

TEMA 4

LISTAS

Hasta ahora, hemos hablado de tipos que prevén la declaración de variables localizadas estaticamente. Una variable estática es la que se declara en un programa, y consecuentemente, localizada por su identificador. Se llama estática porque existe (hay memoria asignada para ella) durante toda la ejecución del bloque (programa, procedimiento o función) a la cual es local. De otro lado, una variable puede ser creada y destruida dinamicamente durante la ejecución de un bloque (sin ninguna relación con la estructura estática del programa). Consecuentemente, tal variable se llama variable dinámica o una variable identificada.

Kahtleen Jensen, Niklaus Wirth

Pascal. Manual user and report

OBJETIVOS DE ESTE CAPITULO:

Obtener conocimiento del segundo tipo importante de Estructuras de Datos: estructuras Dinámicas

Entender los tipos Recursivos de Datos con el caso más sencillo

¿Cuándo utilizar una estructura de datos recursivas?

INDICE TEMA-4

Listas. 9 horas.

Concepto

Clasificación

Listas ordinales: pilas – colas

Listas calificadas

Estructuras relacionadas

1. Concepto

La Lista Lineal es una estructura de datos:

Homogénea: todos sus componentes son del mismo tipo.

Secuencial: cada elemento va seguido de otro del mismo tipo o de ninguno.

Ordenada: sus componentes se almacenan según cierto orden.

Representación 

colocando sus elementos entre paréntesis, separados por comas:

(e1, e2, ..., en) 

lista distinta a la (en, e1, ..., e2)

Una lista puede implementarse con arrays, ficheros secuenciales o punteros.

2. Clasificación:

Según el modo de acceso al elemento siguiente:

Listas Densas: los elementos siguen una secuencia física; para llegar a un elemento, hay que pasar por todos los anteriores a él.

Listas Enlazadas: cada elemento contiene la información necesaria para llegar al siguiente.

Por la información utilizada para acceder a sus elementos:

Ordinales 

los elementos se colocan en la lista por orden de llegada y se accede a ellos por su posición (p. ej., una pila).

Calificadas 

los elementos se clasifican por una clave; pueden estar ordenados o no estarlo.

3. Listas ordinales: pilas – colas

Pila: lista ordinal con modo de acceso LIFO.

Implementación del TAD pila mediante listas

Especificación del TAD:

Type Pila = ^Componente;

Componente = Record

elemento: TipoElemento;

siguiente: Pila

end;

Operaciones del TAD:

- Crear una pila: inicializarla
- Cima: consulta la cabeza de la pila
- Apilar: introduce un nuevo elemento en la pila
- Desapilar: extrae el elemento en la cabeza de la pila
- Vacía: averigua si una pila no tiene nada en su interior

Implementación del TAD:

Nombre	Par. E.	Par. S.	Efecto	Restricciones
CrearPila	Pila	Pila	Se dispone de una estructura tipo pila accesible y vacía	Ninguna
Cima	Pila	Elemento	No modifica la estructura	Pila vacía
Apilar	Pila, Elemento	Pila	Aumenta en 1 el tamaño de la pila	Pila llena
Desapilar	Pila	Pila, Elemento	Disminuye en 1 el tamaño de la pila	Pila vacía
Vacía	Pila	Boolean	Ninguno	Ninguna

Cabeceras de las operaciones:

Procedure CrearPila(var Pila: Tpila);

Function Cima(Pila: Tpila): Elemento;

Procedure Apilar(var Pila: Tpila; E: Elemento);

Procedure Desapilar(var Pila: Tpila; var E: Elemento);

Function Vacía(Pila: Tpila): Boolean;

{ ***** }

{ }

{ Unidad 'U_Pila.pas': Especificación e implementación }

{ del TAD y operaciones para una pila con punteros }

{ }

{ Turbo Pascal 6.0 }

{ 10-Abr-97 }

{ }

{ ***** }

unit U_Pila;

interface

Type TipoElemento = **Integer**;

TPila = ^Componente;

Componente = **Record**

elemento: TipoElemento;

siguiente: TPila

end;

