RESUMEN AyED II

TIPOS ABSTRACTOS DE DATOS

ESPECIFICACIÓN:

- * INDICAR SU NOMBRE
- * ESPECIFICAR CONSTRUCTORES (Proc o Func mediante los cuales pueda crear elementos del tipo q' estoy especificando)
- * ESPECIFICAR OPERACIONES
  - * INDICAR LOS TIPOS DE CADA OP. Y CONSTRUCTOR (el encabezado de los Proc o Funciones)
  - * EXPLICAR QUÉ HACEN MEDIANTE LEN. NATURAL
  - * ALGUNAS OPS PUEDEN TENER RESTRICCIONES (Pre- condiciones)
- * ESPECIFICAR UNA OPERACIÓN DE DESTRUCCIÓN (Libera memoria utilizada por los elementos)

```
spec List of T where
constructors
    fun empty() ret l: List of T
    {- crea una lista vacía -}

    Proc add(in e: T, in/out l: List of T)
    {- Agrega el elemento e al
       comienzo de la lista -}

destroy
    proc destroy(in/out l: List of T)
    {- Libera memoria en caso necesario -}

operations
    Fun is_empty(l: List of T) ret b: bool
    {- Devuelve true si l es vacía -}

    Fun head(l: List of T) ret e: T
    {- Devuelve el primer elemento de la lista -}
    {- Pre: not is_empty(l) -}
    ....
```

IMPLEMENTACIÓN

- * DEFINIR UN NUEVO TIPO CON EL NOMBRE DEL TAD
- * IMPLEMENTAR CADA CONSTRUCTOR
- * IMPLEMENTAR CADA OPERACIÓN
- * IMPLEMENTAR OP. DE DESTRUCCIÓN

```
implement List of T where

type node of T = tuple
                   elem: T
                   next: pointer to (node of T)
                 end tuple

typ List of T = Pointer to (node of T)

fun empty() ret l: List of T
    l := NULL
end fun

Proc add(in e: T, in/out l: List of T)
    var p: Pointer to (node of T)
    alloc(p)
    p->elem := e
    p->next := l
    l := p
end Proc

....
```

OPERATIONS DE LISTAS:

Proc TAIL(in/out l: List of T) (ELIMINA EL PRIMER ELEM.)

Proc TAKE(in/out l: List of T, in n: nat) (DEJA LOS 1^{ros} n ELEMS)

Proc DROP(in/out l: List of T, in n: nat) (ELIMINA LOS 1^{ros} n ELEMS)

LIMITACIÓN AL HACER COSAS CON ARREGLOS

FALTA DE FLEXIBILIDAD EN EL TAMAÑO Y TAMBIEN BASTANTE DIFICULTAD PARA REALIZAR OPERACIONES DE INSERCIÓN Y ELIMINACIÓN DE ELEMENTOS DE MANERA EFICIENTE. YA QUE LOS ARREGLOS SON TIPOS ESTÁTICOS, DEBEMOS ESPECIFICAR UN TAMAÑO FIJO AL MOMENTO DE LA DECLARACIÓN.

RECURRENCIAS

ECUACIÓN DE RECURRENCIA

$$T(n) = \begin{cases} c & \rightarrow \text{SI LA ENTRADA ES PEQUEÑA O SIMPLE} \\ a * T\left(\frac{n}{b}\right) + g(n) & \rightarrow \text{CASO CON TAMAÑO} \end{cases}$$

a = CUANTAS LLAMADAS RECURSIVAS HAY

b = QUE FRACCIÓN DEL ORIGINAL ES

g(n) = COSTO COMPUTACIONAL DEL PROCESO DE DESCOMPOSICIÓN Y COMBINACIÓN

$$T(n) \text{ ES DEL ORDEN DE } \begin{cases} n^{\log_b a} & \text{SI } a > b^k \\ n^k \log n & \text{SI } a = b^k \\ n^k & \text{SI } a < b^k \end{cases}$$

DONDE K ES EL ORDEN DE g(n)

ORDENACIÓN ELEMENTAL

INSERTION SORT

- MEJOR CASO → ARREGLO ORD.
- PEOR CASO → ORD. AL REVÉS
- COMPARA E INTERCAMBIA

```
Proc insertion_sort (in/out a: array[1..m] of T)
  For i:=2 to m do
    j:=i
    do j>1 ∧ a[j]<a[j-1] → swap(a, j-1, j)
                           j:=j-1
    od
  od
endProc
```

SELECTION SORT

- SELECCIONA EL MENOR, LO INTERCAMBIA
- SELECCIONA EL MENOR DE LOS RESTANTE

```
Proc selection_sort (in/out a: array[1..m] of T)
  var minp: nat
  For i:=1 to m do
    minp:=i
    For j:=i+1 to m do
      if a[j] < minp then minp:=j fi
    od
    swap(a, i, minp)
  od
endProc
```

ORDENACIÓN AVANZADA

MERGE SORT (ORDENACIÓN POR INTERCALACIÓN)

- VAMOS DIVIDIENDO EL ARREGLO EN MITADES, HASTA QUE HAYA UN SOLO ELEMENTO (UN SOLO ELEMENTO YA ESTÁ ORDENADO)
- VAMOS INTERCALANDO ORDENADAMENTE LAS PARTES

QUICK SORT (ORDENACIÓN RÁPIDA)

- DIVIDE EL ARREGLO ORIGINAL EN 2: UNA CON ELEM. MENORES QUE UN PIVOT Y OTRA CON ELEM MAYORES
- SE REALIZA LO MISMO DE MANERA RECURSIVA HASTA QUE QUEDE UN SOLO ELEMENTO
- EN CADA PASO EL PIVOT ESTÁ EN SU LUGAR