Punteros. Estructuras dinámicas

Contenido:

Memoria Dinámica y Punteros

Estructuras Dinámicas

- Una estructura de datos se dice *dinámica* si su tamaño cambia en tiempo de ejecución del programa.

El tipo puntero

- Un *puntero* es una variable que apunta o referencia a una *ubicación de memoria* en la cual hay datos
- El contenido del puntero es la *dirección* de esa ubicación
- A través del puntero se puede:
  1. "crear" la ubicación de memoria (*new o crear*)
  2. acceder a los datos en dicha ubicación (↑ desreferenciación)
  3. "destruir" la ubicación de memoria (liberar)

Declaración de una variable puntero

Ejemplos de declaraciones

```
tipo
    ptrint   = pointer_to_integer; (* puntero a entero *)
    ptrbool  = pointer_to_boolean; (* puntero a boolean *)

    nodo = registro
              . . .
              . . .
           finreg

    ptrcelda = pointer_to_nodo; (* puntero a un registro *)
```

Sintaxis de la declaración

En general:

- **`tipo`** *identificador* = pointer_to_*identificador_de_tipo*

donde:

- *identificador* es el nombre del tipo puntero que se define
- *identificador_de_tipo* corresponde a un tipo predefinido o definido previo a la declaración.
- el tipo asociado con *identificador_de_tipo* puede ser cualquiera de los tipos definidos

Valores y operaciones con punteros

- valores iníciales son indefinidos (como toda variable)
- el valor guardado en una variable puntero es una dirección de memoria, por lo tanto dependiente de la arquitectura del computador
- No es posible leer o escribir valores de tipo puntero (*read o write*)
- La única comparación permitida es la igualdad (*operador* ==)

El procedimiento new o crear

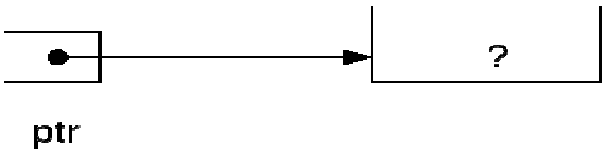
Se invoca de la siguiente manera:

```
new(ptr) o crear(ptr)
```

Donde `ptr` es una variable de tipo pointer_to_`T`.

El efecto de esta operación es:

1. Se crea un espacio nuevo para una variable de tipo T (tanta memoria como requiera el tipo)
2. La dirección de este espacio se guarda en la variable ptr



El proceso ejecutado por new se denomina **asignación dinámica de memoria**

Observar que:

- estas acciones se realizan en tiempo de **ejecución**
- la variable creada de tipo T queda con un valor inicial indefinido
- la variable creada no tiene nombre (identificador)

El operador ↑

Supongamos que se ejecuta:

```
new(ptr1)
new(ptr2)
```

Es posible hacer uso de las variables creadas usando las expresiones:

- `ptr1↑` variable apuntada por `ptr1`
- `ptr2↑` variable apuntada por `ptr2`

Observaciones

- La expresion `ptr1↑` puede ser usada en cualquier parte donde se admite una variable del tipo correspondiente, por ejemplo:
  - parte izquierda y derecha de una asignación
  - expresiones del tipo apropiado
- La expresión `ptr↑` produce un error si `ptr` no fue inicializado.
- Una variable puntero se inicializa por new o por asignación de otro puntero ya inicializado.

Ejemplos

Sean las declaraciones

```
tipo
   ptrnum = pointer_to_integer
var
   ptrnum ptr1, ptr2
```

Las siguientes instrucciones son válidas:

```
   new(ptr1)
   new(ptr2)
   ptr1↑ ← 12
   ptr2↑ ← ptr1↑ + 4
   leer(ptr1↑)              (* suponga que se ingresa 3 *)
   escribir(ptr2↑ * ptr1↑)   (*¿qué despliega?          *)
```

Punteros como parámetros por valor

Cuando se pasa un puntero como parámetro por valor, los resultados no son los esperados:

```
 tipo
    pint = pointer_to_integer;
 var
    pint p

procedimiento ALGO_HACE(pint q1,q2) (*paso por valor*)
 inicio
     q1↑ ← q2↑ * 2
     q2↑ ← q1↑ + 2
 fproc

// Principal
 inicio
    new(p)
    p↑ ← 4
    ALGO_HACE (p,p)
    escribir(p↑)      (*¿qué valor despliega?*)
 fin
```

