Final de PRED

14 de Junio del 2011

1) **(3 puntos)** Se pide realizar la especificación algebraica del tipo de datos cola de enteros con las operaciones usuales de **crear_cola, insertar** (operación que inserta un nuevo elemento a la cola), **borrar** (operación que borra el elemento que está al principio de la cola), **primero** (operación que obtiene el primer elemento de la cola) y **longitud** (operación que devuelve el número de enteros de la cola).

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 índice o clave en el diccionario y **suprimir** un elemento dado un índice o clave del diccionario.
La implementación 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)** Un heap (además de una técnica de implementación de la gestión dinámica de memoria) es un tipo de árbol binario que tiene la propiedad que el valor que hay en cada nodo es menor que los valores que hay en sus dos hijos. Se pide diseñar un algoritmo que dado un heap decide si cumple la propiedad del heap recorriendo el árbol por niveles. Se sabe que este algoritmo sólo se utilizará para árboles binarios completos o semicompletos y por tanto la representación del heap ha de ser mediante una tabla.

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.

```
1  class A  {
2
3     Private: int x;
4     protected: int y;
5     public:
6        int z;
7        A(){
8                                  x=4;
9                                  y= 4;
```

```
39  class C: protected  B {
40
41    private: int x3;
42    protected: int y3;
43    Public: int z3;
44       C(){
45                                  x3=8;
46                                  y3=8;
47                                  z3=8;
```

```
10                              z=4;
11        }
12
13       virtual int  g(int i) {
14                              A a;
15                              a.x= a.x +2;
16                              a.y= a.y +2;
17                  Return a.x + a.y +i;
18                  }
19  };
20
21  class B: public  A {
22
23   private: int x2;
24   protected: int y2;
25   public: int z2;
26     B(){
27                              x2=6;
28                              y2=6;
29                              z2=6;
30     }
31
32     Virtual int  g(int i) {
33                              B b;
34                              A a;
35                              b.x2 = b.x2 +2;
36                              a.z = a.z + 2;
37                              return i + b.x2 +a.z;}
38  };
```

```
48     }
49     virtual int g2(int i)   {
50                                  A a;
51                                  B b;
52                                  C c;
53                                  a.y = a.y+2;
54                                  b.y2 = b.y2 +2;
55                                  c.y3 = c.y3  + 2;
56                                  return  b.z2 + c.y3 + i;
57                                  }
58  };
59
60                int main() {
61                                  A  a;
62                                  B  b;
63                                  C  c;
64
65
66
67                                  cout << a.g(4) << endl;
68                                  a=b;
69                                  cout << a.g(4) << endl;
70                                  cout << c.g2(4) << endl;
71                                  cout << b.z2 << endl;
72                                  cout << b.y2 << endl;
73                                  cout << b.z << endl;
74                                  cout << b.y << endl;
75                                  cout << c.z << endl;
76                                  cout << c.z2 << endl;
77                                  cout << c.y3 << endl
78                                  };
```

1)

Especificación Cola_enteros

Usa Enteros, Naturales

Ops

Crear_cola:-> Cola_enteros (Constructora)
insertar: Enteros x Cola_enteros -> Cola_enteros (Constructora)
primero: Cola_enteros -> Enteros (parcial)
borrar: Cola_enteros -> Cola_enteros (parcial)
long: Cola_enteros -> Naturales

Axiomas

primero(Crear_cola) ↑
borrar(Crear_cola) ↑
primero(añadir(i, Crear_cola)) = i
¬ vacia(c) => primero (insertar(i,c))=primero(c)
borrar(insertar(i, Crear_cola)) = Crear_cola
¬ vacia(c) => borrar(insertar(i,c))= insertar(i,borrar(c))
long(Crear_cola)=0
long(insertar(i,c))=1+ long(c)
vacia(Crear_cola)= cierto
vacia(insertar(i,c))= falso

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
diccionario = tabla [0..N-1] de ind_val;

representación
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:** Ha de haber al menos un valor_nodef() en la tabla }
{**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)

funcion es_heapin(t:tabla[1..N] de enteros) **retorna** esh:booleano
{Pre:Cierto}
{Post: esh ⇔ t[1..N] es un heap}
var i:natural **fvar**

```
i:=1;
esh:=cierto;
{Inv: Los nodos del árbol de 1 hasta i-1 cumplen la propiedad del heap y ¬esh =>
t[i] no cumple la propiedad del heap}

Mientras (i<= N div 2 -1) ∧ esh hacer

  Si t[i] >= t[2*i] ∨ t[i] >= t[2*i+1] entonces esh:=falso;
  sino i:=i+1;
  fsi
fmientras
i:=i+1;
Si esh entonces Si N mod 2 = 0 entonces esh := esh ∧ t[i] < t[2*i]
                                        sino esh := esh ∧ t[i] < t[2*i] ∧ t[i] < t[2*i+1]
                    fsi
fsi
retorna esh
ffuncion
```

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: 16 16 20 6 4