UNIVERSIDAD FRANCISCO DE VITORIA
VINCE IN BONO MALUM

Estructuras de Datos y Algoritmos

Tema 4.1. Tipos de datos lineales. Pilas

Prof. Dr. P. Javier Herrera

Contenido

- Pilas: Conceptos generales
- Operaciones básicas
- Especificación algebraica
- Implementación estática
- Repaso punteros
- Implementación dinámica
- Ejercicio: Evaluación de expresiones en forma postfija

Pilas: Conceptos generales

- Estructura de datos lineal cuya característica principal es que el acceso a los elementos se realiza en orden inverso al de su almacenamiento, siguiendo el criterio de *el último en entrar es el primero en salir*.

- Se las denomina estructuras LIFO (*Last In, First Out*) o pilas. El comportamiento de las pilas es completamente independiente del tipo de los datos almacenados en ellas, por lo que se trata de un tipo de datos parametrizado.

- La ventaja de las pilas es que el acceso a la estructura, tanto para su modificación (inserción y borrado) como para la consulta de los datos almacenados, se realiza en un único punto (la *cima* de la pila), lo que facilita implementaciones sencillas y eficientes.

- A pesar de su sencillez, se trata de una estructura con múltiples aplicaciones en el diseño de algoritmos, como la evaluación de expresiones o la implementación de la recursión.

Operaciones básicas

- El TAD de las pilas cuenta con las siguientes operaciones:
  - crear la pila vacía,
  - apilar un elemento,
  - desapilar el elemento en la cima,
  - consultar el elemento en la cima, y
  - determinar si la pila es vacía.

Especificación algebraica

especificación *PILAS*[*ELEM*]
usa *BOOLEANOS*
tipos *pila*
operaciones

pila-vacía	:		$\rightarrow$ *pila*	{ constructora }
apilar	:	*elemento pila*	$\rightarrow$ *pila*	{ constructora }
desapilar	:	*pila*	$\rightarrow_p$ *pila*	
cima	:	*pila*	$\rightarrow_p$ *elemento*	
es-pila-vacía?	:	*pila*	$\rightarrow$ *bool*	

- Como el orden de apilación es fundamental para la posterior consulta y eliminación, las constructoras son **libres** (no son necesarias ecuaciones de equivalencia).

Especificación algebraica

variables

e : elemento

p : pila

ecuaciones

desapilar(pila-vacía) = error

desapilar(apilar(*e*, *p*)) = *p*

cima(pila-vacía) = error

cima(apilar(*e*, *p*)) = *e*

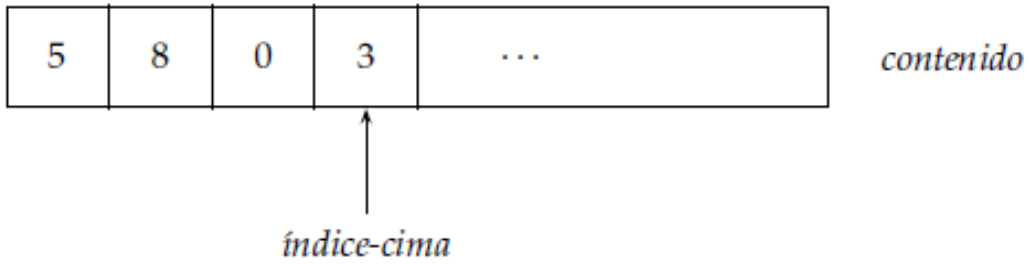
es-pila-vacía?(pila-vacía) = cierto

es-pila-vacía?(apilar(*e*, *p*)) = falso

fespecificación

Implementación estática

apilar(3, apilar(0, apilar(8, apilar(5, pila-vacía))))



tipos

pila = **reg**

contenido[1..*N*] **de** elemento

índice-cima : 0..*N*

freg

ftipos

- Almacenamos los elementos de la pila en un vector.
- Mediante un índice apuntamos a la cima.

Implementación estática

fun pila-vacía() **dev** *p* : pila $\{O(1)\}$
    p.índice-cima := 0 { Cuando la pila está vacía la *cima* vale 0}
ffun

fun es-pila-vacía?(*p* : pila) **dev** *b* : bool $\{O(1)\}$
    b := (*p.índice-cima* = 0)
ffun

proc apilar(**e** *e* : elemento, *p* : pila) $\{O(1)\}$
    si *p.índice-cima* = *N* **entonces** error(Espacio insuficiente)
    si no
        p.índice-cima := *p.índice-cima* + 1
        p.contenido[*p.índice-cima*] := *e*
    fsi
fproc

Implementación estática

```
proc desapilar(p : pila)                    {O(1)}
    si p.índice-cima = 0 entonces error(Pila vacía)
    si no
        p.índice-cima := p.índice-cima − 1          { El elemento no se borra físicamente }
    fsi
fproc

fun cima(p : pila) dev e : elemento {O(1)}
    si p.índice-cima = 0 entonces error(Pila vacía)
    si no
        e := p.contenido[p.índice-cima]
    fsi
ffun
```

Repaso punteros

- Un puntero es una variable que contiene la dirección de otra variable.
- La declaración de los punteros suele exigir la especificación de los tipos de las variables a los que puede "apuntar".
- Un puntero puede no apuntar a ninguna variable. Entonces vale NIL.
- El contenido de los punteros del mismo tipo se puede copiar.
- Existen funciones para asociar zonas de memoria a los punteros: $\text{reservar}(p)$.
- Existen funciones para liberar zonas de memoria a los punteros: $\text{liberar}(p)$.

