1 – Variante de implementación: Modifique la implementación básica de pila de modo que el TDA se construya utilizando únicamente un arreglo de 15 posiciones enteras. Considere que la posición 5 trabaja como cima de la pila mientras que las posiciones anteriores y posteriores a ésta se usan para datos. Desarrolle las operaciones *iniciar_pila*, *agregar_pila* y *pila_llena* adaptadas a la implementación propuesta.

Implementando el TDA pila

Si bien la implementación del TDA debe realizarse utilizando únicamente un arreglo, es necesario que ésta contemple un espacio para almacenar datos (contenedor de datos) y una referencia al último dato (cima o tope). Para ello se destina la quinta posición del arreglo para guardar el valor de cima y las posiciones restantes (anteriores y posteriores) para almacenar los datos.

La definición del TDA pila correspondiente a esta implementación es la siguiente:

```
CONSTANTES
     MAX=15
TIPOS
   tpila=ARREGLO [1..MAX] de ENTERO
```

```
const int MAX=15;
typedef int tpila[MAX];
```

La operación *iniciar_pila* se realiza mediante un procedimiento que inicializa la pila. La inicialización crea una pila vacía asignando a cima el valor adecuado. En este caso, la quinta posición (índice 5 en pseudocódigo, índice 4 en C/C++), que funciona como cima de la pila, recibe el valor de inicialización (cero en pseudocódigo, -1 en C/C++).

```
PROCEDIMIMENTO iniciar_pila(E/S p:tpila)
INICIO
   p[5]←0
FIN
```

```
void iniciar_pila(tpila &p)
{
  p[4]=-1;
}
```

La operación *pila_llena* se realiza mediante una función lógica que verifica la disponibilidad de espacio en la pila. Cuando cima (quinta posición) alcanza la última posición del arreglo (MAX en pseudocódigo, MAX-1 en C/C++) entonces la pila está completa (llena) y la función será verdadera.

```
FUNCIÓN pila_llena(E p:tpila):LOGICO
VARIABLES
   llena: logico
INICIO
   SI (p[5]=MAX) ENTONCES
      llena←V
   SINO
      llena←F
   FIN_SI
   pila_llena←llena
FIN
```

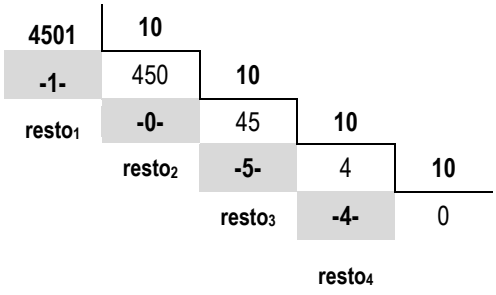
```
bool pila_llena(tpila p)
{bool llena;
  if (p[4]==MAX-1)
    llena=true;
  else
    llena=false;
  return llena;
}
```

La operación *agregar_pila* se realiza mediante un procedimiento que permite añadir un nuevo elemento a la pila. Antes de añadir el nuevo valor es necesario verificar el espacio disponible (mediante la operación *pila_llena*). Si la operación es posible entonces se modifica el valor de cima (ubicada en la quinta posición) y luego se asigna el nuevo elemento a la posición indicada por cima. Aquí es importante considerar que la posición utilizada como cima (p[5] en pseudocódigo, p[4] en C/C++) no debe sobrescribirse con datos. Por esta razón, cuando cima apunta a la quinta posición inmediatamente se corrige su valor para apuntar a una posición de datos.

```
PROCEDIMIENTO agregar_pila(E/S p:tpila,
                           E nuevo: entero)
INICIO
   SI (pila_llena(p)=V) ENTONCES
      ESCRIBIR "PILA LLENA"
   SINO
      p[5]=p[5]+1
      SI (p[5]=5) ENTONCES
        p[5]=p[5]+1
      FIN_SI
      p[p[5]]=nuevo
    FIN_SI
FIN
```

```
void agregar_pila(tpila &p, int nuevo)
{
  if (pila_llena(p)==true)
    cout << "PILA LLENA" << endl;
  else
    { p[4]++;
      if (p[4]==4)
        p[4]++;
      p[p[4]]=nuevo;
    }
}
```

Ejemplo 2 – Aplicación del TDA pila: Considerando que, la extracción de los dígitos de un número entero puede realizarse siguiendo el proceso descripto a continuación (por ejemplo, para el número 4501), desarrolle un algoritmo que aplique el TDA pila y sus operaciones básicas para determinar si un valor indicado por el usuario es capicúa o no.



Proceso

Paso 1: Se divide el número *N* en 10, conservándose el resto obtenido para calcular el número invertido.

