Estructuras de Datos y Algoritmos

TAD pila genérica (implementación estática)

Lección 6

Universidad Zaragoza

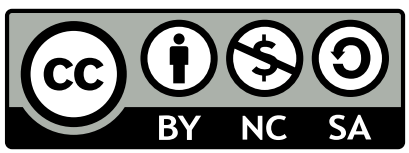
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Curso 2024/2025

Grado en Ingeniería Informática

Universidad de Zaragoza

Aula 0.04, Edificio Agustín de Betancourt



Índice

Universidad Zaragoza

Índice

Universidad Zaragoza

TAD pila

Especificación

```
espec pilasGenéricas
    usa boleanos, naturales
    parámetro formal
        género elemento
    fpf
    género pila
    {Los valores del TAD pila representan secuencias de elementos con acceso
        LIFO (last in, first out), esto es, el último elemento añadido será el
        primero en ser borrado}
    operaciones
        crear: -> pila
        {Devuelve una pila vacía, sin elementos}
        apilar: pila p, elemento e -> pila
        {Devuelve la pila resultante de añadir e a p}
        esVacía?: pila p -> booleano
        {Devuelve verdad si y sólo si p no tiene elementos}
        parcial cima: pila p -> elemento
        {Devuelve el último elemento apilado en p.
        Parcial: la operación no está definida si p es vacía}
        desapilar: pila p -> pila
        {Si p es no vacía, devuelve la pila resultante de eliminar de p el
         último elemento que fue apilado.
        En caso contrario, devuelve una pila igual a p}
        altura: pila p -> natural
        {Devuelve el número de elementos de p}
fespec
```

Índice

Universidad Zaragoza

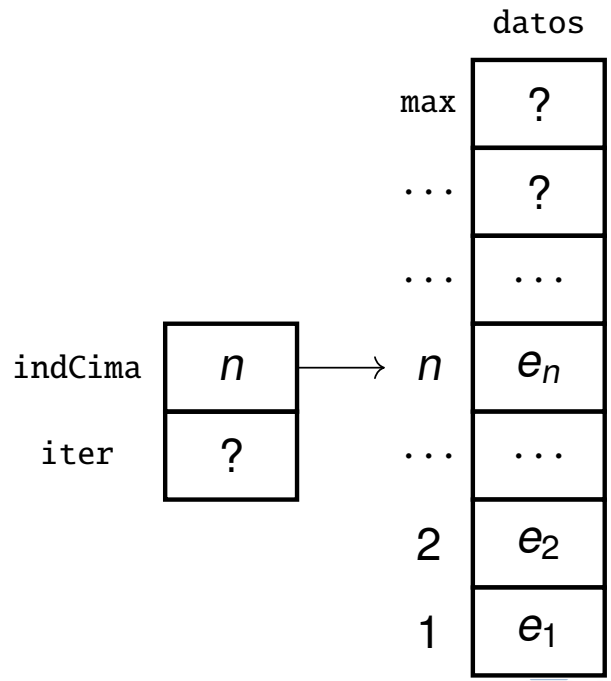
TAD pila

Implementación estática – representación de datos

Representación estática (o contigua)

- Vector con espacio para un máximo `max` de elementos
- Núm. elementos válidos en el vector: contador `indCima`, $0 \leq$ `indCima` $\leq$ `max`

```
constante max = ...
tipo pila = registro
                dato: vector[1..max] de elemento;
                indCima, iter: 0..max
            freg
```



