ragionando sull’esame del 31/3/2014

il testo dell’esame è nel sito moodle degli esercizi

Teoria

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

```
//PRE= (a è albero corretto)
int F(Nodo* a) {
if(!a)
  return 0;
if (a->left)    return 1+F(a->left);
if(a->left || a->right) return 2+F(a->left)+F(a->right);
return  3+F(a->left)+F(a->right);
} //POST=(viene considerato il cammino dalla radice ad una foglia che è il più a sinistra possibile, cioè ogni volta che c'è il figlio sinistro si va a sinistra e solo se non c'è si va a destra, e ad ogni nodo di questo cammino si associa un intero in questo modo:
-se si va a sinistra, 1;
-se si va a destra, 2;
-per la foglia finale, 3.
La funzione restituisce la somma di tutti gli interi del cammino
```

(2) Considerare il seguente programma e dite se è corretto o no. Se pensate che sia corretto, spiegate i passi dell'esecuzione. Se pensate che sia sbagliato, spiegate con precisione perché.

```
int* f(int **p){int b=3,*x=&b; **p=*x; *x=**p; return x; }
main() {int y=5, b=2,*q=&b; *f(&q)=y*2; cout<<y<<b<<*q;}
```

il main passa l'L-valore di q che diventa l'R-valore di p. x punta a b locale alla funzione f, e **p è invece il b del main che riceve l'R-valore del b di f, poi *x==**p non fa nulla perché significa che il b di f riceve l'R-valore del b del main (che ha appena ricevuto l'R-valore del b di f!). Infine si ritornal'R-valore di x che punta al b di f! Quindi è **un dangling pointer** e *f(&q) lo dereferenzia. Errore di tipo: si accede una variabile (b di f) che è stata deallocata.

Ho inserito anche la seconda versione della domanda (2) usata nel compito:

(2-b) Considerare il seguente programma e dite se è corretto o no. Se pensate che sia corretto, spiegate i passi dell'esecuzione. Se pensate che sia sbagliato, spiegate con precisione perché.

```
int* f(int **p){int b=3,*x=&b; **p=*x;  x=*p; return x; }
main() {int y=5, b=2,*q=&b; *f(&q)=y*2; cout<<y<<b<<*q;}
```

In questa versione in f c'è x=*p che cambia completamente le cose rispetto al precedente esercizio. Infatti x riceve *p cioè l'R-valore di q che punta a b del main, quindi f ritorna l'indirizzo di b del main e la sua dereferenziazione non causa errori: b del main riceve 10 e quindi la stampa produce, 5, 10, 10.

Il programma è corretto

3) Un programma può essere scritto su più file. Ogni file può venire compilato da solo, ma alla fine, tutti i file dovranno venire compilati assieme. Supponiamo di avere un programma scritto su due file e supponiamo che le funzioni scritte su ciascuno dei due file usino uno stesso tipo struttura P ed una stessa funzione f che sono come segue:

```
struct P{int a,b; P* next;};
P* f(P* x){.....}
```

Esattamente cosa di P e cosa di f deve essere scritto su ciascun file? Spiegare brevemente la risposta.

un file deve avere l'intera definizione di f, mentre l'altro deve contenere solo il suo prototipo. Entrambi i file devono contenere l'intera definizione di P (esattamente la stessa). Per spiegazioni leggere il capitolo 9.

parte iterativa

si tratta di fare un ciclo che invoca ripetutamente 2 funzioni, diciamo p1 e p2 che fanno il seguente

p1: calcola la posizione del primo nodo di T t. c. info1=P[i], chiamiamo x1 questa posizione

p2: a partire da x1 calcola di quanto si può prolungare al massimo il match, cioè se il nodo x+1 ha campo info1=P[i+1], se x+2 ha campo info1=P[i+2]eccetera fin dove possibile, chiamiamo x2 la lunghezza di questo match

allora con x1 e x2 possiamo costruire un nuovo nodo da aggiungere a X e poi ritornare all'inizio ciclo e ricominciare con p1 e poi p2, i aumenta di x2

