preparazione II compitino

seconda puntata

1) Dire se il seguente programma è corretto o meno e perché. In caso pensiate sia corretto dite cosa stampa e perché.

```
const int x=99, *p=&x;
int* q= const_cast<int*>(p);
(*q)++;
cout<<x<<*q<<*p<<endl;
```

1')

```
int x=99, *const p=&x;
int * const * q=&p;
(**q)++;
```

2) Cercate di scrivere una PRE ed una POST sensate

```
int F(nodo* R){
if(! R)  return 0;
if(R->left && R->right)  return F(R->left) + F(R->right);
if(R->left|| R->right) return 1+F(R->left) + F(R->right);
return 0;
}
```

```
int F(nodo*R)
{ if(!R->left && !R->right)  return 1;
   if(!R->left) return F(R->right);
   if(!R->right) return F(R->left);
   return F(R->left)+F(R->right);}
```

Esercizio 1 del 3/3/2014

Si legge da "input" gli interi k, y e dim, in cui y può essere un intero qualsiasi, mentre k e dim sono maggiori di 0, dopo di che **si costruisce una lista concatenata con dim nodi** del tipo seguente:
struct nodo{int info1,info2; nodo* next; nodo(int a=0, int c=0, nodo* b=0){info1=a;info2=c; next=b;}};
ed in cui nel campo info1 di ciascuno dei dim nodi si deve inserire un valore letto da "input" (ce ne sono dim) mentre in info2 si mette semplicemente la posizione del nodo nella lista, cioè 0,..,dim-1.

A questo punto si richiedono 2 funzioni F1 ed F2 entrambe col seguente prototipo:
nodo* F1/2(nodo*&L, int y, int k, int & v) e che soddisfano le seguenti coppie di pre- e post-condizioni:

PRE_F1=(L è una lista corretta, y e v sono interi qualsiasi, mentre k>0, L=vL)

POST_F1=(se vL contiene meno di k nodi con campo info1=y, allora L=vL e F1 restituisce 0, altrimenti, L è la lista ottenuta da vL togliendo da vL gli ultimi (per posizione) k nodi con campo info1=y, mentre F1 restituisce la lista dei nodi tolti da vL in ordine di posizione in vL.....è possibile che sia anche necessario aggiungere qualcosa su v)

Esempio: Sia L la lista costruita leggendo i dim interi da "input", come segue:
L=(3,0),(2,1),(0,2),(3,3),(0,4),(0,5)(1,6),(0,7),(2,8),(0,9),(3,10),
supponiamo anche che k=2 e y=3. Evidentemente L contiene almeno 2 nodi con info1=3 (ne contiene 3) e quindi sia F1 che F2 modificheranno la lista che ricevono in input.
F1 restituirà col return la lista (3,3),(3,10), mentre L sarà diventata: (3,0),(2,1),(0,2),(0,4),(0,5)(1,6),(0,7),(2,8),(0,9)
mentre F2 restituirà col return (3,0),(3,3) con L che sarà diventata:
=(2,1),(0,2),(0,4),(0,5)(1,6),(0,7),(2,8),(0,9),(3,10).
Se k fosse 4, entrambe le funzioni restituirebbero 0 lasciando L inalterata. Se k fosse 3, invece le 2 funzioni avrebbero identico output, sia col return che col parametro L.

PRE=(INP contiene dim interi, dim>=0, n def.)

```
void crea(nodo*&L, int dim, ifstream & INP, int n)
{
  int x;
  if(dim>0)
    {
      INP>>x;
      L=new nodo(x,n,0);
      crea(L->next, dim-1,INP,n+1);
    }
  else
   L=0;
}
```

POST=(restituisce in L lista con i dim interi di INP e indici crescenti da n in poi)

```
nodo* F1(nodo*&L, int y, int k, int & v)
{
 if(L)
  {
   if (L->info1==y)
     {
       v++;
       nodo*T=F1(L->next, y, k, v);
       if(v<k)
        {
          nodo*w=L;
          L=L->next;
          w->next=T;
          v++;
          return w;
        }
        else
         return T;
     }
    return F1(L->next,y,k,v);
  }
  else
 {
    if(v>=k)
       v=0;
    else
       v=k;
    return 0;
}
}
```

PRE_F1=(L è una lista corretta, y e v sono interi qualsiasi, mentre k>0, L=vL, v=vv)

