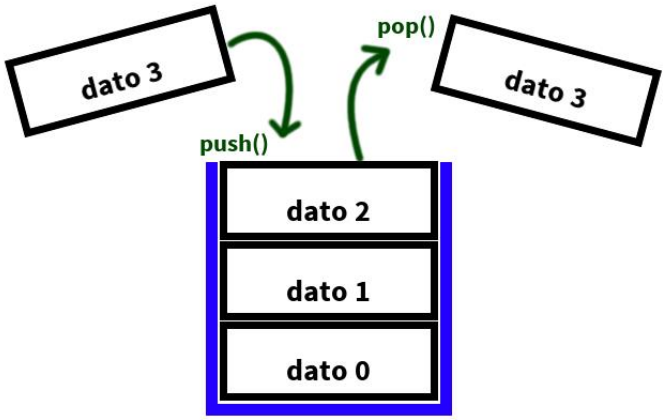
Ontiveros Lara Claudia Sarahi 17212167

Pilas

Una pila (stack en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite *almacenar y recuperar datos*. Se aplica en multitud de ocasiones en informática debido a su simplicidad y ordenación implícita en la propia estructura.
Para el manejo de los datos se cuenta con dos operaciones básicas: apilar (push), que coloca un objeto en la pila, y su operación inversa, retirar (o desapilar, pop), que retira el último elemento apilado.



En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado TOS, Top of Stack en inglés). La operación retirar permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo TOS.

En general, una pila tiene su utilidad cuando interesa recuperar la última información generada (el estado inmediatamente anterior).

Asociadas con la estructura pila existen una serie de operaciones necesarias para su manipulación. Éstas son:

Iniciación de la estructura:

- Crear la pila (*CrearPila*): La operación de creación de la pila inicia la pila como vacía.

Operaciones para añadir y eliminar información:

- Añadir elementos en la cima (*Apilar*): pondrá un nuevo elemento en la parte superior de la pila.
- Eliminar elementos de la cima (*Desapilar*): lo que hará será eliminar el elemento superior de la pila.

Operaciones para comprobar tanto la información contenida en la pila, como el propio estado de la cima:

- Comprobar si la pila está vacía (*PilaVacia*): Esta operación es necesaria para verificar la existencia de elementos de la pila.

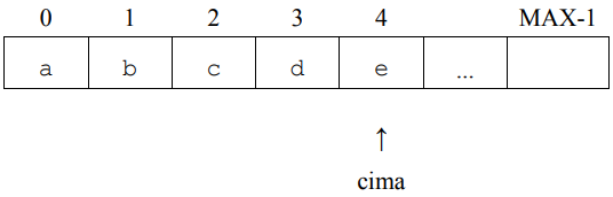
- Acceder al elemento situado en la cima (*CimaPila*): Devuelve el valor del elemento situado en la parte superior de la pila.

Como en el caso de cualquier contenedor de datos las operaciones relevantes tienen que ver con el almacenamiento (*Apilar*), eliminación (*Desapilar*) o consulta (*CimaPila*) de información.

La especificación correcta de todas estas operaciones permitirá definir adecuadamente el tipo pila.

Ejemplos:

Sea una pila p cuyos elementos son (a, b, c, d, e), siendo e su cima. La representación gráfica de la implementación de p mediante un array sería: p = (a, b, c, d, e)




```
class Pila
{
        public:
            Pila ();
            bool Apilar (Valor);
            bool Desapilar ();
            bool CimaPila (Valor &);
            bool PilaVacia ();
        private:
            typedef Valor Vector[MAX];
            Vector datos;
            int cima;
};
```

Operación CrearPila

La creación de la pila se realizará mediante el constructor por defecto. La tarea que deberá realizar será establecer un estado inicial en el que no existen elementos en la pila:

```
Pila::Pila ()
{
        cima = -1;
}
```

