Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 5

Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T3. Tipos Abstractos de Datos (TAD)

- **Concepto.**
- Tipos de datos lineales:
  - Pilas
  - Colas
  - Listas

UNIVERSIDAD NEBRIJA

ÍNDICE

Un ***Tipo Abstracto de Datos (TAD)*** es un conjunto de **datos** y de **operaciones** definidos mediante una especificación.

Los **datos** y las **operaciones** básicas definidas se estudian sin considerar la **implementación**:

- Se piensa en ***qué*** se puede hacer con los datos
- No se piensa ***cómo*** hacerlo

Los ***tipos de datos predefinidos*** tienen este mismo enfoque, indicar el **conjunto de datos** y especificar una **serie de operaciones básicas y válidas**:

int

bool

float

char

Lo mismo con los ***tipos de datos*** incluidos en la ***librería estándar (std)***:

std::string

std::vector<int>

std::array<int>

...

- **Proceso de Abstracción**

  El análisis permite obtener una especificación concreta que no puede cambiarse.

- **Proceso de Implementación**

  Siempre se puede mejorar.

```
class Persona {
private:
    string nombre;
    vector<shared_ptr<Mascota>> las_mascotas;

public:
    Persona();
    ~Persona();

    void nueva_mascota(shared_ptr<Mascota> una_mascota);

    string getNombre() const;

    vector<shared_ptr<Mascota>> nombreMascotas() const;
    vector<shared_ptr<Mascota>> pesoMascotas() const;
};
```

```
class Mascota {
private:
    string nombre;
    float peso;
    shared_ptr<Persona> propietario;

public:
    Mascota (const string &nom, shared_ptr<Persona> prop);
    ~Mascota();

    shared_ptr<Persona> get_propietario() const;

    float get_peso() const;
    void set_peso(float p);

    void mostrar() const;
};
```

```
Persona::Persona() {}

Persona::~Persona() {}

void Persona::nueva_mascota(shared_ptr<Mascota> una_mascota) {
    las_mascotas.push_back(una_mascota);
}
vector<shared_ptr<Mascota>> Persona::nombreMascotas() const {
    vector<shared_ptr<Mascota>> ret = las_mascotas;
//    quicksort(ret, 0, ret.size());
    return ret;
}
vector<shared_ptr<Mascota>> Persona::pesoMascotas() const {
    vector<std::shared_ptr<Mascota>> ret = las_mascotas;
//    bubble_sort(ret);
    return ret;
}
```

```
Mascota::Mascota(const string &nom, shared_ptr<Persona> prop) :
    nombre(nom), propietario(prop) {
    peso = -1;  // -1 cuando no se conoce
}

Mascota::~Mascota () {}

shared_ptr<Persona> Mascota::get_propietario() const {
    return propietario;
}

void Mascota::set_peso(float p) {
 peso = p;
}

float Mascota::get_peso() const {
    return peso;
}

void Mascota::mostrar() const {
    cout << "Soy una mascota y mi propietario es ";
    propietario->getNombre();
    cout << " y mi nombre es " << nombre << endl;
}
```

T3. Tipos Abstractos de Datos (TAD)

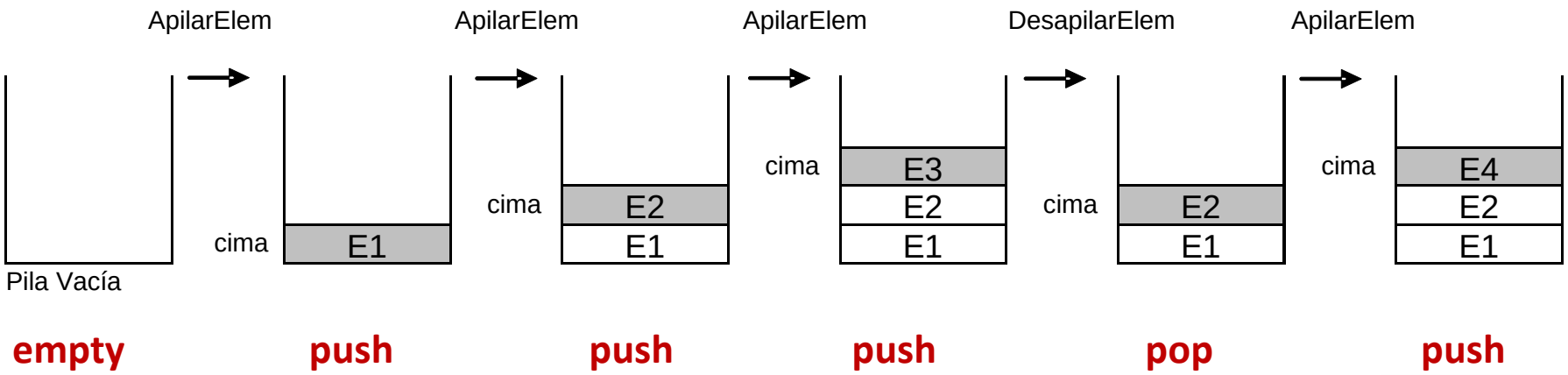
- Concepto.
- **Tipos de datos lineales:**
  - **Pilas**
  - Colas
  - Listas

