La soluzione qui riportata è tratta dal compito dello studente Simone Cortellesi

```
//=======================================
//=================1° ES=================
//=======================================
int SeqBitonica (int v[], int n) {
//PREC.: n > 2 (la minima sequenza bitonica ha 3 elementi)
//POSTC.: restituisce 1 se gli elementi dell'array formano una seq. bitonica,
//0 altrim
int i = 0;
assert(n > 2);
while (i < n-2 && v[i] < v[i+1]) i++;
if (i == 0 || i == n-2) return 0;
while (i < n-1 && v[i] > v[i+1]) i++;
if (i == n-1) return 1; else return 0;
}


//=======================================
//===============2° ES.=================
//=======================================
int nSomma(TreePtr t){
/* prec: l'albero in input è un albero binario i cui nodi contengono interi.
postc: dà 1 se ogni nodo a livello pari contiene la somma dei nodi nel
sottoalbero in esso radicato, 0 altrimenti*/
int ris = 1;
nApp(t, 0, &ris);
return ris;
}

int nApp(TreePtr t, int liv, int* r) {
/*'prec: t != NULL,
post: restituisce la somma dei nodi di t, radice inclusa
*/
int somma = 0;
if (foglia(t)) return t->elem;
if (t->left) somma += nApp(t->left, liv+1, r);
if (t->right) somma += nApp(t->right, liv+1, r);
if (t->elem != somma && liv % 2 == 0) *r = 0;
return t->elem + somma;
}


//=======================================
//=============3° ES.====================
//=======================================

//=============File Aula.h==============
typedef struct aula* aulaptr;
typedef struct dotaz* dotazptr;
//costruttore
aulaptr costr (char* n);
//PREC.: n != NULL
//POSTC.: restituisce il puntatore ad una struttura di tipo aula
//con nome inizializzato a n e gli altri attributi a valori di default

//distruttore
void distr (aulaptr p);
//PREC.: p != NULL
//POSTC.: dealloca la memoria associata a p

//estrattori
char* a_nome (aulaptr p);
//PREC.: a != NULL
//POSTC.: restituisce il nome di p
```

```
float a_sup (aulaptr p);
//PREC.: p != NULL
//POSTC.: restituisce la superficie di p
int a_cap (aulaptr p);
//PREC.: p != NULL
//POSTC.: restituisce il numero di studenti che p può contenere
int a_prev (aulaptr p);
//PREC.: p != NULL
//POSTC.: restituisce il numero di studenti previsti per p
dotazptr a_prev (aulaptr p);
//PREC.: p != NULL
//POSTC.: restituisce le dotazioni per p

//funzionalità
void set_sup (aulaptr p, float s);
//PREC.: p != NULL && s > 0
//POSTC.: s è la superficie di p
void set_cap (aulaptr p, int k);
//PREC.: p != NULL && k >= 0
//POSTC.: l'aula p contiene al più k studenti
void set_prev (aulaptr p, int k);
//PREC.: p != NULL && k >= 0
//POSTC.: si prevede che l'aula p conterrà k studenti
void set_dotaz (aulaptr p, dotazptr q);
//PREC.: p != NULL && q != NULL
//POSTC.: l'aula p fornisce le dotazioni q
int a_ok (aulaptr p, float d);
//PREC.: p != NULL && d > 0
//POSTC.: restituisce 1 se l'aula p può garantire ad ogni studente almeno
//  d metri quadri, 0 altrimenti
int a_piena (aulaptr p);
//PREC.: p != NULL
//POSTC.: restituisce 1 se si prevede che l'aula sarà piena, 0 altrim

//=============File Paese.c===============
typedef struct aula {
char* nome;
float sup;
int cap;
int prev;
dotazptr dot;
}
typedef struct dotaz {
char* dotazione;
dotaz* succ;
}

//=======================================
//=============4° ES.====================
//=======================================
Dati di test a scatola nera per

int SeqBitonica (int v[], int n) {
//PREC.: n > 2 (la minima sequenza bitonica ha 3 elementi)
//POSTC.: restituisce 1 se gli elementi dell'array formano una seq. bitonica,
//0 altrim

Diamo per scontato che il valore di n sia la dimensione del vettore v.
Dalle post condizioni cerchiamo di suddividere il dominio dei dati di input in
classi di elementi equivalenti rispetto al test:
n = 0,1,2 => ris = 0: nell'array ci deve essere una prima parte strettamente
   crescente (e quindi devo avere almeno 2 elementi) e poi una parte
   strettamente decrescente (e quindi devo avere almeno un altro elemento).
Per n > 2 dobbiamo prendere un vettore con n elementi che contiene una prima
   sequenza (di k elementi) crescente e una seconda sequenza (di n-k elementi)
   decrescente. Dobbiamo considerare tutti i casi per 0 < k < n. Prendiamo n = 5
```

e consideriamo i casi k = 2, k=3 e k=4; per ogni caso, diamo un esempio di vettore per il quale la soluzione è 1 e uno per il quale la soluzione è 0:

v=[1,2,3,4,5] il risultato è 0 per tutti i k.
v=[2,3,2,1,0] con risultato 1 per k=2 e 0 per k =3,4
v=[1,2,3,2,1] con risultato 1 per k=3 e 0 per k =2,4
v=[1,2,3,4,3] con risultato 1 per k=4 e 0 per k =2,3.

Glass box

```
int SeqBitonica (int v[], int n) {
//PREC.: n > 2 (la minima sequenza bitonica ha 3 elementi)
//POSTC.: restituisce 1 se gli elementi dell'array formano una seq. bitonica,
//0 altrim
int i = 0;
assert(n > 2);
while (i < n-2 && v[i] < v[i+1]) i++;
if (i == 0 || i == n-2) return 0;
while (i < n-1 && v[i] > v[i+1]) i++;
if (i == n-1) return 1; else return 0;
}
```

Visto che la precondizione è trattata con l'assert, la controlliamo chiamando la funzione su valori che le violano. Prendiamo un valore di n < 3.
Sia quindi n >= 3. Consideriamo il primo ciclo while; 2 casi

1. v[0] >= v[1]: in questo caso, la guardia del ciclo non è soddisfatta, non si entra nel while (pertanto i resta a 0) e l'if porta a restituire 0.
2. v[0] < v[1]: 2 casi:
   a. itera finchè i = n-2: p.e., v = [1,2,3,4] e restituisce 0
   b. si ferma per i = k, con 0 < k < n-1. A questo punto, la guardia dell'if non è soddisfatta e passa al secondo while; di nuovo, 2 casi:
      i. v[k] >= v[k+1]: in questo caso, la guardia del ciclo non è soddisfatta, non si entra nel while (pertanto i resta a k) e l'if porta a restituire 0.
      ii. v[k] < v[k+1]: 2 casi:
          - itera finchè i = n-2: p.e., v = [1,2,3,2] e restituisce 1
          - si ferma per i < n-1. A questo punto, la guardia dell'if non è soddisfatta e porta a restituire 0.