Pilas (Stacks):

Una pila es un tipo especial de lista enlazada en la que sólo se pueden insertar y eliminar nodos en uno de los extremos de la lista, al que llamaremos "TOPE" o "CIMA".

Las pilas ofrecen dos operaciones fundamentales, que se conocen como "push" o "apilar" y "pop" o "desapilar", respectivamente "empujar" y "tirar". Además, las escrituras de datos siempre son inserciones de nodos, y las lecturas siempre eliminan el nodo leído.

Estas características implican un comportamiento de lista LIFO (Last In First Out), el último en entrar es el primero en salir.

El nombre de la estructura deriva del uso coloquial, cuando decimos: una pila de platos que lavar, una pila de libros en nuestra biblioteca, etc. Sólo es posible añadir platos en la parte superior de la pila en el fregadero, y sólo pueden tomarse del mismo extremo (lo contrario, podría ser desastroso).

En la ejecución de los programas estas estructuras son fundamentales. La recursividad se implementa en un computador con la ayuda de un "snack" o "pila de activación".

El uso que se les da a las pilas es independiente de su implementación, es decir, se hace un encapsulamiento. Por eso se considera a la pila como un tipo abstracto de datos.

Representación de Pilas

Su representación en C, es como sigue:

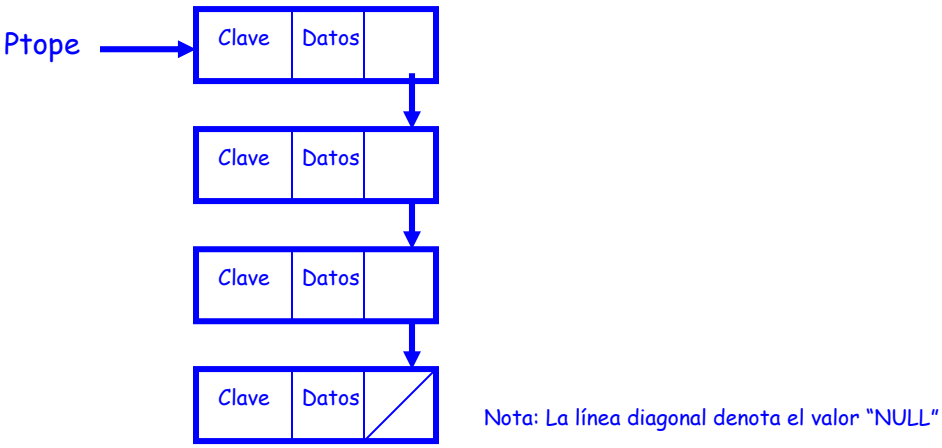
```
typedef struct _nodo {
  int dato;
  struct _nodo *proximo;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Pila;
```

tipoNodo es el tipo para declarar nodos, evidentemente.
pNodo es el tipo para declarar punteros a un nodo.
Pila es el tipo para declarar pilas.

Hay dos representaciones posibles, dependiendo de la orientación de los nodos:

Primera representación

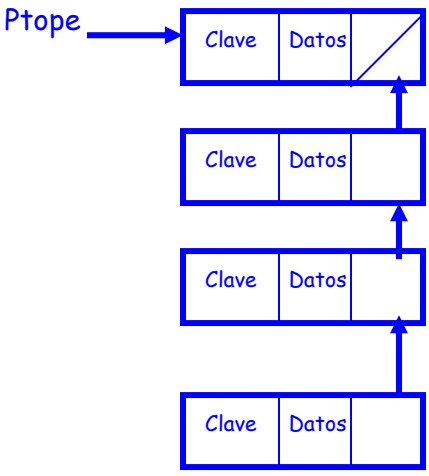


Nota: La línea diagonal denota el valor "NULL"

Viendo el gráfico, es evidente, que en esta representación, una pila es una lista donde Ptope es el apuntador al primer elemento. (en listas enlazadas lo llamamos, "primero"). Este apuntador, indica el extremo donde se permitirán las eliminaciones e inserciones de nuevos elementos o nodos.

Si escogemos esta representación, la función APILAR se limita al algoritmo de inserción en la primera posición de una lista simplemente enlazada. La función DESAPILAR, sería la eliminación del primer elemento de la lista.

Segunda representación



En esta representación, una pila es una lista donde Ptope es el apuntador al último elemento.

Si escogemos esta representación, la función APILAR se limita al algoritmo de inserción en la última posición de una lista simplemente enlazada, con la ventaja que conocemos esta posición, ya que está apuntada por Ptope. La función DESAPILAR, sería la eliminación del último elemento de la lista.

