TEMA 5: TIPO DE DATOS ABSTRACTO *PILA*

ALGORITMOS Y ESTRUCTURAS DE DATOS

M. Colebrook Santamaría
J. Riera Ledesma

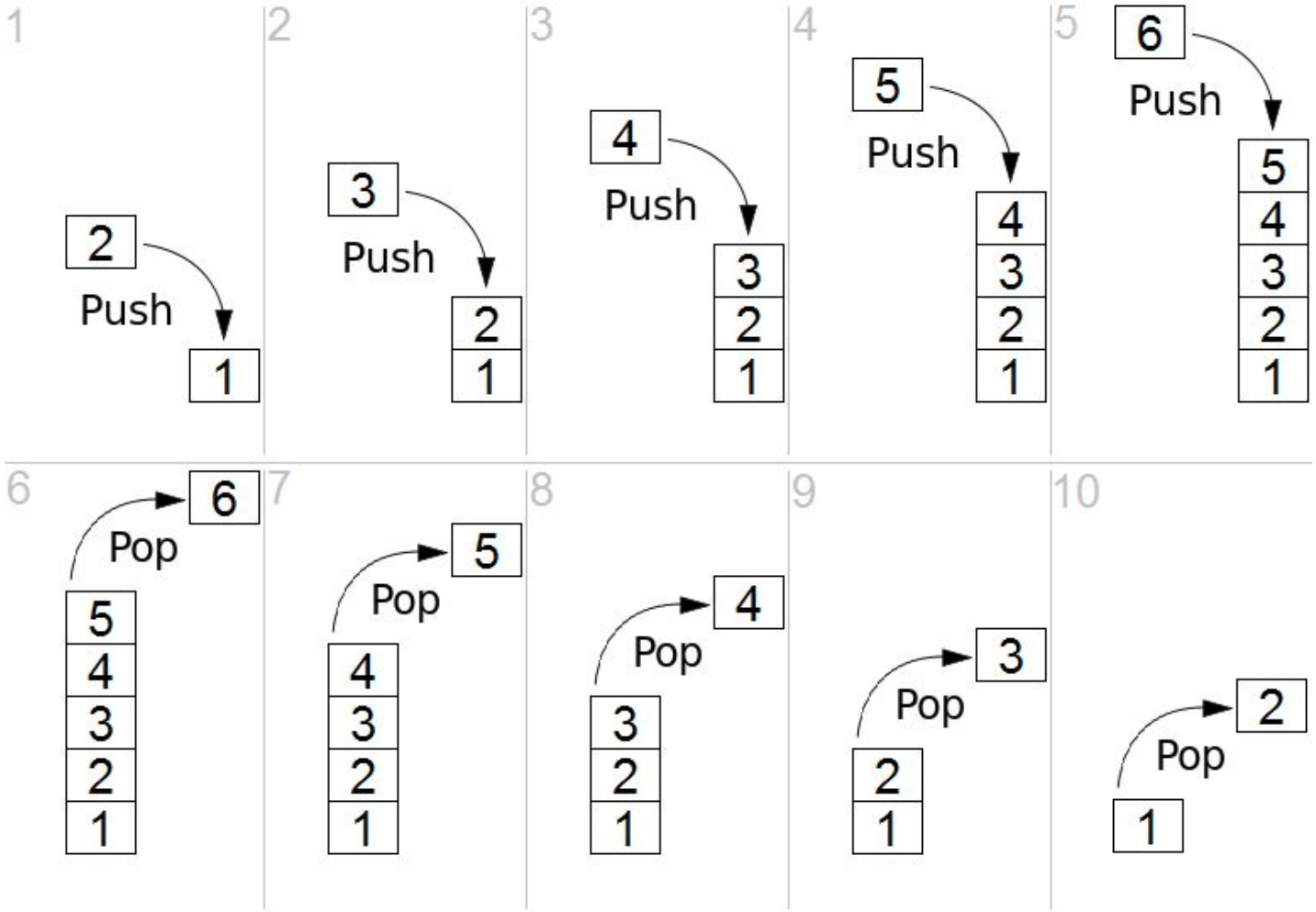
Objetivos

- Especificación formal del TDA pila.
- Implementación del TDA pila mediante estructuras estáticas y objetos dinámicos.
- Operaciones sobre pilas:
  - Inserción (*push*)
  - Extracción (*pop*)
  - Lectura del top de la pila (*top*, *peek*)

Especificación formal del TDA pila (1)

- Una **pila** (***stack*** en inglés) es una estructura de datos con un modo de acceso a sus elementos de tipo **LIFO** (*Last In First Out*: último en entrar, primero en salir).
- Como veremos posteriormente, el accesos tipo LIFO se consigue con las operaciones ***push*** y ***pop***.
- Por analogía, una operación apilar (*push*) equivaldría a colocar un plato sobre una pila de platos, y una operación retirar (*pop*) al efecto contrario.