{ Procedimientos que se exportan }

Procedure CrearPila(**var** P: TPila);

Function Vacía(P: TPila): **Boolean**;

Function Cima(P: TPila): TipoElemento;

Procedure Apilar(**var** Pila: Tpila; E: Elemento);

Procedure Desapilar(**var** Pila: Tpila; **var** E: Elemento);

Implementation

uses crt;

Procedure CrearPila (**var** P: TPila);

{-----}

{ Inicializa una pila, recibida como parámetro }

{ con el valor 'nil' }

{-----}

begin

P := **nil**;

end;

Function Vacía (P: TPila): **Boolean**;

{-----}

{ Comprueba si la pila recibida como parámetro, }

{ está vacía, devolviendo un valor true en este }

{ caso, false en caso contrario }

{-----}

begin

Vacía := P = **nil**

end;

Function Cima (P: TPila):TipoElemento;

{-----}

{ Devuelve el primer elemento de la pila }

{ recibida como parámetro }

{-----}

begin

if Vacía(P)

then writeln ('Pila vacía') { << Acciones a tomar >> }

else Cima := P^.elemento

end;

Procedure Apilar(var P: Tpila; E: Elemento);

{-----}

{ Introduce un elemento dado en una pila }

{-----}

var aux: TPila;

begin

new (aux);

aux^.siguiente := P;

aux^.elemento := E;

P := aux

end;

Procedure Desapilar(var Pila: Tpila; var E: Elemento);

{-----}

{ Extrae y devuelve un elemento de una pila }

{ (por definición, el último que se introdujo), }

```

{ detectando previamente si la pila está vacía }

{-----}

var aux: Pila;

begin

if Vacía( P )

then writeln ( 'Pila vacía; no puede extraerse ningún elemento')

{ << Acciones a tomar >> }

else

begin

E := P^.elemento;

aux := P;

P := P^.siguiente;

dispose ( aux )

end

end;

```

¿Como se implementaría

el procedimiento

`ImprimirPila' de este programa?

Programa principal de prueba:

uses U_pila;

var pil: Pila;

i: integer;

begin

CrearPila (Pil);

ImprimirPila (Pil);

```

for i := 1 to 10 do
  Apilar ( Pil, i );
  ImprimirPila ( Pil );
for i := 1 to 11 do
  Desapilar ( Pil );
  ImprimirPila ( Pil );
writeln;
end.

```

Cola: lista ordinal con modo de acceso **FIFO**.

Implementación del TAD cola mediante listas

Especificación del TAD:

Type Puntero = ^Componente;

Componente = **Record**

elemento: TipoElemento;

siguiente: Puntero

end;

Cola = **Record**

principio, final: Puntero

end;

Operaciones del TAD:

- Crear una cola: inicializarla
- Primero: devuelve el elemento que ocupe la primera posición
- Introducir: introduce un nuevo elemento en la cola
- Borrar: elimina el elemento en la cabeza de la cola
- Vacía: averigua si una cola no tiene nada en su interior

Implementación del TAD:

Nombre	Par. E.	Par. S.	Efecto	Restricciones
CrearCola	Cola	Cola		Ninguna

			Se dispone de una estructura tipo Cola accesible y vacía	
Primero	Cola	Elemento	No modifica la estructura	Cola vacía
Encolar	Cola, Elemento	Cola	Aumenta en 1 el tamaño de la Cola	Cola llena
Desencolar	Cola	Elemento, Cola	Disminuye en 1 el tamaño de la Cola	Cola vacía
Vacía	Cola	Boolean	Ninguno	Ninguna

Cabeceras de las operaciones:

Procedure CrearCola (var C: Cola);

Function Primero (C: Cola): Elemento;

Procedure Apilar (var C: Cola; E: TipoElemento);

Procedure Desapilar (var C: Cola);

Function Vacía (C: Cola): Boolean;

