Final de PRED

16 de Junio del 2010

1) **(3 puntos)** Se pide realizar la especificación algebraica del tipo de datos lista de listas de entero. Cada lista de enteros tendrá asociado un entero (no necesariamente igual a su posición) que identifica a esa lista en la lista de listas. Por el contrario, los elementos de las listas de enteros se identifican por su posición implícita empezando por la posición 1. Las operaciones son las siguientes:
-**crear_lista**: crea la lista de listas vacía.
-**insertar_cabeza**: operación que dada una lista de listas, un identificador *ln* y un valor, inserta el valor como primer elemento de la lista con identificador *ln*.
-**insertar_izq**: operación que dada una lista de listas, un identificador *ln*, una posición *pos* y un valor inserta el valor inmediatamente antes que el elemento que ocupa la posición *pos* de la lista con identificador *ln*.
-**suprimir**: operación que dada una lista de listas, un identificador *ln* y una posición *pos*, borra el elemento de la posición *pos* de la lista con identificador *ln*.

2) **(3 puntos)** Se desea implementar el tipo de datos **Diccionario** con las operaciones usuales de **crear_diccionario**, **actualizar** el valor de un índice o clave en un diccionario, **consultar** el valor de un indice o clave en el diccionario y **suprimir** un elemento dado un índice o clave del diccionario.
La implentación se hará mediante tabla de hash abierto (en el libro de Ricardo Peña se denomina tabla dispersa cerrada). Se pide lo siguiente:
a) Definir el tipo de la estructura de datos, su invariante y equivalencia de representación.
b) Implementar la operación **suprimir**.

3) **(2 puntos)** Sea un árbol binario, con valores enteros no repetidos en sus nodos del cual se conocen 2 recorridos:
Preorden: 6,2,1,4,3,5,7,9,8 y
Inorden: 1,2,3,4,5,6,7,8,9

a) Decir cómo se puede deducir la estructura del árbol y dar una representación gráfica del mismo.
b) Dar el recorrido por niveles del mismo

4) **(2 puntos)** Dado el siguiente programa en C++, decid los números de las líneas del programa que producen un error de compilación justificando el motivo, y a continuación, dar el resultado que produciría el programa sin las líneas erróneas.

```
class A  {

   private: int x;
   protected: int y;
   public:
      int z;
      A(){
                                 x=2;
                                 y= 2;
                                 z=3;
       }

      virtual int  g(int i) {
                                 A a;
                                 a.x= a.x +2;
                                 a.y= a.y +2;
                 return a.x + a.y +i;
                 }
};

class B: public  A {

 private: int x2;
 protected: int y2;
 public: int z2;
    B(){
                                 x2=4;
                                 y2=4;
                                 z2=4;
    }

    virtual int  g(int i) {
                                 B b;
                                 A a;
                                 b.x2 = b.x2 +2;
                                 a.z = a.z + 2;
                                 return i + b.x2 +
                                 a.z;}
};
```

```
class C: protected  B {

 private: int x3;
 protected: int y3;
 public: int z3;
    C(){
                                 x3=6;
                                 y3=6;
                                 z3=6;
    }
    virtual int g2(int i)   {
                                 A a;
                                 B b;
                                 C c;
                                 a.y = a.y+2;
                                 b.y2 = b.y2 +2;
                                 c.y3 = c.y3  + 2;
                                 return  b.z2 + c.y3 + i;
                                 }
};

          int main() {
                                 A *a;
                                 B *b;
                                 C *c;
                                 a= new A;
                                 b= new B;
                                 c= new C;
                                 cout << a->g(4) << endl;
                                 a=b;
                                 cout << a->g(4) << endl;
                                 cout << c->g2(4) << endl;
                                 cout << b->z2 << endl;
                                 cout << b->y2 << endl;
                                 cout << b->z << endl;
                                 cout << b->y << endl;

                                 cout << c->z << endl;
                                 cout << c->z2 << endl;
                                 cout << c->y3 << endl
                                 };
```

1)
Especificación Lista_Lista_ent
Usa Entero
Ops
Crear_lista:-> Lista_Lista_ent (Constructora)
Insertar_cabeza: Lista_Lista_ent x Entero X Entero -> Lista_Lista_ent (Constructora)
Insertar_izq: Lista_Lista_ent X Entero x Entero x Entero -> Lista_Lista_ent (parcial)
Insertar_der: Lista_Lista_ent x Entero x Entero x Entero -> Lista_Lista_ent (parcial)
Suprimir: Lista_ent x Entero x Entero -> Lista_Lista_ent (parcial)
Axiomas
Definimos una operación auxiliar long:Lista_Lista_ent x Entero -> Entero que dada una lista de listas y un identificador de lista nos devuelve la longitud de esa lista

long (Crear_lista,idl)= 0
Idl' ≠ idl => long (Insertar_cabeza(ll,idl',v),idl) = long(ll,idl)
Idl'=idl => long (Insertar_cabeza(ll,idl',v),idl) = 1+long(ll,idl)

N <1 => insertar_izq(ll,idl,n,i) indef
n>long(ll,idl) => insertar_izq(ll,idl,n,i) indef
n=0 => suprimir(ll,idl,i) indef
n>long(ll,idl) => suprimir(ll,idl,i) indef

