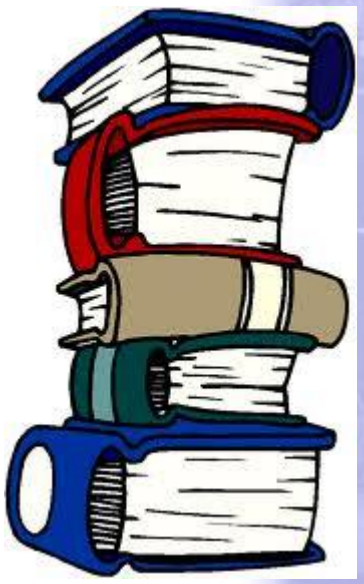
Estructuras y Tipos de Datos

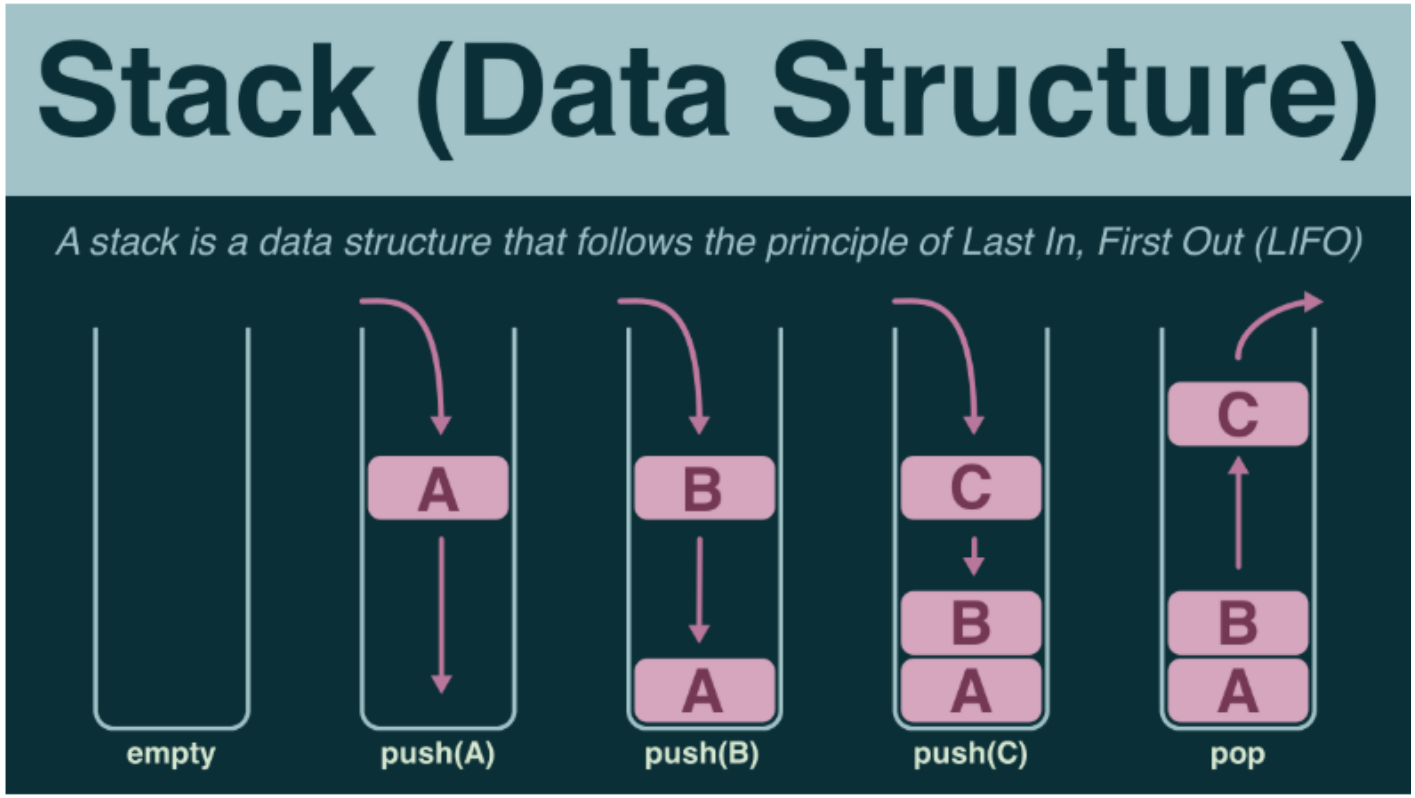
Tipo Abstracto de Datos Listas

Tipo de Dato Abstracto Pila

Una pila es un tipo especial de lista donde todas las inserciones y supresiones tienen lugar por un extremo denominado tope, una pila es una lista que se maneja bajo la política donde lo último en entrar primero en salir, denominado LIFO (Por su siglas en ingles). Una situación intuitiva de pila son por ejemplo una pila de platos o una pila de libros, situaciones donde puede ser conveniente agregar o quitar elementos por la parte superior o tope de la pila.