{ ***** }

{ }

{ Unidad 'U_Cola.pas': Especificación e implementación }

{ del TAD y operaciones para una Cola con punteros }

{ }

{ Turbo Pascal 6.0 }

{ Abr-97 }

{ }

{ ***** }

unit U_Cola;

interface

Type Puntero = ^Componente;

Componente = **Record**

elemento: TipoElemento;

siguiente: Puntero

end;

TCola = **Record**

principio, final: Puntero

end;

{ Procedimientos que se exportan }

Procedure CrearCola (**var** C: TCola);

Function Vacía (C: TCola): **Boolean**;

Function Primero (C: TCola): TipoElemento;

Procedure Apilar (**var** C: TCola; E: TipoElemento);

Procedure Desapilar (**var** C: TCola);

Procedure ImprimirCola (C: TCola);

Implementation

uses crt;

Procedure CrearCola (**var** C: TCola);

{-----}

{ Inicializa una cola, recibida como parámetro }

{ con los valores: }

{ principio := nil; final := nil; }

{-----}

begin

C.principio := **nil**;

C.final := **nil**

end;

Function Vacía (C: TCola): **Boolean**;

{-----}

{ Decide si la cola C tiene algún elemento en su }

```

{ interior }

{-----}

begin

Vacía := C.principio = nil

end;

Procedure Primero ( C: TCola; var E: TipoElemento );

{-----}

{ Devuelve en E el primer elemento de la cola }

{-----}

begin

if Vacía( C )

then writeln ( 'Cola vacía' ) { << Acciones oportunas >> }

else E := C.principio^.elemento

end;

Procedure Apilar ( var C: TCola; E: TipoElemento );

{-----}

{ Almacena el elemento E en la cola C, después }

{ del último }

{-----}

var nuevo: Puntero;

begin

new ( nuevo );

nuevo^.elemento := E;

nuevo^.siguiente := nil;

with C do

begin

```

if vacía (C)

then principio := nuevo

else final^.siguiente := nuevo;

final := nuevo

end

end;

Procedure Desapilar (**var** C: TCola);

{-----}

{ Elimina el primer elemento de la cola C }

{-----}

var aux: Puntero;

begin

if Vacía(C)

then writeln ('Cola vacía') { << Acciones oportunas >> }

else with C do

begin

aux := principio;

principio := principio^.siguiente;

dispose (aux);

if principio = **nil** **then** final = **nil**

end

end;

¿Cómo se efectuaría el procedimiento:

Procedure ImprimirCola (C: TCola); ?

Imprime el contenido de la cola C

4. Listas calificadas

Los elementos de la lista están dispuestos de acuerdo con una relación de orden sobre el valor de una clave, normalmente esta relación es de mayor a menor.

Operaciones típicas sobre la lista: creación, búsqueda, inserción, borrado.

Inserción de un elemento en lista calificada en el lugar que corresponda

Borrado de un elemento en lista calificada requiere buscar primero el elemento a borrar

```
{ ***** }
```

```
{ }
```

```
{ Unidad 'Lista_Ordenada.pas': Especificación e }
```

```
{ implementación del TAD y operaciones con punteros }
```

```
{ }
```

```
{ Turbo Pascal 6.0 }
```

```
{ Abr-97 }
```

```
{ }
```

```
{ ***** }
```

```
unit Lista_Ordenada ;
```

```
interface
```

```
Type TLista = ^Nodo;
```

```
Nodo = Record
```

```
Clave: TClave;
```

```
Siguiente: Tlista
```

```
end;
```

```
{ Procedimientos que se exportan }
```

```
Function Busqueda ( Lista: TLista; Clave: TClave ): Tlista;
```

```
Procedure Insercion ( var Lista: TLista; Clave: TClave );
```

```
Procedure Borrado ( var Lista: TLista; Clave: TClave );
```

Implementation

uses crt;

Function Busqueda (Lista: TLista; Clave: TClave): Tlista;

