

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	<code>const int MAX=15;</code>
<code>MAX=15</code>	<code>typedef int tpila[MAX];</code>
TIPOS	
<code>tpila=ARREGLO [1..MAX] de ENTERO</code>	

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++).

PROCEDIMIENTO iniciar_pila (E/S p:tpila)	<code>void iniciar_pila(tpila &p)</code>
INICIO	{
<code>p[5] ← 0</code>	<code>p[4] = -1;</code>
FIN	}

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	<code>bool pila_llena(tpila p)</code>
VARIABLES	{bool llena;
<code>llena: logico</code>	<code>if (p[4]==MAX-1)</code>
INICIO	<code>llena=true;</code>
<code>SI (p[5]=MAX) ENTONCES</code>	<code>else</code>
<code>llena ← V</code>	<code>llena=false;</code>
SINO	<code>return llena;</code>
<code>llena ← F</code>	}
FIN_SI	
<code>pila_llena ← llena</code>	
FIN	

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

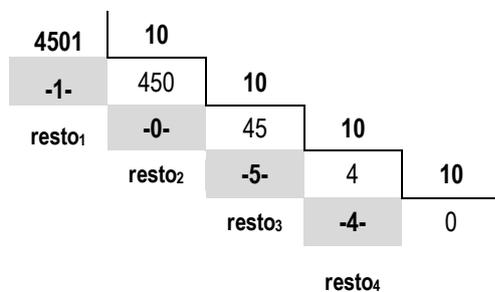
```

```

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 descrito 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 ($4501 \neq 1045$).

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 descrito 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 caracteres.

2) Dadas las siguientes definiciones de TDA pila adapte las operaciones básicas según lo especificado. Luego genere archivos *hpp* para cada una de ellas.

a)

Constantes

MAX=15

Tipos

tcontenedor=ARREGLO [1..MAX] de CARACTERES

tpila=REGISTRO

datos:tcontenedor

cima:entero

FIN_REGISTRO

Obs: los datos se cargan desde las últimas posiciones hacia las primeras del arreglo.

b)

Constantes

MAX=10

Tipos

tpila=ARREGLO [1..MAX] de ENTEROS

Obs: la primera posición del arreglo se utilizará como indicador de la pila.

d)

Constantes

MAX=6

Tipos

tcontenedor=ARREGLO [1..MAX] de REALES

tpila=REGISTRO

datos1:tcontenedor

datos2:tcontenedor

datos3:tcontenedor

cima:ENTERO

FIN_REGISTRO

c)

Constantes

MAX=5

Tipos

tcontenedor=ARREGLO [1..MAX] de ENTEROS

tpila=REGISTRO

datos1:tcontenedor

datos2:tcontenedor

FIN_REGISTRO

Obs: la última posición del segundo arreglo se utilizará como indicador de la pila

3) Considerando que en el siguiente programa se implementó el TDA pila mediante listas enlazadas identifique:

- Variante de lista utilizada (simple, doble, único puntero, punteros de inicio y final, circular, etc.)
- Operaciones básicas de listas adaptadas para obtener las del TDA pila
- Objetivo del programa

¿Será posible, haciendo mínimas modificaciones, ejecutar este programa usando las implementaciones del ejercicio 2?

```

#include <iostream>

using namespace std;

typedef struct tpila *ppila;
typedef struct tpila{
    int dato;
    ppila sig;
};

void iniciar_pila(ppila &p);
void crear_nodo(ppila &nuevo,int valor);
void agregar_pila(ppila &p,int nuevo);
int quitar_pila(ppila &p);
int tope_pila(ppila p);
bool pila_vacia(ppila p);

main()
{ ppila p;
  int num,salida=1;
  cout << "Ingrese un numero:";
  cin >> num;
  iniciar_pila(p);
  while (num>0)
  { agregar_pila(p,num);
    num--;
  }
  while (pila_vacia(p)!=true)
    salida=salida*quitar_pila(p);
  cout << "Resultado: " << salida << endl;
}

void iniciar_pila(ppila &p)
{
  p=NULL;
}

void crear_nodo(ppila &nuevo,int valor)
{
  nuevo=new tpila;
  if (nuevo!=NULL)
  { nuevo->dato=valor;
    nuevo->sig=NULL;
  }
}

void agregar_pila(ppila &p,int valor)
{ ppila nuevo;
  crear_nodo(nuevo,valor);
  if (nuevo==NULL)
    cout << "PILA LLENA" << endl;
  else
  { nuevo->sig=p;
    p=nuevo;
  }
}

int quitar_pila(ppila &p)
{ int extraido;
  ppila nodo;
  if (pila_vacia(p)==true)
    extraido=0;
  else
  { extraido=p->dato;
    nodo=p;
    p=p->sig;
    nodo->sig=NULL;
    delete(nodo);
  }
  return extraido;
}

int tope_pila(ppila p)
{ int tope;
  if (pila_vacia(p)==true)
    tope=0;
  else
    tope=p->dato;
  return tope;
}

bool pila_vacia(ppila p)
{
  return p==NULL;
}

```

- 4) En base al ejercicio 3, genere librerías *hpp* para el TDA *pila* considerando las siguientes variantes de listas:
- listas simples (único puntero, punteros de inicio y final, circular)
 - 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.
 - lista dobles (único puntero, punteros de inicio y final, circular)
 - 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.