Tipo de Dato Abstracto Pila



Tipo de Dato Abstracto Pila

Operaciones

El TAD pila incluye por lo general, seis operaciones. Donde S es una pila y x es un elemento.

Anula(S), convierta a S en una pila vacía.

Top(S), devuelve el valor del elemento que está en el tope de la pila.

Pop(S), suprime el primer elemento de la pila S, puede ser conveniente que retorne el primer elemento.

Push(S,x), Mete o inserta el elemento x en la pila S.

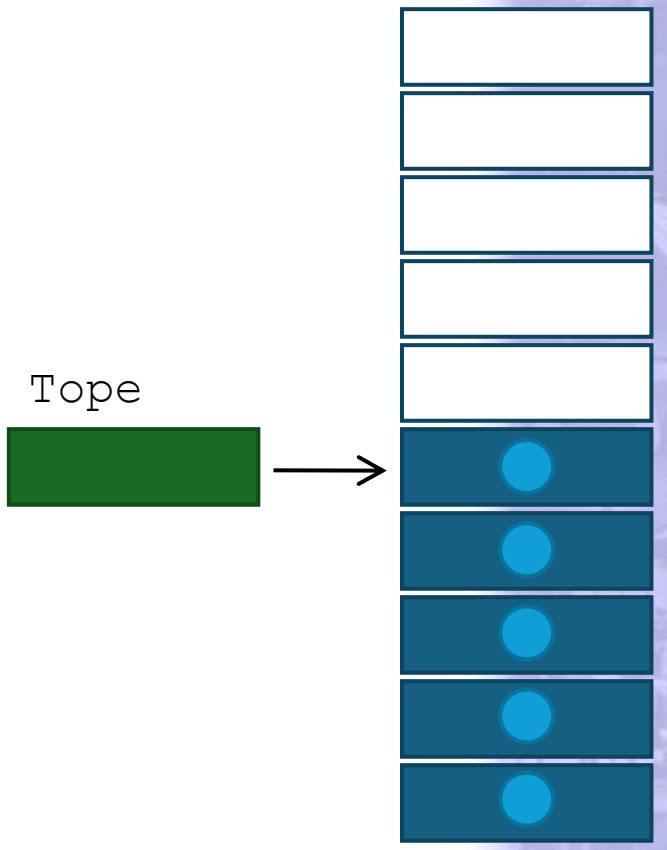
Vacía(S), es cierto si S está vacía, en caso contrario falso.

LLena(S), es cierto si S está llena, en caso contrario falso.

Tipo de Dato Abstracto Pila

Implementación de pila con arreglos

Dada la naturaleza de las pilas, donde las inserciones y supresiones ocurren por un extremo o tope, la mejor manera de implementarlas con arreglos es anclando la base de pila al inicio del arreglo y dejar que crezca hacia el otro extremo, un índice llamado tope (stack pointer) indicará la posición actual del primer elemento de la pila.



Tipo de Dato Abstracto Pila

Implementación de pila con arreglos

```
#include <stdio.h>
#include <iostream>
using namespace std;

#define MAXPILA 20

//
// Implementación de pilas con arreglos
class pila {
private:
    int elem[MAXPILA];
    int tope;
public:
    pila(){ tope = -1;};
    void vaciar(){ tope = -1;};
    bool push(int x);
    bool pop (int& x);
    bool top(int& x);
    bool vacia(){ return (tope==-1);};
    bool llena(){ return (tope==MAXPILA-1);};
};
```

Tipo de Dato Abstracto Pila

Implementación de pila con arreglos

```
//
// Inserta el elemento enviado como parámetro en la pila, falso si la pila está llena
bool pila::push(int dato){
    if (llena())
        return false;
    elem[++tope]=dato;
    return true;
}

//
// Devuelve el elemento del tope de la pila, falso si la pila está vacia
bool pila::top(int& dato){
    if(vacia())
        return false;
    dato=elem[tope];
    return true;
}

//
// Saca el elemento del tope de la pila y lo devuelve el el parámetro dato
bool pila::pop(int& dato){
    if(vacia())
        return false;
    dato=elem[tope--];
    return true;
}
```

Tipo de Dato Abstracto Pila

Implementación de pila con arreglos

```
main(){

    pila P;
    int e;

    cout << "Implementacion de Pila" << endl;

    P.push(1);
    P.push(2);
    P.push(3);
    P.push(4);
    P.push(5);

    P.top(e);
    cout << endl << "El elemento del tope es " << e << endl;

    while(P.pop(e))
        cout << e << endl;
}
```