Especificación formal del TDA pila (2)



Especificación formal del TDA pila (3)

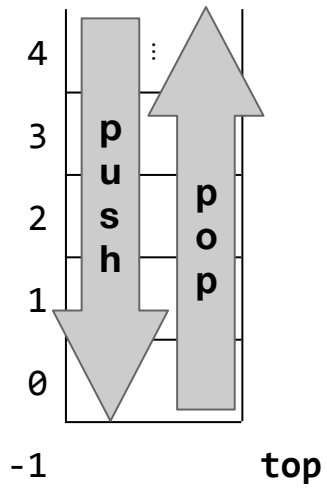
- Las pilas suelen emplearse en los siguientes contextos:
  - Evaluación de expresiones en **notación postfija** (notación polaca inversa)
  - Implementación de recursividad
  - Reconocedores sintácticos de lenguajes independientes del contexto

Implementación del TDA pila mediante estructuras estáticas (1)

- Para la versión con estructuras **estáticas** (de tamaño fijo predefinido), usaremos un objeto de tipo `vector_t<T>`:

```
template <class T>
class stack_v_t
{
private:
  vector_t<T> v_;
  int         top_;

public:
  ...
};
```



Implementación del TDA pila mediante estructuras estáticas (2)

- Los métodos iniciales son:

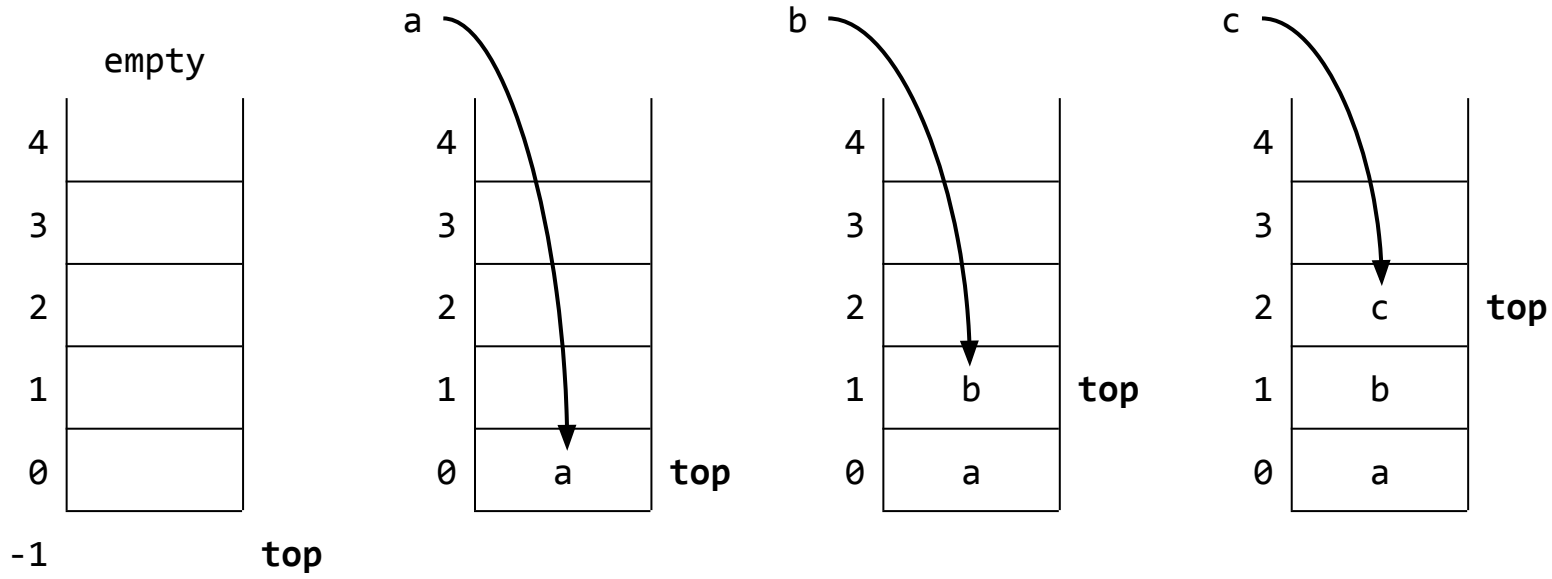
```
// constructor y destructor
stack_v_t(int max_sz): v_(max_sz), top_(-1) {}
~stack_v_t(void) {}

bool empty(void) const  // pila vacía
{
  return (top_ < 0);
}

bool full(void) const  // pila llena
{
  return (top_ == v_.get_size() - 1);
}
```

Operaciones: Inserción (*push*) (1)

- Queremos insertar los siguientes elementos en una `stack_v_t<char>` de tamaño 5: `a`, `b`, `c`, ...