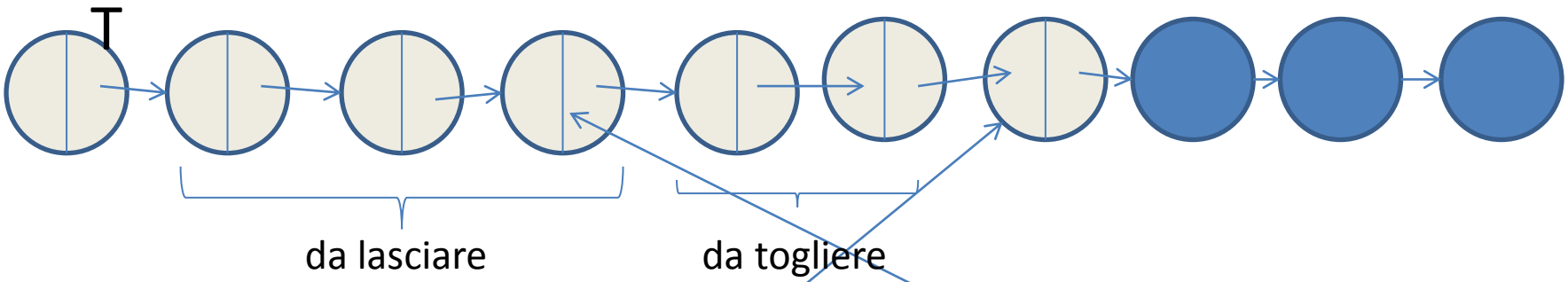
bisogna pensare che p1potrebbe non trovare alcun match, in questo caso p1 potrebbe restituire un valore speciale, per esempio -1
se questo succede, il ciclo di M1 deve terminare.
X non può più essere allungata.

```
while (T && i<dimP && !stop )
{
 int x1=p1(T,P[i]);
 if (x1==-1) then stop=true;
 else
 int x2=p2(sposta(T,x1),P,i,dimP);
 //aggiungi nodo(x1,x2) a X e i=i+x2;
}
```

sposta è una funzione che da col return il x1-esimo nodo di T

Parte ricorsiva

supponiamo che X=(3,2)->Resto e che T sia così:



visto che T è passato per riferimento, esso è il campo next del nodo che precede il primo di T
dobbiamo raggiungere il campo next di questo nodo (Z1) e farlo puntare a questo (Z2)

è facile con 2 opportune funzioni F1 e F2 come segue

```
nodo*& F1(nodo*&T, int inf1)
{
  if(inf1==0) return T;
  return F1(T->next, inf1-1);
}
```
//POST=(restituisce per riferimento un puntatore all'info1-esimo nodo di T ed è alias del campo next del nodo precedente)

```
nodo* F2(nodo* T,int info2)
{
  if(info2==0) return T;
  nodo*x=T;
  T=T->next;
   delete x;
  return F2(T,info2-1);
}
```

//POST=(dealloca i primi inf2 nodi di T e restituisce quello che resta)

```
void TB(nodo*& T, nodo*X)
{
  if(X  && T)
   {
     nodo *& Z1=F1(T,X->info1);
     nodo * Z2=F2(Z1,X->info2);
     Z1=Z2;
     TB(Z1,X->next);
  }
}
```

Si osservi che F1 deve restituire una variabile (per riferimento) visto che Z1 viene poi usata a sinistra dell'assegnazione Z1=Z2, mentre F2 può semplicemente restituire un valore visto che Z2 è usata a destra dell'assegnazione.

Si osservi anche nodo *& Z1=F1(T,X->info1) che fa in modo che Z1 sia alias del campo next dell'ultimo nodo della parte percorsa da F1.

Con questi elementi scrivere TC dovrebbe essere facile

La prova di correttezza di TB funziona così:
casi base: non c'è niente da fare e T resta uguale.

passo induttivo: F1 ed F2 trasformano T come richiesto dal primo elemento di X, l'invocazione ricorsiva con parametro Z1 che è il resto della lista T garantisce per l'ipotesi induttiva che il resto della lista verrà trasformato come richiesto dal resto di X.

C’è un’altra soluzione possibile in cui la ricorsione di TB fa tutto senza bisogno di F1 e F2

```
void TB(nodo*& T, nodo*X)
{
  if(X && T)
   {
     if(X->info1)
      {X->info1--; TB(T->next,X);} // (1)
     else
       if(X->info2)
         {X->info2--; nodo* y=T; T=T->next; delete y; TB(T,X);} //(2)
        else //X->info1=0 e X->info2=0
          TB(T,X->next);
  }
}
```

//la ricorsione (1) fa quello che fa F1: quando la ricorsione (1) finisce e inizia la (2), T è esattamente uguale a Z1 resituita da F1 nella soluzione precedente e la sequenza di T=T->next della ricorsione (2) ottiene lo stesso risultato di Z1=Z2 della soluzione precedente.