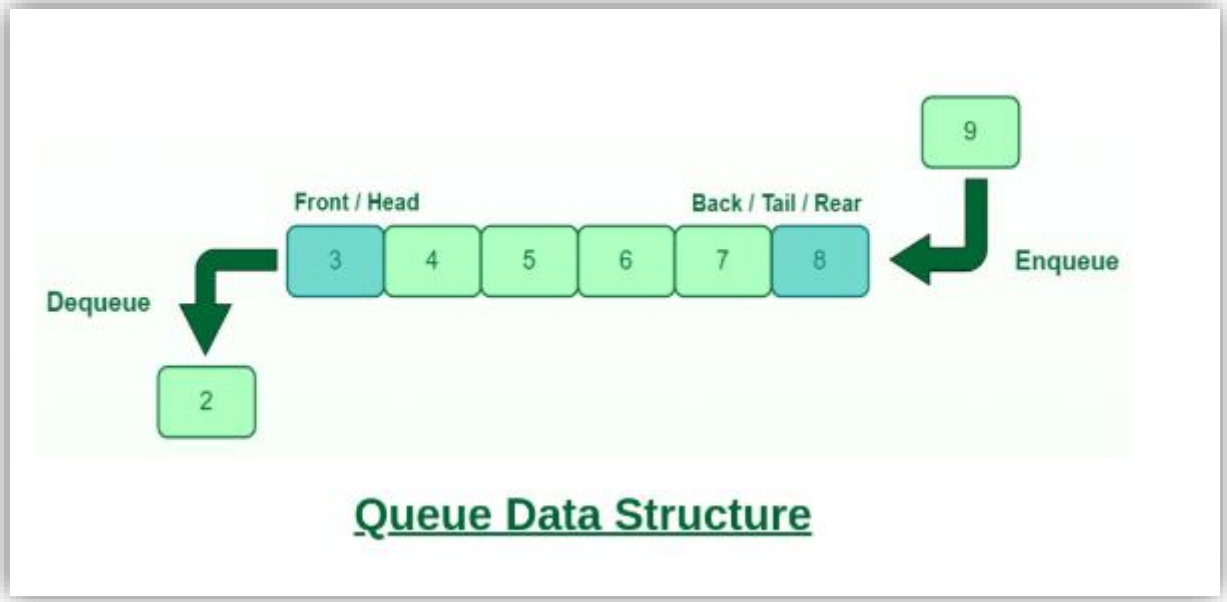
```
Implementacion de Pila
El elemento del tope es 5
5
4
3
2
1
```

Tipo de Dato Abstracto Cola

Una cola son otro tipo especial de lista, en el cual los elementos se insertan por un extremo (el posterior) y se suprimen en el otro (el anterior o frente), es decir se maneja bajo la política donde lo primero en entrar primero en salir, denominado FIFO (Por su siglas en ingles).

Las colas son algo muy común en la vida diaria y un TAD ampliamente utilizado en aplicaciones de simulación, en el planificador del sistema operativo, manejo de entrada/salida, comunicaciones etc.

Tipo de Dato Abstracto Cola



Tipo de Dato Abstracto Cola

Las operaciones ofrecidas por el TAD Lista pueden ser utilizadas para implementar el TAD cola. A continuación se muestran operaciones sobre colas, donde Q es una cola y x un elementos.

Anula(Q), convierte Q en una la cola vacía.

Top(Q), Retorna el primer elemento de Q, e decir el que lleva más tiempo en Q.

Encolar(Q,x), Añade o Inserta en una cola Q el elemento x.

Desencolar(Q), Suprime el primer elemento de Q.

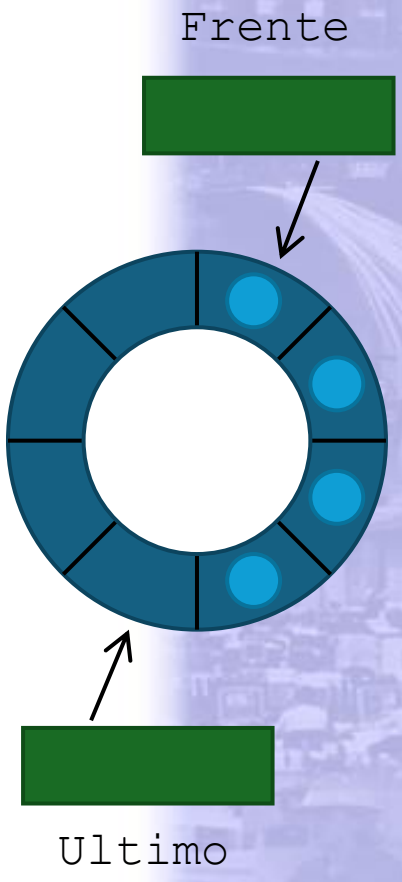
Vacía (Q), es cierta si Q está vacía, en caso contrario falso.