TAD pila

Implementación estática – módulo genérico `pilas`

```
módulo genérico pilas
parámetro
    tipo elemento
exporta
    constante max = 100
    {Altura máxima de cualquier pila representable en el tipo pila, limitada por
        una decisión de implementación.}
    { Es un valor arbitrario }
    tipo pila
    {Los valores del TAD pila representan secuencias de longitud menor o igual
        que el valor de la constante 'max' de elementos con acceso LIFO (last in,
        first out), esto es, el último elemento añadido (o apilado) será el
        primero en ser borrado (o desapilado)}

    procedimiento crearVacía(sal p: pila)
    {Devuelve una pila vacía, sin elementos}

    procedimiento apilar(e/s p: pila; ent e: elemento; sal error: booleano)
    {Si altura(p)<max, entonces devuelve en el mismo parámetro p la pila
        resultante de añadir e a p, y error=falso. En caso contrario,
        devuelve error=verdad y no modifica p.}

    procedimiento desapilar(e/s p: pila)
    {Si p es no vacía, devuelve la pila resultante de eliminar de p el último
        elemento que fue apilado. Si p es vacía la deja igual}

...
```

```
...

    función cima(p: pila) devuelve elemento
    {Pre: p es no vacía} {Devuelve el último elemento apilado en p}

    función esVacía(p: pila) devuelve booleano
    {Devuelve verdad si y sólo si p no tiene elementos}

    función altura(p: pila) devuelve natural
    {Devuelve el nº de elementos de p, 0 si no tiene elementos}

    procedimiento iniciarIterador(e/s p: pila)
    {Prepara el cursor del iterador para que el siguiente elemento a visitar
        sea la cima de la pila p, si existe (situación de no haber visitado
        ningún elemento)}

    función existeSiguiente(p: pila) devuelve booleano
    {Devuelve falso si ya se han visitado todos los elementos de p.
        Devuelve verdad en caso contrario.}

    procedimiento siguiente(e/s p: pila; sal e: elemento; sal error: booleano)
    {Implementa las operaciones "siguiente" y "avanza" de la especificación,
        es decir: Si existeSiguiente(p), error toma el valor falso, e toma el
        valor del siguiente elemento de la pila, y se avanza el cursor del
        iterador al  elemento siguiente de la pila.
        Si no existeSiguiente(p), error toma el valor verdad, e queda indefinido
        y p queda como estaba.}

...
```

```
...
implementación

    tipo pila = registro
                    dato: vector[1..max] de elemento;
                    indCima, iter: 0..max
                freg

    procedimiento crearVacía(sal p: pila)
    {Devuelve una pila vacía, sin elementos}
    principio
        p.indCima := 0
    fin

    procedimiento apilar(e/s p: pila; ent e: elemento; sal error: booleano)
    {Si altura(p)<max, entonces devuelve en el mismo parámetro p la pila
        resultante de añadir e a p, y error=falso. En caso contrario,
        devuelve error=verdad y no modifica p.}
    principio
        si p.indCima < max entonces
            error := falso;
            p.indCima := p.indCima + 1;
            p.dato[p.indCima] := e
        sino
            error := verdad
        fsi
    fin
...
```

```
...
    procedimiento desapilar(e/s p: pila)
    {Si p es no vacía, devuelve la pila resultante de eliminar de p el último
        elemento que fue apilado. Si p es vacía la deja igual}
    principio
        si p.indCima > 0 entonces
            p.indCima := p.indCima - 1
        fsi
    fin

    {Versión NO robusta. Es decir, el código no chequea si se cumple la
    precondición de pila no vacía.}
    función cima(p: pila) devuelve elemento
    {Pre: p es no vacía} {Devuelve el último elemento apilado en p}
    principio
            devuelve (p.dato[p.indCima])
    fin

    {Versión robusta}
    procedimiento cima(ent p: pila; sal e: elemento; sal error: booleano)
    {Si p es vacía, error toma el valor verdad y se deja e indefinido. Si p no es
            vacía, error toma el valor falso y e toma el valor de la cima de p.}
    principio
        si esVacía(p) entonces
            error := verdad
        sino
            error := falso;
            e := p.dato[p.indCima]
        fsi
    fin
```

```
...
    función esVacía(p: pila) devuelve booleano
    {Devuelve verdad si y sólo si p no tiene elementos}
    principio
        devuelve (p.indCima = 0)
    fin

    función altura(p: pila) devuelve natural
    {Devuelve el nº de elementos de p, 0 si no tiene elementos}
    principio
        devuelve p.indCima
    fin

    procedimiento iniciarIterador(e/s p: pila)
    {Prepara el iterador para que el siguiente elemento a visitar sea la cima
        de la pila p, si existe (situación de no haber visitado ningún elemento)}
    principio
        p.iter := p.indCima
    fin
...
```

```
...
    función existeSiguiente(p: pila) devuelve booleano
    {Devuelve falso si ya se han visitado todos los elementos de p.
        Devuelve verdad en caso contrario.}
    principio
        devuelve (p.iter > 0)
    fin

    procedimiento siguiente(e/s p: pila; sal e: elemento; sal error: booleano)
    {Implementa las operaciones "siguiente" y "avanza" de la especificación,
        es decir: Si existeSiguiente(p), error toma el valor falso, e toma el
        valor del siguiente elemento de la pila, y se avanza el iterador al
        elemento siguiente de la pila.
        Si no existeSiguiente(p), error toma el valor verdad, e queda indefinido
        y p queda como estaba.}
    principio
        si existeSiguiente(p) entonces
            error := falso;
            e := p.dato[p.iter];
            p.iter := p.iter - 1
        sino
            error := verdad
        fsi
  fin

fin {del módulo}
```