Paso 2: Se divide el cociente (entero) obtenido en la división anterior nuevamente por 10, y se conserva el resto para calcular el número invertido.

Paso 3: Se repite el paso 2 hasta que el cociente obtenido sea cero, calculándose entonces el número invertido y comparándose con el original para determinar si es capicúa o no.

El número NO es capicúa (4501 <> 1045).

A fin de resolver el problema planteado (comprobar si un número es capicúa o no) se propone diseñar una función lógica que, utilizando el concepto de pila, genere el valor inverso al original para luego compararlo y determinar si se trata de un capicúa o no.

```
FUNCIÓN capicua(E num: entero):LÓGICO
VARIABLES
     p:tpila
     aux,i,inverso:entero
INICIO
   iniciar_pila(p)
   aux←num
   MIENTRAS (aux > 0) HACER
     agregar_pila(p,aux mod 10)
     aux←aux div 10
   FIN_MIENTRAS
   inverso←0
   i←0
   MIENTRAS (pila_vacia(p) <> V) HACER
     inverso←inverso+quitar_pila(p)*10^i
     i←i+1
   FIN_MIENTRAS
   capicua←num = inverso
FIN
```

```
bool capicua(int num)
{ tpila p;
  int aux,i,
  float inverso;
  iniciar_pila(p);
  aux=num;
  while(aux > 0)
  { agregar_pila(p,aux % 10);
    aux=aux / 10;
  }
  i=0;
  inverso=0;
  while (pila_vacia(p)!=true)
 {inverso=inverso + quitar_pila(p)*pow(10.0,i);
    i++;
 }
  return num==inverso;
}
```

La función implementa el proceso descripto en el enunciado mediante un bucle que almacena cada dígito obtenido en una pila. Luego, en un segundo bucle, se extraen los dígitos de la pila y se multiplican, respectivamente, por 10, 100, 1000, etc. El valor calculado corresponde al inverso del número original. Finalmente, el número y su inverso se comparan para determinar si se trata de un capicúa o no (el resultado de la comparación se asigna a la función).

EJERCICIOS

1) De acuerdo a la definición del TDA pila, implemente el TDA y sus operaciones fundamentales, considerando:
   - TDA pila requiere un contenedor de datos y un indicador del último elemento.

```
tcontenedor=ARREGLO [1..MAX] de ELEMENTOS
tpila=REGISTRO
        datos:tcontenedor
        cima:ENTERO
      FIN_REGISTRO
```

   - Una operación de inicialización que permita crear (inicializar) una pila vacía.
   - Una operación de inserción que permita agregar un nuevo elemento a la pila (siempre como último elemento).
   - Una operación que determine si el contenedor de datos está completo.
   - Una operación que extraiga elementos de la pila (siempre el último elemento almacenado).
   - Una operación que determine si la pila no contiene elementos (pila vacía).
   - Una operación que permita consultar el último elemento almacenado en la pila.

   Suponga que la implementación corresponde a una pila de números enteros.

2) Dadas las siguientes definiciones de TDA pila adapte las operaciones básicas a estas implementaciones.

a)

```
Constantes
    MAX=15
Tipos
tpila=ARREGLO [1..MAX] de ENTEROS
```

Obs: la primera posición se utiliza como indicador de la pila

b)

```
Constantes
    MAX=4
Tipos
tcontenedor=ARREGLO [1..MAX] de CARACTERES
tpila=REGISTRO
        datos1:tcontenedor
        datos2:tcontenedor
        datos3:tcontenedor
        cima:entero
      FIN_REGISTRO
```

c)

```
Constantes
    MAX=7
Tipos
tcontenedor=ARREGLO [1..MAX] de REALES
tpila=REGISTRO
        datos1:tcontenedor
        datos2:tcontenedor
        cima1:entero
        cima2:entero
      FIN_REGISTRO
```

d)

```
Constantes
    MAX=10
Tipos
tcontenedor=ARREGLO [1..MAX] de ENTEROS
tpila=REGISTRO
        datos1:tcontenedor
        datos2:tcontenedor
      FIN_REGISTRO
```

Obs: la primera posición del segundo arreglo se utiliza como indicador de la pila

3) Defina la estructura que permita implementar dos pilas de enteros sobre un mismo vector de 10 elementos, de manera que se aproveche al máximo el vector. Considere que la primera pila almacena elementos desde la posición inicial del contenedor hacia la última, mientras que la segunda pila almacena datos desde la última posición hacia la primera. Además desarrolle las operaciones básicas, teniendo en cuenta que las operaciones deben contar con un parámetro adicional que indique sobre qué pila se ejecutan (1 ó 2). Controle que las pilas no se sobre escriban.