UNIVERSIDAD NEBRIJA

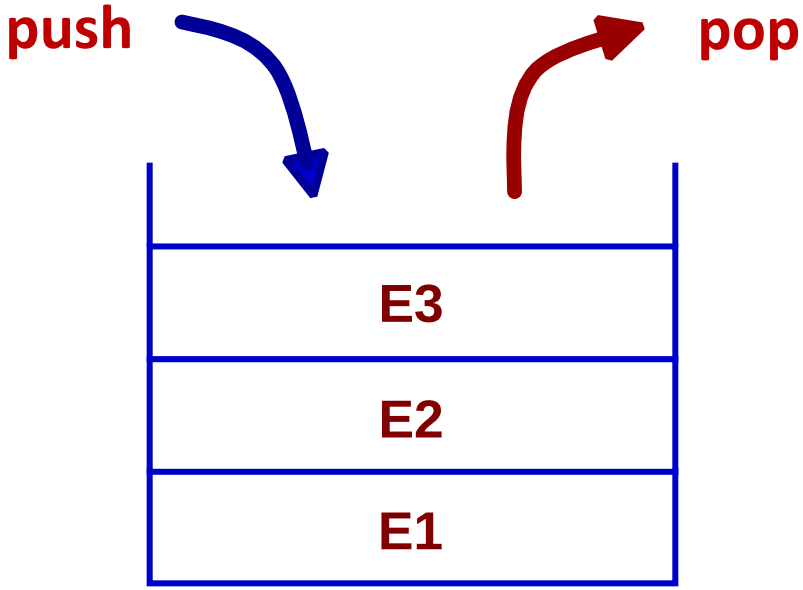
ÍNDICE

Una **PILA (STACK)** es una **estructura ordenada** y de **elementos homogéneos**, en la que podemos ***apilar*** o ***desapilar*** elementos en una **única posición** llamada ***CIMA***, y siguiendo una política ***LIFO*** (**Last In, First Out**).

Es una estructura de datos **Lineal.**



PILAS



Operaciones Básicas de Pila

empty: Determina si la pila está vacía o no.

Precondición: Ninguna.

Postcondición: Decide si la pila p tiene elementos o no.

Por tanto, la pila p no se modifica.

top: Devuelve el elemento de la cima de la pila.

Precondición: La pila p no puede estar vacía.

Postcondición: Obtiene el elemento que ocupa la cima de la pila p.

Por tanto, la pila p no se modifica.

Operaciones Básicas de Pila

push: **Inserta un elemento en la pila y se obtiene la pila con un elemento más.**

Precondición: Ninguna.

Postcondición: Almacena en la pila p el elemento e.

Por tanto, la pila p se modifica.

pop: **Elimina el elemento de la cima de la pila y devuelve la pila resultante.**

Precondición: La pila p no puede estar vacía

Postcondición: Elimina de la pila p el elemento que ocupa la cima.

Por tanto, la pila p se modifica.

```
struct TipoDato {
    string elem;
    int aa;
};

class Nodo {
    private:
        TipoDato dato;
        shared_ptr<Nodo> next;
    public:
        Nodo():next(nullptr){}
        Nodo(TipoDato const &d, shared_ptr<Nodo> const &ptr):dato(d),next(ptr){}

        TipoDato getDato() const;
        void setDato(const TipoDato &newDato);
        shared_ptr<Nodo> getNext() const;
        void setNext(const shared_ptr<Nodo> &newNext);
};

class Stack {
    public:
        Stack():front(nullptr){}

        bool empty() const;
        void push(const TipoDato &dato);
        void pop();
        TipoDato top() const;

    private:
        shared_ptr<Nodo> front;
};
```

Ejercicio: Implementar las funciones miembro de la clase Pila considerando el **TipoDato** un valor de tipo ***int***:

Stack();

bool **empty**() const;

void **push**(const TipoDato &**dato**);

void **pop**();

TipoDato **top**() const;

// Determina el tamaño de la Pila

int **size**() const;

// Vacía la Pila

void **clear**();

Escribir un programa en C++11, realizando las funciones necesarias, que permita generar y gestionar una **pila** de números enteros utilizando sólo las operaciones básicas de pila.

El programa deberá de mostrar un **menú** al usuario que le permita hacer los siguientes procesos:

1. **Introducir números enteros en la Pila:**

   Esta opción, ***introducirElem***, consistirá en solicitar un número entero al usuario y añadirlo a la Pila de números. Opcionalmente, se puede permitir añadir más valores si el usuario así lo indica.

2. **Mostrar el contenido de la Pila:**

   Esta opción, ***mostrarElemPila*****,** consistirá en mostrar por pantalla los números guardados en la Pila.

   Si la Pila está vacía se debe de mostrar un mensaje para indicárselo al usuario.

3. **Elemento mayor de la Pila:**

   Esta opción, ***elMayorElemPila***, consistirá en mostrar por pantalla un mensaje con el MAYOR de los números almacenados en la Pila.

   Para ello, realizar una función miembro de la clase Pila que determine el valor del elemento mayor de la Pila.

   Si la Pila está vacía se debe de mostrar un mensaje para indicárselo al usuario.

4. **Buscar un número en la Pila:**

   Esta opción, ***buscarElemPila***, consistirá en comprobar si un determinado valor solicitado al usuario está guardado en la Pila.

   Si la Pila está vacía se debe de mostrar un mensaje para indicárselo al usuario.

5. **Número de veces que está un número en la Pila:**

   Esta opción, ***vecesElemPila***, consistirá en determinar las veces que un determinado valor solicitado al usuario está en la Pila.

   Si la Pila está vacía se debe de mostrar un mensaje para indicárselo al usuario.

Pila