Índice

TAD pila

Consideraciones finales sobre la implementación estática

- **Coste temporal** de todas las operaciones: $\mathcal{O}(1)$
  - Es decir, **son independientes de la altura** de la pila

TAD pila

Consideraciones finales sobre la implementación estática

- **Coste temporal** de todas las operaciones: $\mathcal{O}(1)$
  - Es decir, **son independientes de la altura** de la pila

Limitaciones

- **Reserva previa de espacio para el máximo previsto de elementos**
  - Desperdicio de memoria si la tasa de ocupación es baja
  - Máximo de tamaño, impuesto en la implementación
- **Implementación de `apilar` como operación parcial**

Limitaciones asumibles. *Como en los tipos predefinidos (e.g., enteros, reales), toda representación impone restricciones de tamaño o redondeo u operacionales que no están presentes en la especificación del tipo*

- *Representación de un conjunto de valores de cardinal ∞ en un espacio finito*

Índice

Universidad Zaragoza

TAD pila

Otras operaciones necesarias

Toda implementación de un contenedor o colección de datos, debe incluir operaciones para:

- **Duplicar** la representación de un valor del TAD
- **Comparación de igualdad** entre dos valores del TAD

Universidad Zaragoza

```
procedimiento duplicar(ent pEnt: pila; sal pSal: pila)
{Hace una copia en pSal de la pila almacenada en pEnt.}
variable i: natural
principio
    pSal.indCima := pEnt.indCima;
    si not esVacía(pEnt) entonces
        para i:= 1 hasta pEnt.indCima hacer
            pSal.dato[i] := pEnt.dato[i]  { *¡ojo!* ver siguiente diapositiva}
        fpara
    fsi
fin

función iguales(p1, p2 : pila) devuelve booleano
{Devuelve verdad si y sólo si p1 y p2 almacenan la misma pila.}
variables igual: booleano; i: natural
principio
    si esVacía(p1) and esVacía(p2) entonces
        devuelve verdad
    sino_si altura(p1) ≠ altura(p2) entonces
        devuelve falso
    sino {ambas pilas tienen el mismo número, no nulo, de elementos}
        igual := verdad;
        i := p1.indCima;
        mientrasQue igual and i > 0 hacer
            igual := (p1.dato[i] = p2.dato[i]); { *¡ojo!* ver siguiente diapositiva}
            i := i - 1
        fmq
        devuelve igual
    fsi
fin
```

TAD pila

Otras operaciones necesarias

¡OJO!

- `pSal.dato[i] := pEnt.dato[i]` **asume** que se puede usar la operación asignación con el tipo `elemento`
- `p1.dato[i] = p2.dato[i]` **asume** que el tipo `elemento` se puede comparar

Se pueden exigir en la especificación:

```
módulo genérico pilas
parámetros
    tipo elemento
    con procedimiento duplicar(ent eEnt: elemento; sal eSal: elemento)
            {procedimiento que duplica la representación de eEnt en eSal}
    con función iguales(e1, e2: elemento) devuelve booleano
            {función que devuelve verdad si y sólo si e1 y e2 almacenan
                el mismo elemento}
exporta
...
```

Universidad Zaragoza

Trabajo encargado

Leer lección 6 de los apuntes.

Empezar Práctica 2

Estructuras de Datos y Algoritmos

TAD pila genérica (implementación estática)

Lección 6

Universidad Zaragoza

1542

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Curso 2024/2025

Grado en Ingeniería Informática

Universidad de Zaragoza

Aula 0.04, Edificio Agustín de Betancourt