Operaciones básicas con pilas:

Las operaciones primitivas que permite el tipo pila son:

- Apilar (x, P) o PUSH: Añadir el elemento "x" en el tope o cima de la pila P.
- Desapilar (P) o POP: Eliminar un elemento en la cima de la pila P.
- Tope (p) : Devuelve el valor del elemento de la parte superior de la pila P.
- Vacia (P): Devuelve "verdadero" si la pila P está vacía y "falso", si no lo está.

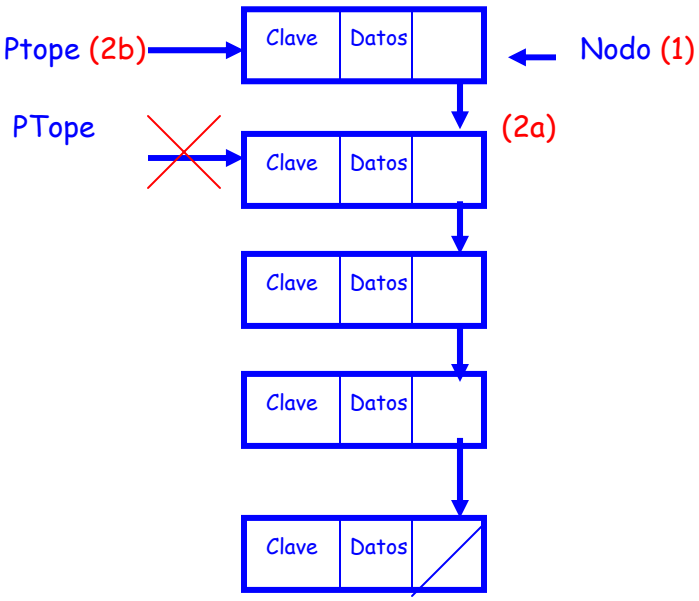
A continuación, veremos en detalle las funciones de apilar y desafilar que son las más importantes usando la primera representación descrita anteriormente.

INSERCIÓN (APILAR o PUSH)

Apilar implica insertar un elemento en el tope de la pila. Sólo debe hacerse la distinción si la pila está vacía o no

Algoritmo APILAR (x,P)

1. Crear el nuevo nodo y asignarle la clave x
2. Si la pila P esta vacía, es decir, Ptope = NULL, se hace
    a. nodo->proximo se hace apuntar a NULL.
    b. Ptope debe apuntar a nodo.
3. Si la pila P no está vacía, entonces
    a. nodo->proximo debe apuntar a Ptope.
    b. Ptope debe ahora apuntar a nodo, ya que éste es el último nodo insertado.

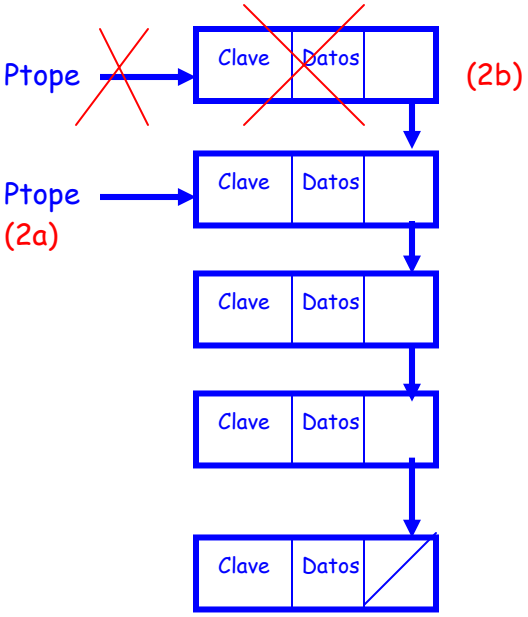


ELIMINACIÓN (DESAPILAR O POP)

Sólo podemos eliminar el elemento que se encuentra en la cima. De lo contrario estaríamos violando el concepto de "pila". Por lo tanto el nodo a eliminar será siempre el que apunte a Ptope.

1. Si la pila no está vacía
    a. Asignamos a Ptope la dirección del segundo nodo de la pila: Ptope->proximo.
    b. Liberamos la memoria asignada al primer nodo, el que queremos eliminar.

Si la pila sólo tiene un elemento, el proceso sigue siendo válido, ya que el valor de Pila->proximo es NULL, y después de eliminar el último nodo la pila quedará vacía, y el valor de Pila será NULL.



OBTENCIÓN O LECTURA DE UN ELEMENTO (TOPE)

Esta función suele usarse antes de desafilar para obtener el elemento que va a ser eliminado. Es muy sencilla, lo que hace es devolver el valor clave del nodo que apunta Ptope.