Idl' ≠ idl =>insertar_cabeza(insertar_cabeza(ll,idl,v),idl',v')=
insertar_cabeza(insertar_cabeza(ll,idl',v'),idl,v)

idl' ≠ idl => insertar_izq(insertar_cabeza(ll,idl',v),idl,n,i)=
insertar_cabeza(inserter_izq(ll,idl,n,i),idl',v)
insertar_izq(llidl, 1,i) = insertar_cabeza (ll,idl,i)
n>1 y n <= long(ll) => insertar_izq(insertar_cabeza(ll,idl,v),idl, n,i) = insertar_cabeza (insertar_izq(ll,idl,n-1,i),idl',v)

idl' ≠ idl =>
suprimir(insertar_cabeza(ll,idl',v),idl,n))=insertar_cabeza(suprimir(ll,idl,n),idl',v)
suprimir(insertar_cabeza(ll,idl,v),idl, 1) = ll
n>1 y n <= long(l) => suprimir(insertar_cabeza(ll,idl,v),idl, n) = insertar_cabeza (suprimir(ll,idl,n-1),idl,v)

fespecificacion

2)

modulo indice
usa natural, booleano
ops
funcion h(i:indice) retorna n:natural
funcion igual (i1,i2:indice) retorna b:booleano

```
        funcion valor_nodef () retorna i:indice
        funcion valor_borrado () retorna i: indice
fmodulo

modulo Diccionario
  usa indice, valor
  ops
    …
    accion suprimir (ent i:indice, ent/sal D:Diccionario)
fops

representación
      diccionario = tabla [0..N-1] de ind_val;
      ind_val = tupla
                  i: indice;
                  v: valor;
                ftupla;
```

Invariante de representación

-Para todo índice i y valor v, <i,v> está en el diccionario si y sólo si está entre h(i) y la primera posición libre de la tabla

- Un mismo índice no puede aparecer 2 veces en la tabla

Equivalencia

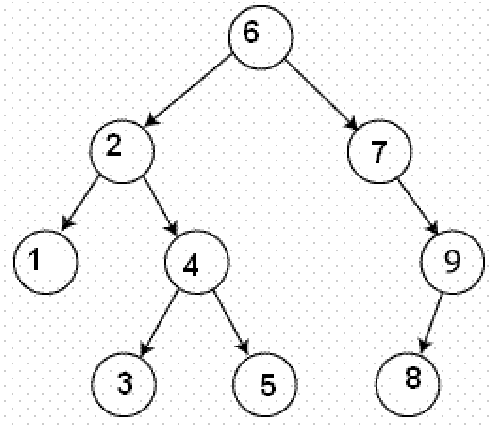
d1 es equivalente a d2 si y sólo si contienen los mismos pares <i1,v1> aunque no necesariamente en el mismo orden

```
accion suprimir(ent i:indice, ent/sal D:Diccionario)
   { Pre: Cierto}
   {Post: Borramos el índice i del diccionario si está, y si no está no hacemos nada}
     var j:natural fiar
      j:= h(i);
      Mientras no igual(d[j].i,i) y no igual(d[j].i, valor_nodef()) hacer
            j:= (j+1)mod N;
      fmientras
      Si d[j].i = i entonces d[j].i := valor_borrado() fsi
faccion
```

3)

a) Sea a un entero x, x', y, y' listas de enteros.

Pre-orden a:**x++y**
In- orden **x'**++ [a]++**y'**
x , **x'** tienen los mismos elementos
y ,**y'** tienen los mismos elementos
el árbol resultado tiene raíz a
x es la enumeración en pre-orden del árbol izquierdo y **x'** es su enumeración en in-orden.
Asimismo **y** es la enumeración en pre-orden del árbol derecho e **y'** su enumeración en in-orden.



En la enumeración en pre-orden, el primer elemento es la raíz (6), por lo que todos losvalores anteriores a 6 en la enumeración en in-orden pertenecen al árbol izquierdo y los siguientes al árbol derecho. La enumeración pre-orden de estos árboles se obtiene tomando de la enumeración pre-orden del árbol los elementos de árbol izquierdo y árbol derecho correspondiente (que están consecutivos).

El resultado será un árbol de raíz 6 cuyo árbol izquierdo tiene como enumeración pre-orden 2,1,4,3,5 y como enumeración in-orden 1,2,3,4,5. De esto se obtiene que el árbol izquierdo tiene raíz 2, y su árbol izquierdo es 1 (valores anteriores a 2 en la enumeración in-orden) y árbol derecho con enumeración pre-orden 4,3,5 y enumeración in-orden 3,4,5. Por lo tanto el árbol derecho del árbol con raíz 2 será un árbol con raíz 4 y como 4 aparece precedido por 3 en la enumeración in-orden su árbol izquierdo será un solo nodo de etiqueta 3 y su árbol derecho será 5.
Volviendo al árbol derecho del árbol con raíz 6 será el árbol cuya enumeración en pre-orden es 7,9,8 e in-orden 7,8,9. De estas dos enumeraciones tenemos que el árbol tiene raíz 7 y árbol izquierdo vacio. Su árbol derecho tiene enumeración en pre-orden 9,8 y 8,9 en in-orden, por lo que su raíz es 9 y su árbol izquierdo es 8 ya que aparece antes del 9 en la enumeración in-orden.

b) El recorrido por níveles sería 6 2 7 1 4 9 3 5 8

4) Los errores serían los siguientes:
Línea 53: y es un atributo protegido en A.
Línea 54: y2 es un atributo protegido en B.
Línea 72: y2 es un atributo protegido en B.
Línea 74: y es un atributo protegido en B.
Línea 75: z es un atributo privado en C.
Línea 76: z2 es un atributo protegido en C.
Línea 77: y3 es un atributo protegido en C.

El resultado sin las líneas erróneas sería: 12 15 16 4 3