4) Utilizando listas simples implemente un *TDA pila* de valores caracteres y las operaciones *iniciar_pila*, *agregar_pila, quitar_pila* y *tope_pila*. Para llevar a cabo la implementación considere las siguientes variantes:

   a) la lista cuenta con un único puntero de *inicio*
      - las operaciones de **inserción/eliminación** se realizan por el **principio** de la lista.
      - las operaciones de **inserción/eliminación** se realizan por el **final** de la lista.

b) la lista cuenta con punteros de *inicio* y *final*
   - las operaciones de **inserción/eliminación** se realizan por el **principio** de la lista.
   - las operaciones de **inserción/eliminación** se realizan por el **final** de la lista.

¿Cuál de las variantes resulta más sencilla? Justifique su respuesta.

5) Aplicando el TDA pila y sus operaciones básicas, desarrolle las estructuras y algoritmos que permitan detectar si una cadena ingresada por el usuario es un palíndromo o no.

Cadena ingresada: OREJERO

La cadena ingresada es un PALÍNDROMO!!!

Observación: Un palíndromo es una palabra que puede leerse igual de izquierda a derecha que de derecha a izquierda, por ejemplo: neuquen, oso, ojo, ala, reconocer, etc.

6) Dada una cadena de caracteres, como la indicada a continuación, formada exclusivamente por llaves de apertura y cierre:

{ { } { { { } } } } ...

Verifique la correcta parentización de la cadena, considerando que por cada llave de apertura, debe existir la correspondiente llave de cierre. Para llevar a cabo esta verificación se procede de la siguiente forma: Se lee carácter a carácter la cadena, almacenando las llaves de apertura en una pila hasta encontrar una llave de cierre, en cuyo caso se extrae la última llave de apertura almacenada. Este proceso se repite hasta finalizar la cadena.

Las siguientes situaciones indican una parentización INCORRECTA:

- al leer una llave de cierre e intentar la extracción de la correspondiente llave de apertura de una pila vacía, o
- al finalizar el recorrido de la cadena, determinar que aún hay llaves de apertura en la pila.

En virtud de lo expuesto, se solicita:

a) defina las estructuras de datos necesarias para resolver la situación planteada.
b) desarrolle los algoritmos que verifiquen la parentización de una cadena ingresada por el usuario.

7) Normalmente, las expresiones matemáticas se especifican en notación interfija siguiendo el formato:

OPERANDO OPERADOR OPERANDO (Por ejemplo: 3 * 7)

Una notación alternativa a ésta, y que permite eliminar el uso de paréntesis para indicar la prioridad de las operaciones, es la notación posfija. Esta notación sigue el formato:

OPERANDO OPERANDO OPERADOR (Por ejemplo: 3 7 *)

Conversión de expresiones de notación interfija a notación posfija

El proceso de conversión utiliza una cadena de salida (cadena posfija) y una pila de operadores. Durante este proceso se analiza carácter por carácter los elementos de la expresión y según se trate de un operando u operador se realiza lo siguiente:

- Cuando se lee un operando éste se almacena directamente en la cadena posfija.
- Cuando se lee un operador éste puede almacenarse en la pila de operadores o provocar la extracción de operadores ya almacenados (en base a su prioridad). En forma general, si el operador leído tiene mayor prioridad que el de la cima de la pila, entonces debe almacenarse en ella. Por el contrario, si el operador tiene menor prioridad que el de la cima de la pila entonces se extrae el operador almacenado y se inserta en la cadena posfija. Esto se repite hasta que el operador leído tenga mayor prioridad que el de la cima de la pila o hasta que ésta quede vacía (guardándose entonces el operador leído).
- Al finalizar la lectura de la cadena interfija, si aún restan operadores en la pila éstos se desapilan y almacenan en la cadena posfija.

Por ejemplo, dada la expresión interfija **3 * 4 ^ 2 – 8 / 4 * 6 + 7 ^ 2** al aplicar la conversión a notación posfija se obtendrá

3 4 2 ^ * 8 4 / 6 * - 7 2 ^ +

Teniendo en cuenta lo descripto, realice lo siguiente:

a) dada la expresión interfija **9 / 2 ^ 3 - 4 + 3 * 7** desarrolle **gráficamente** (y paso a paso) el proceso de conversión de la expresión anterior a notación posfija, utilizando para ello el concepto de ***PILA***.

b) desarrolle un programa que utilizando el TDA pila permita realizar la conversión de una expresión interfija a posfija.

Observación: Considere que las expresiones sólo estarán formadas por números de un dígito (0-9) y los operadores **+**, **-**, *****, **/** y **^**.