Repaso punteros

- Ejemplo:
  - p: **puntero a** nat, declara un puntero a números naturales.
  - reservar(p): hace que el puntero p apunte a una variable de tipo natural.
  - $p\uparrow$ expresa el contenido de la dirección almacenada en p. Entonces, podemos considerar las siguientes instrucciones:
    - $p\uparrow := 5$
    - $p\uparrow = 5$
    - $p\uparrow := p\uparrow * 3$
  - Si declaramos p, q: **puntero a** nat, y hacemos:

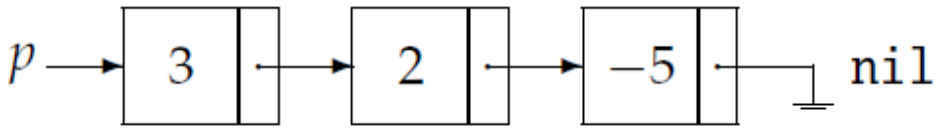
    reservar(p)

    $p\uparrow := 5$

    $q := p$

    Entonces $q\uparrow$ vale 5

Implementación dinámica



tipos

enlace-pila = **puntero a** nodo-pila
nodo-pila = **reg**
valor : elemento
sig : enlace-pila
freg
pila = enlace-pila

ftipos

- Representamos la pila como una lista de nodos.
- Cada nodo de la lista tiene dos campos: el contenido y la dirección del siguiente elemento de la lista.
- El campo *sig* del último elemento de la lista no contiene ninguna dirección (NIL).

Implementación dinámica

fun pila-vacía() **dev** *p* : pila $\{O(1)\}$
    p := nil
ffun

fun es-pila-vacía?(*p* : pila) **dev** *b* : bool $\{O(1)\}$
    b := (*p* = nil)
ffun

proc apilar(**e** *e* : elemento, *p* : pila) $\{O(1)\}$
var *q* : enlace-pila
    reservar(*q*)
    q↑.*valor* := *e* ; *q*↑.*sig* := *p*
    p := *q*
fproc

Implementación dinámica

```
proc desapilar(p : pila)                    {O(1)}
var q : enlace-pila
    si p = nil entonces error(Pila vacía)
    si no
        q := p ; p := p↑.sig ; liberar(q)
    fsi
fproc

fun cima(p : pila) dev e : elemento {O(1)}
    si p = nil entonces error(Pila vacía)
    si no e := p↑.valor
    fsi
ffun
```

Ejercicio: Evaluación de expresiones en forma postfija

- Una expresión aritmética construida con los operadores binarios $+$, -, $*$, $/$ y constantes (representados cada uno por un carácter) se dice que está en forma ***postfija*** si, o bien es una única constante, o bien consiste de dos expresiones en forma postfija una tras otra, seguidas inmediatamente por un operador.

- A continuación se presenta un ejemplo de una expresión escrita en la notación infija habitual junto con su correspondiente forma postfija:

  Forma infija: $(A/(B-C)) * (D+E)$

  Forma postfija: $ABC-/DE+*$

- Diseñar un algoritmo iterativo que calcule el valor de una expresión dada en forma postfija. Se supone que:
  - la expresión viene representada como una secuencia de caracteres,
  - hay disponibles una función valor que asocia a cada constante su valor numérico (real),
  - y una función $\mathrm{aplicar}$ que realiza la operación aritmética indicada sobre dos números reales dados.

Ejercicio: Evaluación de expresiones en forma postfija

```
proc evaluar(exp : secuencia[car], r : real )
var p : pila[real]
    p := pila-vacía()
    reiniciar(exp)
    mientras ¬fin?(exp) hacer
        x := actual(exp) ; avanzar(exp)
        si es-constante?(x) entonces apilar(valor(x), p)
        si no                { operador }
            op2 := cima(p) ; desapilar(p)
            op1 := cima(p) ; desapilar(p)
            r := aplicar(x, op1, op2) ; apilar(r, p)
        fsi
    fmientras
    r := cima(p) ; desapilar(p)
fproc
```

Bibliografía

- Martí, N., Ortega, Y., Verdejo, J.A. *Estructuras de datos y métodos algorítmicos*. Ejercicios resueltos. Pearson/Prentice Hall, 2003. Capítulo 3

- Peña, R.; *Diseño de programas. Formalismo y abstracción*. Tercera edición. Prentice Hall, 2005. Capítulo 6

(Estas transparencias se han realizado a partir de aquéllas desarrolladas por los profesores Clara Segura, Alberto Verdejo y Yolanda García de la UCM, y la bibliografía anterior)