CODIFICACIÓN DE PILAS-STACKS en C

```
#include <stdlib.h>
#include <stdio.h>

typedef struct _nodo {
  int clave;
  struct _nodo *proximo;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Pila;

/* Funciones con pilas: */
void Apilar(Pila *l, int elem);
int Desapilar (Pila *l);
int Tope(Pila *l);


/* ESTO ES SIMPLEMENTE UN EJEMPLO PARA PROBAR LA FUNCIONALIDAD DE LAS RUTINAS */

int main()
{
  Pila pila = NULL;
  pNodo p;

  Apilar(&pila, 100);
  Apilar(&pila, 80);
  Apilar(&pila, 25);
  Apilar(&pila, 30);
  Apilar(&pila, 40);


  printf("%d, ", tope(&pila));
  Desapilar (&pila);
  printf("%d, ", tope(&pila));
  Desapilar (&pila);
  printf("%d, ", tope(&pila));
  Desapilar (&pila);
  printf("%d\n", tope(&pila));

  system("PAUSE");
  return 0;
}
```

```
void Apilar(Pila *pila, int elem)
{
  pNodo nuevo;

  /* Crear un nodo nuevo */
  nuevo = (pNodo)malloc(sizeof(tipoNodo));
  nuevo->clave = elem;

  /* Añadimos la pila a continuación del nuevo nodo */
  nuevo->proximo = *pila;
  /* Ahora, el comienzo de nuestra pila es en nuevo nodo */
  *pila = nuevo;
}

void Desapilar(Pila *pila)
{
  pNodo nodo; /* variable auxiliar para manipular nodo */

  /* Nodo apunta al primer elemento de la pila */
  nodo = *pila;
  if(nodo)        /* la pila debe tener elementos para poder eliminar */
  /* Asignamos a pila toda la pila menos el primer elemento */
  *pila = nodo->proximo;
   /* Borrar el nodo */
  free(nodo);
  }

int Tope (Pila *pila)
{
  pNodo nodo; /* variable auxiliar para manipular nodo */
  int elem;     /* variable auxiliar para retorno */

  /* Nodo apunta al primer elemento de la pila */
  nodo = *pila;
  if(!nodo) return 0; /* Si no hay nodos en la pila retornamos 0 */
    /* Guardamos el valor de la clave de retorno */
  elem = nodo->clave;
  return elem;
}
```

CODIFICACIÓN DE PILAS-STACKS en C++

```
#include <iostream>
using namespace std;

class nodo {
  public:
   nodo(int elem, nodo *sig = NULL)
   {
     clave = elem;
     proximo = sig;
   }

  private:
   int clave;
   nodo *proximo;

  friend class pila;
};

typedef nodo *pnodo;

class pila {
  public:
   pila() : tope(NULL) {}
   ~pila();

   void Apilar(int elem);
   void Desapilar();
   int tope();

  private:
   pnodo ptope;
};

pila::~pila()
{
  pnodo aux;

  while(ptope) {
    aux = ptope;
    ptope = ptope->proximo;
    delete aux;
  }
}
```

```
void pila::Apilar(int elem)
{
  pnodo nuevo;

  /* Crear un nodo nuevo */
  nuevo = new nodo(elem, ptope);
  /* Ahora, el comienzo de nuestra pila es en nuevo nodo */
  ptope = nuevo;
}

void pila::Desapilar()
{

  pnodo nodo;    /* variable auxiliar para manipular nodo */
  /* Nodo apunta al primer elemento de la pila */
  nodo = ptope;
  /* Asignamos a pila toda la pila menos el primer elemento */
  ptope = nodo->proximo;
  /* Borrar el nodo */
  delete nodo;
}
int pila::tope()
{

  pnodo nodo;    /* variable auxiliar para manipular nodo */
  int elem;      /* variable auxiliar para retorno */

  if(!ptope) return 0; /* Si no hay nodos en la pila retornamos 0 */
  /* Nodo apunta al primer elemento de la pila */
  nodo = ptope;
  /* Guardamos el valor de la clave de retorno */
  elem = nodo->clave;
  return elem;
}

int main()
{
  pila Pila;

  Pila.Apilar(20);
  cout << "Apilar(20)" << endl;
  Pila.Apilar(10);
  cout << "Apilar(10)" << endl;
  cout << "Desapilar() = " << Pila.Desapilar() << endl;
  Pila.Apilar(40);
```

```
  cout << "Apilar(40)" << endl;
  Pila.Apilar(30);
  cout << "Apilar(30)" << endl;
  cout << "tope() = " << Pila.tope() << endl;
  Pila.Desapilar();
  cout << "Tope() = " << Pila.Tope() << endl;
  Pila.Desapilar();
  Pila.Apilar(90);
  cout << "Apilar(90)" << endl;
  cout << "tope() = " << Pila.tope() << endl;
  cout << "tope() = " << Pila.tope() << endl;

  cin.get();
  return 0;
}
```