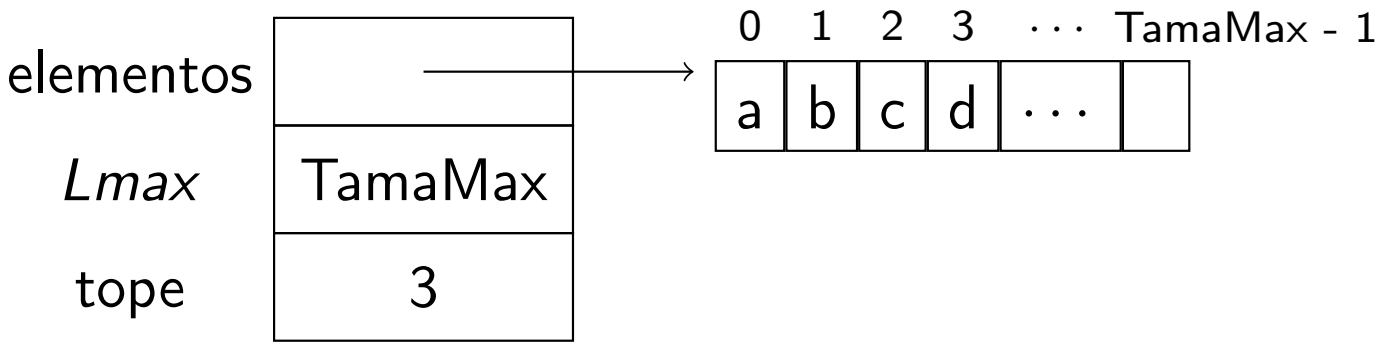
Implementación vectorial pseudoestática

Tamaño de la pila definido por el usuario del TAD en su creación.



Implementación vectorial pseudoestática

Aspectos a considerar en C++

1. Clase: Módulo que encapsula datos (atributos) y operaciones (métodos). Extiende el concepto de estructura de C.
2. En C++ `struct` y `class` son palabras reservadas sinónimas.
   - `struct`: Por defecto, miembros públicos (retrocompatibilidad con C)
   - `class`: Por defecto, miembros privados
3. En C están permitidas la copia y asignación entre estructuras del mismo tipo.
4. En C++, por preservar la compatibilidad, también se permite la copia y asignación de estructuras/clases del mismo tipo.

Implementación vectorial pseudoestática

Constructor de copia

Toda clase definida en C++ tiene un método llamado constructor de copia, que realiza una copia uno a uno de los atributos. Si este método no es válido para una clase, se debe redefinir. El constructor de copia se invoca cuando:

- se pasan parámetros por valor
- una función devuelve el resultado por valor
- se inicializa un objeto a partir de otro

Implementación vectorial pseudoestática

Operador de asignación

Toda clase definida en C++ tiene sobrecargado el operador $=$ como miembro de la clase, que asigna uno a uno los atributos. Si esta asignación no es correcta, se debe redefinir.

Destructor

Toda clase definida en C++ tiene un método llamado destructor que se invoca cuando un objeto termina su vida o, automáticamente, cuando se abandona el ámbito donde se ha definido. El destructor por defecto no hace nada especial, por lo que en muchas ocasiones es necesario definirlo.

Implementación vectorial pseudoestática

Estructuras en memoria dinámica

Supongamos una clase que tiene atributos que son punteros a estructuras en memoria dinámica:

1. La copia y la asignación atributo a atributo crean «alias» de las zonas de memoria dinámica (se copian los punteros, pero no las posiciones de memoria apuntadas). Cambiar este comportamiento requiere definir el constructor de copia y el operador de asignación para la clase en cuestión.
2. El destructor por defecto elimina un objeto, incluidos sus atributos de tipo puntero, pero no libera la memoria dinámica a la que estos apuntan. Para liberarla es necesario definir el destructor de la clase.

Implementación vectorial pseudoestática

Operador new

- Para reservar memoria dinámica se utiliza una expresión como

```
new tipo
```

que devuelve la dirección de un bloque de memoria del tamaño requerido para el `tipo`.