POST_F1=(se vL contiene m nodi con campo info1=y e m+vv<k, allora L=vL e F1 restituisce 0 e v=k, altrimenti, L è la lista ottenuta da vL togliendo da vL gli ultimi (per posizione) min(m,k) nodi con campo info1=y, e restituisce col return la lista dei nodi tolti da vL in ordine di posizione in vL e v =min(m,k))

POST_F1=(se vL contiene meno di k nodi con campo info1=y, allora L=vL e F1 restituisce 0, altrimenti, L è la lista ottenuta da vL togliendo da vL gli ultimi (per posizione) k nodi con campo info1=y, mentre F1 restituisce la lista dei nodi tolti da vL in ordine di posizione in vL.....è possibile che sia anche necessario aggiungere qualcosa su v)

caso base:

!L) vL=L =0

m=0 e se m+v>=k allora v=0 altrimenti v=k come richiesto

passo induttivo:

i) L->info1==y vale PRE_ric e quindi se m ed m' sono il n. di nodi con info1==y in L e L->next allora m=m'+1 e quindi v++ causa il fatto che l'invocazione ricorsiva calcola vv+m=vv+1+m' quindi la POST_ric ci dice che

a) se vv+m<k allora v=k e L->next=vL->next e T=0.

➔ F1 restituisce T, lascia L=vL e v=k

ok con POST

b) se vv+m>=k allora T è la lista degli ultimi min(m',k) nodi di vL->next con info1=y, v=min(m',k) L->next è vL->next a cui sono stati tolti i nodi in T;

(A) v<k ➔ F1 stacca il nodo L dalla lista e lo aggiunge a T, quindi L diventa vL meno m= m'+1 nodi con info1=y e questi nodi vengono restituiti col return nell'ordine giusto, v aumenta di 1 e diventa pari a m che è <=k

(B) v=k ➔ F1 lascia L ferma, restituisce T e lascia v com'è. Funziona sia che T=0 sia che T contenga k nodi.

ii) L->info1!=y allora il lavoro va fatto tutto su L->next, L e v non cambiano

Funzione F2

PRE_F2= =(L è una lista corretta, y e v sono interi qualsiasi, mentre k>0, L=vL, v=vv)

POST_F2=(se vL contiene m nodi con campo info=y, allora se v+m<k allora F2 lascia L=vL,restituisce 0 e v=0,
altrimenti,
-L è la lista ottenuta da vL togliendo da vL i primi max((k-vv),0) nodi con campo info1=y,
-restituisce col return la lista dei nodi tolti da vL
-e v=1
)

```
nodo* F2(nodo*&L, int y, int k, int & v)
{
  if(L)
   {
    if(L->info1==y)
    {
      int n=v; v++;
     nodo* q=F2(L->next,y,k,v);
     if(n<k && v==1)
      {
        nodo*x=L;
        L=L->next;
        x->next=q;
        return x;
       }
      else
         return q;
     }
     else
        return F2(L->next,y,k,v);
    }
```

```
 else
{
    if(v>=k)
      v=0;
     else
      v=1;
     return 0;
}
}
```

caso base:

!L) vL=L =0 return 0

m=0, v+m< k => v=0

v+m>=k v=1

=> POST

passo induttivo:

i) L->info1==y vale PRE_ric e quindi se m ed m’ sono il n. di nodi con info1==y in L e L->next allora m=m’+1 e quindi v++ causa il fatto che l’invocazione ricorsiva calcola vv+m=vv+1+m’ quindi la POST_ric ci dice che

a) se vv+m<k allora v=0 e L->next=vL->next e q=0.

➔ F2 restituisce q, lascia L=vL e v=0

ok con POST

b) se vv+m<=k allora q è la lista dei primi max((k-vv-1),0) nodi di vL->next con info1=y, v=1, e L->next è vL->next a cui sono stati tolti i nodi in q;

(A) v=1 e vv<k ➔ F2 stacca il nodo L dalla lista e lo aggiunge a q, quindi L diventa vL meno i primi max((k-vv),0) nodi con info1=y e questi nodi vengono restituiti col return nell'ordine giusto, v resta 1

(B) v=1 e vv>=k ➔ F2 lascia L ferma, restituisce q e lascia v com'è. Funziona sia che q=0 sia che q contenga k nodi.

(ii) facile