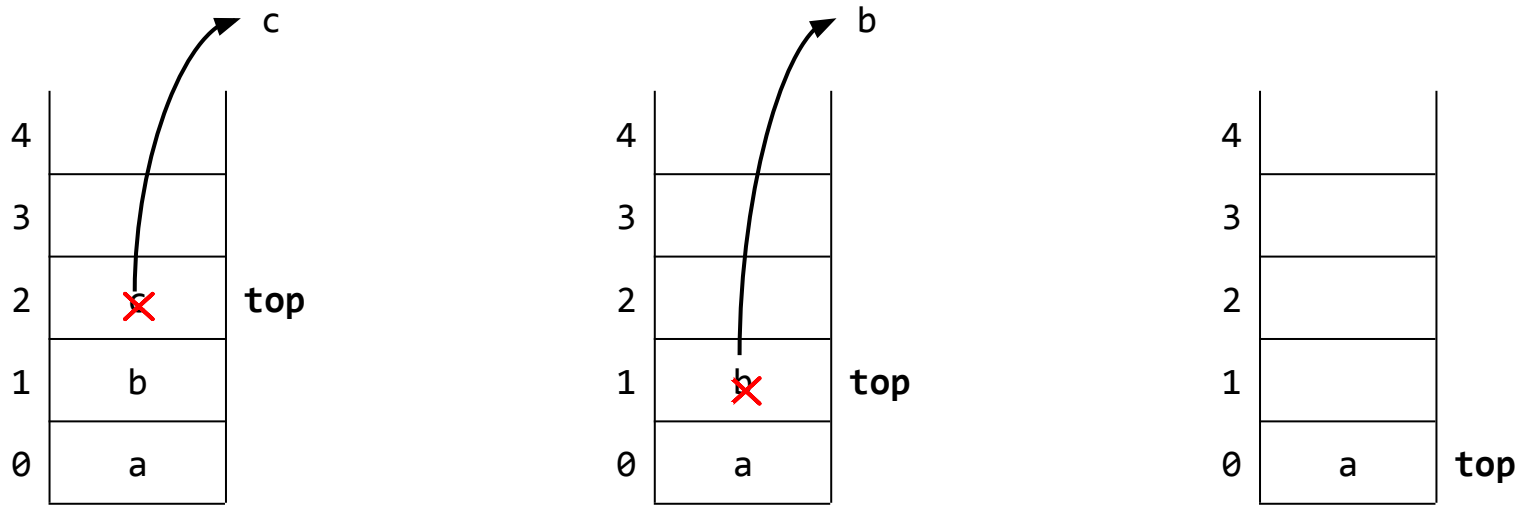


Operaciones: Inserción (*push*) (2)

```
void push(const T& dato)
{
  assert(!full());
  top_++;
  v_[top_] = dato;
}
```

Operaciones: Extracción (*pop*) (1)

- Ahora queremos extraer 2 elementos almacenados en la pila.

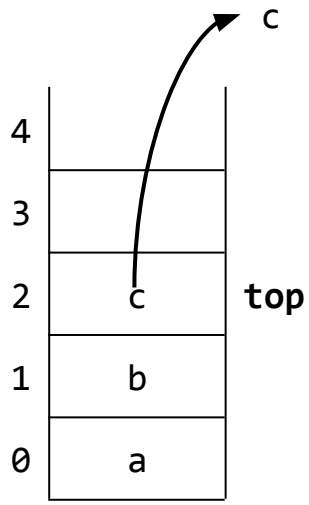


Operaciones: Extracción (*pop*) (2)

```
void pop(void)
{
  assert(!empty());
  top_--;
}
```

Operaciones: Lectura del *top* (1)

- Se puede realizar una lectura del dato que está justo en el `top` de la pila sin tener que extraerlo.



Operaciones: Lectura del *top* (2)

```
const T& top(void) const
{
  assert(!empty());
  return v_[top_];
}
```

Implementación del TDA pila mediante objetos dinámicos (1)

- En el caso de la implementación de un TAD pila con objetos dinámicos, usaremos una lista doblemente enlazada `dll_t<T>`:

```
template <class T>
class stack_l_t
{
private:
  dll_t<T>    l_;

public:
  ...
};
```

Implementación del TDA pila mediante objetos dinámicos (2)

- Los métodos iniciales serán:

```
// constructor y destructor
stack_l_t(void): l_() {}
~stack_l_t(void) {}

// pila vacía
bool empty(void)
{
  return l_.empty();
}
```

Operaciones: Inserción (*push*)

```
void push(const T& dato)
{ // creamos nodo con el dato
  dll_node_t<T>* nodo = new dll_node_t<T>(dato);

  assert(nodo != NULL);

  // lo insertamos en la lista
  l_.push_front(nodo); // l_.insert_head(nodo)
}
```

Operaciones: Extracción (*pop*)

```
void pop(void)
{
  assert(!empty());
  delete l_.pop_front(); // l_.extract_head()
}
```

Operaciones: Lectura del *top*

```
const T& top(void) const
{
  assert(!empty());
  dll_node_t<T>* node = l_.get_head();
  return node->get_data();
}
```

Referencias

- Olsson, M. (2018), “C++ 17 Quick Syntax Reference”, Apress. Disponible en PDF en la BBTK-ULL: https://doi.org/10.1007/978-1-4842-3600-0
- Stroustrup, B. (2002), “El Lenguaje de Programación C++”, Addison Wesley.
- C++ Syntax Highlighting (código en colores): tohtml.com/cpp