- Si la reserva falla, se lanza una excepción estándar `bad_alloc`.
- Si se trata de un tipo del lenguaje, se puede escribir un valor de inicialización entre paréntesis.

```
new tipo(inicializador)
```

- Si el tipo es una clase, la memoria reservada se inicializa con el constructor de la clase. La lista de parámetros, si es necesaria, se escribe a continuación entre paréntesis.

```
new clase(param1, param2,...)
```

Implementación vectorial pseudoestática

Operador new [ ]

- Una expresión como

  ```
  new tipo[n]
  ```

  reserva memoria dinámica para un vector de n elementos del `tipo` y devuelve su dirección.
- El valor n debe ser una expresión de tipo `unsigned`.
- Si el tipo es una clase, cada posición del vector se inicializa con el constructor predeterminado de la clase (el que se puede invocar sin parámetros).
- No hay posibilidad de utilizar otro constructor y si la clase no dispone del predeterminado, el uso del operador `new []` provocará un error de compilación.

Implementación vectorial pseudoestática

Operador delete

- La expresión

  ```
  delete p
  ```

  libera la memoria a la que apunta p, que debe haber sido reservada con `new`.
- Si p apunta a un objeto, previamente se llama al destructor.
- Si p es un puntero nulo, el operador `delete` no hace nada.

Implementación vectorial pseudoestática

Operador delete [ ]

- La expresión

```
delete[] p
```

  libera la memoria ocupada por el vector al que apunta p, que habrá sido reservada con el operador `new []`.
- Si el tipo del vector es una clase, primero se llama al destructor de la clase con cada objeto del vector.

Implementación vectorial pseudoestática (pilavec1.h)

```
1  #ifndef PILA_VEC1_H
2  #define PILA_VEC1_H
3  class Pila {
4  public:
5     typedef int tElemento; // por ejemplo
6     explicit Pila(unsigned TamaMax); // constructor
7     Pila(const Pila& P); // ctor. de copia
8     Pila& operator =(const Pila& P); // asignación entre pilas
9     bool vacia() const;
10    bool llena() const; // Requerida por la implementación
11    const tElemento& tope() const;
12    void pop();
13    void push(const tElemento& x);
14    ~Pila(); // destructor
15 private:
16    tElemento *elementos; // vector de elementos
17    int Lmax; // tamaño del vector
18    int tope_; // posición del tope
19 };
20 #endif // PILA_VEC1_H
```

Implementación vectorial pseudoestática (pilavec1.cpp)

```
1  #include <cassert>
2  #include "pilavec1.h"

4  // Constructor
5  Pila::Pila(unsigned TamaMax) :
6     elementos(new tElemento[TamaMax]),
7     Lmax(TamaMax),
8     tope_(-1)
9  {}

11 // Constructor de copia
12 Pila::Pila(const Pila& P) :
13    elementos(new tElemento[P.Lmax]),
14    Lmax(P.Lmax),
15    tope_(P.tope_)
16 {
17    for (int i = 0; i <= tope_; i++) // copiar el vector
18       elementos[i] = P.elementos[i];
19 }
```

Implementación vectorial pseudoestática (pilavec1.cpp)

```
21 // Asignación entre pilas
22 Pila& Pila::operator =(const Pila& P)
23 {
24    if (this != &P) { // evitar autoasignación
25       // Destruir el vector y crear uno nuevo si es necesario
26       if (Lmax != P.Lmax) {
27          delete [] elementos;
28          Lmax = P.Lmax;
29          elementos = new tElemento[Lmax];
30       }
31       // Copiar el vector
32       tope_ = P.tope_;
33       for (int i = 0; i <= tope_; i++)
34          elementos[i] = P.elementos[i];
35    }
36    return *this;
37 }
```

Implementación vectorial pseudoestática (pilavec1.cpp)

```
39 bool Pila::vacia() const
40 {
41    return (tope_ == -1);
42 }

44 bool Pila::llena() const
45 {
46    return (tope_ == Lmax - 1);
47 }

49 const Pila::tElemento& Pila::tope() const
50 {
51    assert(!vacia());
52    return elementos[tope_];
53 }
```

Implementación vectorial pseudoestática (pilavec1.cpp)

```
55 void Pila::pop()
56 {
57    assert(!vacia());
58    --tope_;
59 }

61 void Pila::push(const tElemento& x)
62 {
63    assert(!llena());
64    ++tope_;
65    elementos[tope_] = x;
66 }

68 // Destructor
69 Pila::~Pila()
70 {
71    delete[] elementos;
72 }
```