{-----}

{ Búsqueda en lista ordenada. }

{ Se supone que no existen claves repetidas. }

{ }

{ Si no se encuentra la clave, la función devuelve }

{ **nil.** }

{-----}

var MayorOigual: **Boolean**;

begin

MayorOigual := **false**;

while (**not** encontrado) **and** (Lista <> **nil**) **do**

if Lista^.Clave >= Clave **then**

MayorOigual := **true**

else Lista := Lista^.Siguiente;

if MayorOigual **then**

if Lista^.Clave 0 Clave **then**

Busqueda := Lista

else Busqueda := **nil**;

end;

Procedure Insercion (**var** Lista: TLista; Clave: TClave);

{-----}

{ Inserción en lista ordenada }

{ }

{ Nunca se insertan claves repetidas }

{-----}

var Anterior, Actual: TLista;

MayorOigual: **Boolean**;

Procedure Insertar;

var Nuevo: TLista;

begin

new (Nuevo);

Nuevo^.clave := Clave;

Nuevo^.Siguiente := Actual;

if Actual = Lista **then** Lista:= Nuevo

else Anterior^.Siguiente := Nuevo

end;

begin (* Inserción *)

MayorOigual := **false**;

Actual := Lista;

while (**not** encontrado) **and** (Actual <> **nil**) **do**

if Actual^.Clave >= Clave **then**

MayorOigual := **true**

else begin

Anterior := Actual;

Actual := Actual^.Siguiente

end;

if Actual = **nil** **then** Insertar

else if MayorOigual **then**

if Actual^.Clave > Clave **then** Insertar

end; (* Inserción *)

Procedure Borrado (**var** Lista: TLista; Clave: TClave);

{-----}

{ Borrado en lista ordenada }

{-----}

var Anterior, Actual: Tlista;

MayorOigual: **Boolean**;

begin

MayorOigual := **false**;

Anterior := Lista;

Actual := Lista;

while (**not** encontrado) **and** (Actual <> **nil**) **do**

if Actual^.Clave >= Clave

then MayorOigual := **true**

else begin

Anterior := Actual;

Actual := Actual^.Siguiente

end;

if MayorOigual **then**

if Actual^.Clave = Clave **then**

begin

if Actual = **nil** (* hay que borrar el primer elemento *)

then Lista := Lista^.Siguiente

else Anterior^.Siguiente := Actual^.Siguiente;

dispose (Actual);

end

end;

¿Cuál sería la formulación Recursiva de las operaciones de Búsqueda, Inserción y Borrado?

¿Cuál sería la formulación Recursiva de las operaciones de Búsqueda, Inserción y Borrado?

Function Busqueda (Lista: TLista; Clave: Tclave): TLista;

{-----}

{ Búsqueda recursiva en lista ordenada. }

{ }

{ Devuelve un puntero al elemento que contiene }

{ la clave `Clave' }

{-----}

var nuevo: Puntero;

begin

if Lista = **nil** (* La clave no existe *)

then Busqueda = **nil**

else

if Lista^.Clave < Clave

then Busqueda := Busqueda (Lista^.Siguiente, Clave)

else if Lista.Clave = Clave

then Busqueda := Lista

else Busqueda := **nil**

end;

Procedure Insercion (**var** Lista: TLista; Clave: TClave);

{-----}

{ Inserción recursiva en lista ordenada }

{ }

{ Nunca se insertan claves repetidas }

{-----}

Procedure Insertar;

var Nuevo: TLista;

begin

new (Nuevo);

Nuevo^.Clave := Clave;

Nuevo^.Siguiente := Lista;

Lista:= Nuevo

end; (* Insertar *)

begin (* Inserción *)

if Lista = **nil** **then** Insertar

else if Lista^.Clave < Clave

then Insercion (Lista ^.Siguiente, Clave);

else if Lista^.Clave > Clave

then Insertar

end; (* Inserción *)

La unidad que sigue se he programado en Turbo Pascal, v. 7.0

La unidad que sigue se he programado en Turbo Pascal, v. 7.0

Cap. 4 ED-I. LISTAS

Jesús Alonso S. Dpt. OEI p-15-

ESCUELA UNIVERSITARIA DE INFORMÁTICA ESTRUCTURAS DE DATOS-I

Jesús Alonso S. Dpt. OEI