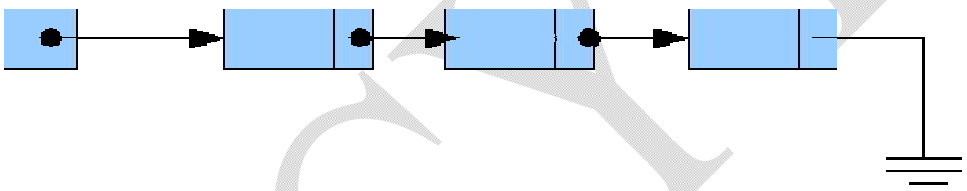
Listas

El tipo Lista

- Una **lista** es una *secuencia* o *sucesión* finita de elementos.
- Se puede representar como:
  - **arreglo:** *Tamaño fijo*, no se pueden agregar y/o quitar elementos.
  - **listas enlazadas:** utilizando *punteros*. permite agregar y/o quitar elementos. Ocupa un espacio proporcional al tamaño.

Listas enlazadas

- Cada elemento se almacena en un nodo (también se le llama celda)
- Cada nodo contiene la información del elemento y un puntero al siguiente nodo
- Para acceder a la lista basta con conocer el puntero al primero elemento
- ¿Cómo se representa una lista vacía? ¿cómo se reconoce el último elemento de la lista?



El puntero nulo

- Existe una constante especial nil que es llamada el *puntero nulo*.
- nil pertenece a todos los tipos de la forma ↑T
- nil no representa una dirección de memoria
- nil↑ da un error en tiempo de ejecución
- El valor nil se puede asignar directamente a una variable de tipo puntero:

  p ← nil

- ¿Para qué sirve?
  - Representar la lista vacía
  - distinguir el último elemento de una lista encadenada

Representación de Listas Enlazadas

```
tipo
    lista = pointer_to_nodo

    nodo = registro
              T elemento
              pointer_to_nodo siguiente
           freg
```

Longitud de una lista

Calcular la cantidad de elementos de una lista

```
funcion longitud(lista l): integer;
var
    integer contador
    lista p
inicio
    contador ← 0
    p ← l
    mientras p ≠ nil hacer
       contador ← contador + 1;
       p ←  p↑.siguiente;    (* avanzar a la siguiente celda o nodo *)
    fmientras

    retornar(contador)
fin
```

Búsqueda en una lista

Buscar un elemento en una lista

```
funcion pertenece(T elem; lista l): boolean
var
   lista p
inicio
   p ← l
   mientras (p ≠ nil) and (p↑.elemento ≠ elem) hacer
       p ← p↑.siguiente
   fmientras

   retornar(p ≠ nil)
fin
```

Agregar elemento al principio

```
procedim agregar_al_principio(ref lista l; T elem)
var lista p
inicio
    new(p);               (*crear nuevo nodo*)
    p↑.elemento ← elem    (*cargar el elemento*)

    (* ajuste de punteros *)
    p↑.siguiente ← l;
    l ←  p;
end;
```

Agregar elemento al final

```
procedim agregar_al_final(ref lista l; T elem)
var
  lista p,q
inicio
    new(p)               (*crear nuevo nodo o celda*)
    p↑.elemento ← elem   (*cargar el elemento*)
    p↑.siguiente ← nil   (*es el último*)

    si l == nil then
        l ← p
    sino
        (* busco el último de l *)
        q ←  l
        while q↑.siguiente ≠ nil do
           q ← q↑.siguiente
        fmientras

        q↑.siguiente ← p (*enlazando p a continuación del último*)
    fsi
fin
```

El procedimiento `liberar`

- indica al sistema que un nodo (celda de memoria) ya no será utilizada
- el sistema recupera el espacio y lo puede reutilizar
- se invoca así:

  `liberar(ptr)`

  donde `ptr` es un puntero al nodo en cuestión. Luego del liberar, `ptr` queda *indefinido*

- se utiliza cuando se borran elementos de una estructura dinámica

Borrar Primero de una Lista

Suponemos la lista no vacía

```
procedim borrar_primero(ref lista l)
var
    lista p
inicio
    p ← l
    l ← l↑.siguiente
    liberar(p)
fin
```

Borrar la lista entera

Para liberar todo el espacio ocupado por una lista es necesario liberar celda por celda.

```
procedim borrar_lista(lista l)
var
   lista p
inicio
   mientras l ≠ nil hacer
       p ← l
       l ← l↑.siguiente
       liberar(p);
   fmientras
fin
```