8) Dada una expresión en notación posfija (*operando1 operando2* ***operador***) es posible calcular su resultado aplicando el concepto de PILA como se indica a continuación:

- Cuando se lee un **operando** en la expresión de entrada éste se almacena en una **pila de operandos**.
- Cuando se lee un **operador** en la expresión de entrada se extraen dos **operandos** de la **pila de operandos** y se resuelve la operación correspondiente, almacenándose el resultado obtenido nuevamente en la pila. Esto se repite por cada operador leído en la cadena de entrada.
- Al finalizar la lectura de la cadena posfija la **pila de operandos** contiene un único elemento que es el resultado final de la expresión.

Por ejemplo, dada la expresión posfija **8 5 4 * 2 / + 2 6 *** - al aplicar el proceso de cálculo se obtendrá ***6***.

Teniendo en cuenta lo descripto, realice lo siguiente:

a) dada la expresión posfija **9 3 2 ^ / 4 5 * - 6 + 8 2 /** - desarrolle **gráficamente** (y paso a paso) el proceso de cálculo de la expresión anterior, utilizando para ello el concepto de ***PILA***.

b) desarrolle un programa que utilizando el TDA pila calcule el valor de la expresión.

9) La sucesión de Fibonacci, una sucesión matemática infinita, consta de una serie de números naturales dónde cada término se obtiene sumando los 2 términos precedentes, salvo los 2 primeros que valen 1. Por ejemplo, el sexto término (8) se calcula como la suma de los valores correspondientes al cuarto (3) y quinto (5) término.

- Sucesión de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Utilizando el *TDA pila* y sus operaciones básicas codifique un algoritmo que permita calcular un término cualquiera de la secuencia de Fibonacci.

¿Cómo modificaría el algoritmo anterior si debe adaptarlo a la serie numérica 1, 3, 5, 9, 17, 31, 57, ...? Considere que cada término de la serie se genera como suma de los 3 términos precedentes, salvo los 3 primeros que valen 1, 3 y 5 respectivamente.

10) Dada la siguiente definición de datos, diseñe un algoritmo que permite invertir el contenido de una lista doble (con un único puntero al *inicio*) utilizando el *TDA pila*. Tenga en cuenta que los *nodos* cambiarán de posición al invertir la lista.

```
const int MAX=50;
typedef struct tnodo *pnodo;
typedef struct tnodo {char dato;
                      pnodo ant;
                      pnodo sig; };
typedef pnodo tcontenedor[MAX];
typedef struct tpila {tcontenedor datos;
                      int cima;
                      };
```

11) Dado el TDA pila y sus operaciones básicas

a) Utilice una implementación basada en **arreglos** para

- calcular el factorial de un número
- calcular el inverso de un número entero
- determinar si un número es capicúa o no

b) Utilice una implementación basada en **listas** (simples o dobles) para

- calcular la suma de los dígitos de un número entero.
- calcular un término de la serie de Fibonacci.
- calcular el valor de una expresión posfija.

12) Analice el siguiente fragmento de código, describa las acciones que realiza y determine su propósito:

a)
```
int misterio (int m, int n)
{
     tpila p;
     int s;

     iniciar_pila(p);
      while (n>0)
     {
         agregar_pila(p,m);
         n--;
     }
     s=0;
     while (pila_vacia(p)==false)
        s=s + extraer_pila(p);
     return s;
}
```

b)
```
int incognita (int numero)
{
    tpila p;
    pnodo n,m;
    int i,s;

    iniciar_pila(p);
    while (numero > 0)
    {
       numero=numero / 10;
       crear_nodo(n,1);
       agregar_pila(p,n);
    }
    s=0;
    while (pila_vacia(p)==false)
    {
       m=extraer_pila(p);
       s=s + m->dato;
       delete(m);
    }
    return s;
}
```

c)
```
int digito(char valor)
{
  int salida;
  switch (valor)
  {
     case '0': salida=0; break;
     case '1': salida=1; break;
     case '2': salida=2; break;
     case '3': salida=3; break;
     case '4': salida=4; break;
     case '5': salida=5; break;
     case '6': salida=6; break;
     case '7': salida=7; break;
     case '8': salida=8; break;
     case '9': salida=9; break;
  }
  return salida;
}
```

```
float enigma (tcad num)
{
    tpila p;
    int j,i;
    float s;
    iniciar_pila(p);
    for(i=0;i < strlen(num);i++)
       agregar_pila(p,digito(num[i]));
    s=0;
    j=0;
    while (pila_vacia(p)==false)
    {  s=s + extraer_pila(p)*pow(10.0,j);
       j++;
    }
    return s;
}
```

‘2’ ‘0’ ‘1’ ‘8’

num