LLena (Q), es cierta si Q está llena, en caso contrario falso.

Tipo de Dato Abstracto Cola

Implementación de colas con arreglos circulares

La implementación de listas con arreglos mostrada anteriormente no es muy eficiente para la implementación de colas ya que implica mover todos los elementos durante la supresión lo que implica un orden de n para esta operación. Para evitar este coste se debe adoptar un enfoque diferente, es imaginarse el arreglo como un círculo donde primera posición sigue a la última y dos índice que indican el frente y la cola, de esta forma insertar y suprimir se realizan en un número constantes de pasos.



Tipo de Dato Abstracto Cola

Implementación de colas con arreglos circulares

```
#include <stdio.h>
#include <iostream>
using namespace std;

#define MAXCOLA 20

//
// Implementación de colas con arreglos circulares
class cola {
private:
    int elem[MAXCOLA];
    int longitud;    // longitud de la cola
    int frente;      // Ubicación en el arreglo circular del primer elemento
    int ultimo;      // Ubicación en el arreglo circular del último elemento

public:
    cola(){ longitud=frente=ultimo=0;};
    void vaciar(){ longitud=frente=ultimo=0;};
    bool encolar(int x);
    bool desencolar (int& x);
    bool top(int& x);
    bool vacia(){ return (longitud==0);};
    bool llena(){ return (longitud==MAXCOLA);};
};
```

Tipo de Dato Abstracto Cola

Implementación de colas con arreglos circulares

```
// Inserta el elemento enviado como parámetro en la cola, falso si la cola está llena
bool cola::encolar(int dato){
    if (llena())
        return false;
    longitud++;
    elem[ultimo] = dato;
    ultimo = (ultimo+1)%MAXCOLA;
    return true;
}
//
// Devuelve el primer elemento de la cola, falso si la cola está vacia
bool cola::desencolar(int& dato){
    if(vacia())
        return false;
    longitud--;
    dato=elem[frente];
    frente = (frente+1)%MAXCOLA;
    return true;
}
//
// Saca el elemento del tope de la pila y lo devuelve el el parámetro dato
bool cola::top(int& dato){
    if(vacia())
        return false;
    dato=elem[frente];
    return true;
}
```

Tipo de Dato Abstracto Cola

Implementación de colas con arreglos circulares

```
main(){

    cola Q;
    int e;

    cout << "Implementacion de Colas" << endl;

    Q.encolar(1);
    Q.encolar(2);
    Q.encolar(3);
    Q.encolar(4);
    Q.encolar(5);

    Q.top(e);
    cout << endl << "El elemento del tope es " << e << endl;

    while(Q.desencolar(e))
        cout << e << endl;
}
```

```
Implementacion de Colas
El elemento del tope es:1
1
2
3
4
5
```

Ejemplo Uso Tipo de Dato Abstracto Pila

Probar expresiones para paréntesis equilibrados

```
//
// Implementación de pilas con arreglos
class pila {
private:
    char elem[MAXPILA];
    int tope;
public:
    pila(){ tope = -1;};
    void vaciar(){ tope = -1;};
    bool push(char x);
    bool pop (char& x);
    bool top(char& x);
    bool vacia(){ return (tope==-1);};
    bool llena(){ return (tope==MAXPILA-1);};
};
```

```
const string OPEN = "([{";

const string CLOSE = ")]}";

bool isOpen (char ch){
    return OPEN.find(ch) != string::npos;
}

bool isClose(char ch) {
    return CLOSE.find(ch) != string::npos;
}
```

Ejemplo Uso Tipo de Dato Abstracto Pila

Probar expresiones para paréntesis equilibrados

```
bool isBalanced(const string& expresion){
    pila p;
    bool balanceado = true;

    string::const_iterator iter = expresion.begin();
    while (balanceado && (iter != expresion.end())){
        char next_ch = *iter;
        if (isOpen(next_ch)){
            p.push(next_ch);
        } else if (isClose(next_ch)){
            if (p.vacia()){
                balanceado = false;
            } else{
                char top_ch;
                p.pop(top_ch);
                balanceado = OPEN.find(top_ch) == CLOSE.find(next_ch);
            }
        }
        ++iter;
    }
    return balanceado && p.vacia();
}
```

Ejemplo Uso Tipo de Dato Abstracto Pila

Probar expresiones para paréntesis equilibrados

```
main(){

    cout << "Introducir una expresion:\n";
    string expresion;
    while (getline(cin, expresion) && (expresion != "")){
        cout << endl << expresion;
        if (isBalanced(expresion)){
            cout << " Esta Equilibarada" << endl;
        } else {
            cout << "  No esta Equilibrada" << endl;
        }
        cout << "\nIntroducir otra expresion: \n";
    }
    return 0;
}
```