Operación PilaVacia

Esta operación permitirá determinar si la estructura tiene o no elementos almacenados. Aunque el array empleado tiene un número de elementos fijo (MAX), no todos ellos representan valores almacenados en la pila. Sólo están almacenados en la pila los valores comprendidos entre los índices 0 (fondo de la pila) y cima. Por lo tanto, la pila estará vacía cuando la cima indique una posición por debajo del fondo:

```
bool Pila::PilaVacia ()
{
        return (cima == -1);
}
```

Operación Apilar

La operación de inserción normalmente se conoce por su nombre inglés Push, o Apilar. La operación, aplicada sobre un pila y un valor, almacena el valor en la cima de la pila. Esta operación está restringida por el tipo de representación escogido. En este caso, la utilización de un array implica que se tiene un número máximo de posibles elementos en la pila, por lo tanto, es necesario comprobar, previamente a la inserción, que realmente hay espacio en la estructura para almacenar un nuevo elemento. Con está consideración, el algoritmo de inserción sería:

```
bool Pila::Apilar (Valor x)
{
        bool ok;
        if (cima == MAX)
            ok = false;
        else
        {
            cima++;
            datos[cima] = x;
            ok = true;
        }
        return (ok);
}
```

Operación Desapilar

La operación de borrado elimina de la estructura el elemento situado en la cima. Normalmente recibe el nombre de Pop en la bibliografía inglesa. Eliminar un elemento de la pila consiste fundamentalmente en desplazar (decrementar) la

cima. Puesto que no es posible eliminar físicamente los elementos de un array, lo que se hace es dejar fuera del rango válido de elementos al primero de ellos.

```
bool Pila::Desapilar (void)
{
        bool ok;
        if (cima == -1)
            ok = false;
        else
        {
            cima--;
            ok = true;
        }
        return (ok);
}
```

Operación CimaPila

La operación de consulta permite conocer el elemento almacenado en la cima de la pila, teniendo en cuenta que si la pila está vacía no es posible conocer este valor.

```
bool Pila::CimaPila (Valor & x)
{
        bool ok;
        if (cima == -1)
            ok = false;
        else
        {
            x = datos[cima];
            ok = true;
        }
        return (ok);
}
```

Array:

ADS

```
generic type TDato is private;
 package Pila_generica is
          type TPila (Maximo : Positive) is private;
          Llena, Vacia, FueraDeRango : exception;
          procedure Push ( Pila : in out TPila; Dato : in TDato );
          procedure Pop ( Pila : in out TPila; Dato: out TDato );
```

```
        function PilaVacia (Pila : in TPila) return Boolean;
        function PilaLlena (Pila : in TPila) return Boolean;
        function Valor ( Pila : in TPila; Index : Positive ) return TDato;

 private
        type TVector is array (Positive range <>) of TDato;
        type TPila (Maximo : Positive) is record
                    Contenido : TVector (1 .. Maximo);
            Cima : Natural := 0;
         end record;
 end;
```

En Python:

```
class Stack(object):
def __init__(self):
self.stack_pointer = None

def push(self, element):
self.stack_pointer = Node(element, self.stack_pointer)

def pop(self):
e = self.stack_pointer.element
self.stack_pointer = self.stack_pointer.next
return e

def peek(self):
return self.stack_pointer.element

def __len__(self):
i = 0
sp = self.stack_pointer
while sp:
i += 1
sp = sp.next
return i

class Node(object):
def __init__(self, element=None, next=None):
self.element = element
self.next = next
```

```
if __name__ == '__main__':
# small use example
s = Stack()
[s.push(i) for i in xrange(10)]
print [s.pop() for i in xrange(len(s))]
```

Referencias:

Cairó, O., & Guardati, S. (2002). Estructuras de datos (3rd ed.). México: McGrawHill.

Joyanes Aguilar, L., & Zahonero Martínez, I. (1998). Estructura de datos. Algoritmos, abstracción y objetos. (Ed. rev.). Madrid, España: McGraw-Hill.