- 5) Aplicando el TDA pila y sus operaciones básicas, diseñe un programa que detecte si una cadena de caracteres es o no un palíndromo. Considere que el programa omite diferencias entre mayúsculas y minúsculas e ignora los símbolos especiales. Compruebe el funcionamiento del programa utilizando las librerías *hpp* (estáticas y dinámicas) generadas en los primeros ejercicios para el *TDA pila*.

Cadena ingresada: *Anita lava la tina*
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) Aplicando el TDA pila y sus operaciones básicas, diseñe un programa que obtenga el valor invertido de un número entero ingresado por el usuario. Compruebe el funcionamiento del programa utilizando las librerías *hpp* (estáticas y dinámicas) generadas en los primeros ejercicios para el *TDA pila*.

Número ingresado: 2015
Número invertido: 5102

- 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 el proceso se analiza carácter por carácter los elementos de la expresión y según se trate de un operando u operador se procede:

- 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) dadas las siguientes expresiones interfijas desarrolle **gráficamente** (y paso a paso) el proceso de conversión de éstas a notación posfija, utilizando para ello el concepto de **PILA**.

- $3 ^ 3 / 9 - 4 * 2 + 1$
- $3 * 5 - 8 / 2 + 4 ^ 2$
- $4 + 3 ^ 2 + 9 * 2 / 3 - 1$

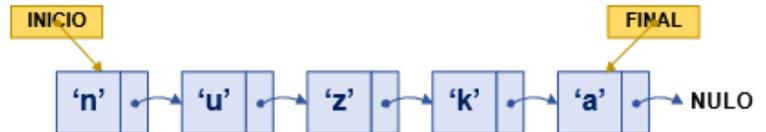
- b) desarrolle un programa que, utilizando el TDA pila y sus operaciones básicas, permita realizar la conversión de una expresión interfija a posfija. Verifique el funcionamiento del programa utilizando las librerías *hpp* (estáticas y dinámicas) generadas en los primeros ejercicios para el *TDA pila*.

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


```

typedef struct tnode *pnode;
typedef struct tnode {
    char letra;
    pnode sig;
};
typedef struct tlista {pnode inicio;
    pnode final;
    int contador;
};
typedef pnode tcontenedor[30];
typedef struct tpila{
    tcontenedor datos;
    int cima;
};

```



12) Dados los siguientes módulos realice la prueba de escritorio de éstos y determine su propósito. Compruebe los resultados obtenidos utilizando las librerías *hpp* (estáticas y dinámicas) generadas en los primeros ejercicios para el TDA pila.

```

a) int oculto (int n)
{
    tpila p;
    int r=0,i=1;
    iniciar_pila(p);
    while (n > 0)
    { if (i%2!=0)
        { agregar_pila(p,i);
          n--;
        }
      i++;
    }
    while (pila_vacia(p)==false)
    { r=r+quitar_pila(p);
    }
    return r;
}

```

```

b) void incognita (tcad &c)
{
    tpila p;
    int l;
    iniciar_pila(p);
    l=strlen(c)-1;
    while (l >= 0)
    { if (c[l]>='A' && c[l]<='Z')
        { agregar_pila(p,c[l]+32);
        }
      else
        { agregar_pila(p,c[l]);
        }
      l--;
    }
    l=0;
    while (pila_vacia(p)==false)
    { c[l]=quitar_pila(p);
      l++;
    }
}

```

```

c) bool misterio (tcad n)
{
    bool b;
    tpila p;
    int i=0;
    iniciar_pila(p);
    while (i < strlen(n)/2)
    { agregar_pila(p,n[i]);
      i++; }
    b=true;
    if (strlen(n)%2!=0)
        i++;
    while (pila_vacia(p)==false && b==true)
    { if (n[i]!=quitar_pila(p))
        { b=false;
        }
      else
        { i++;
        }
    }
    return b;
}

```

```

d) bool misterio(int num)
{
    tpila p1,p2;
    bool b=true;
    int s1=0,s2=0;
    while (num>9)
    {
        iniciar_pila(p1);
        iniciar_pila(p2);
        while (num>0)
        {
            if (b==true)
            {
                agregar_pila(p1,num%10);
                b=false;
            }
            else
            {
                agregar_pila(p2,num%10);
                b=true;
            }
            num=num/10;
        }
        s1=0;
        s2=0;
        while (!pila_vacia(p1) || !pila_vacia(p2))
        {
            if (!pila_vacia(p1))
                s1=s1+quitar_pila(p1);
            if (!pila_vacia(p2))
                s2=s2+quitar_pila(p2);
        }
        num=abs(s1-s2);
    }
    return num==0;
}

```

