Compito di Programmazione del 21 marzo 2011

Teoria:

(i) Data la seguente funzione ricorsiva, inserire appropriate PRE e POST

```
//PRE= ??
int F(Nodo* a) {
 if (!a->left && !a->right) return 0;
 if (!a->left &&  a->right) return F(a->right);
 if (!a->right &&  a->left) return F(a->left);
 return 2+F(a->left)+F(a->right);
} //POST=??
```

(ii) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio il motivo.

```
int* f(int **p){int b=3,*x=&b; *p=x; x=*p; return x+1; }

main() {int b[]={2,3},*q=b; *f(&q)=*q; cout<<b[0]<<b[1]<<*q;}
```

Programmazione: il problema da considerare è lo stesso sia per la parte iterativa che per quella ricorsiva. Si tratta di estrarre da una lista concatenata data L[1] i nodi in posizione k, 2k, 3k ... per un dato k>1. Si devono restituire 2 liste, quella dei nodi "rimasti" e quella dei nodi "estratti". Entrambe le liste vanno restituite attraverso i parametri L ed E passati per riferimento.

Esempio: *data L= 0-> 1-> -4-> 2-> 3 e k=2, si deve produrre le due liste: L= 0-> -4-> 3 e E= 1-> 2.*
Per k=3, si deve produrreL= 0-> 1-> 2-> 3 ed E=-4.

Attenzione: Si osservi che nessun nuovo nodo va creato e nessun nodo della lista originale va distrutto.

Esercizio iterativo: la funzione iterativa da realizzare deve avere il seguente prototipo:
void F(nodo*&L,nodo*&E,int k)

a) Descrivere Pre e Postcondizione della funzione F;

b) realizzare la funzione F;

c) scegliere un ciclo significativo di F ed associare ad esso un invariante (che descriva al meglio quello che il ciclo calcola);

Esercizio ricorsivo: la funzione ricorsiva da realizzare deve avere il seguente prototipo:
void G(nodo*&L,nodo*&E,int k, int i) in cui il parametro i serve a specificare la posizione del nodo corrente nella lista iniziale;

a) descrivere Pre e Postcondizione della funzione G;

b) realizzare la funzione G;

c) delineare una dimostrazione di correttezza di G;

Opzionale: l'ipotesi che k>1 riveste un'importanza diversa per l'esercizio iterativo e per quello ricorsivo. Spiegare qual è questa differenza e perché è così.

---

[1] I nodi di L sono del solito tipo struct nodo{int val; nodo* next;};

//Parte iterativa:
PRE=(L è una lista non vuota, E è una lista vuota, k>1, sia vL=L)

```
void F(nodo*&L, nodo*& E, int k)
{
 int i=1;
 nodo*x=L,*q=0;
 E=0;
 while(x->next)
  {
   if(i+1==k) //x->next e' da estrarre
        {
         nodo*p=x->next;
         x->next=x->next->next;
         p->next=0; // per essere sicuri che E finisce con nodo con campo next=0
         if(!E)
          E=q=p;
         else
          {
           q->next=p;
           q=p;
          }
         i=0;      // viene messo a 1 dopo l'if
        }
   i++;
   x=x->next;
  }
}
```

Notazione: data L, L[n] è il nodo n-esimo di L. Sia r=lung(L)/k
POST=(se r >0 allora E=vL[k]->…->vL[r*k], altrimenti E=0, L=vL-E)

//Esercizio ricorsivo:

PRE=(L ed E liste anche vuote, k>0, 0<i<=k, vL=L)

```
void G(nodo*&L,nodo*&E,int k, int i)
{
 if(L)
  {
   if(i==k)
        {
         E=L;
         L=L->next;
         G(L,E->next,k,1);
        }
   else
        G(L->next,E,k,i+1);
  }
 else
  E=0;  //campo next dell'ultimo nodo di E viene messo a 0
}
```

POST=(E=L[k-i+1]->L[2*k-i+1]-> e così via, L=vL-E)
L'importante è osservare che i indica che si mantengono in L i primi (k-i) nodi, per poi togliere il (k-i+1)-esimo, dopo di che si mantengono in L i prossimi k-1, se ne toglie uno e così via.