Scritto di Programmazione del 20 luglio 2011

Teoria

1) Scrivere PRE e POST opportune per la seguente funzione ricorsiva:

```
//PRE= ??
bool H(nodo*L)
{
 if(!L)
   return true;
 return !H(L->next);
}
//POST= ??
```

2) Considerate il seguente programma e decidete se contiene qualche errore o no. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio il motivo. In entrambi i casi sarà apprezzato un disegno dell'evoluzione della memoria che contiene i dati durante l'esecuzione. Si ricorda che la dereferenziazione ha precedenza sulla somma e sulla differenza.

```
int & g(int ** x){int *p=*x-1; *p=**x+*p; return *(p-1);}
main() {int X[]={1,2,3}, *q=X+2; g(&q)=X[1]; cout<<X[0]<<X[1]<<X[2]<<endl;}
```

Programmazione: Dato un intero k>0, si tratta di leggere interi da cin, continuando fino a che non si legga la sentinella -2 e, dopo averla letta, si deve distinguere tra due situazioni:

i) prima di leggere la sentinella -2 sono stati letti al più k interi;

ii) prima di leggere la sentinella -2 sono stati letti più di k interi.

Nel caso (i) si devono stampare tutti i valori letti (prima della sentinella) nello stesso ordine in cui sono stati letti. Nel caso (ii) si devono stampare gli ultimi k valori letti (prima della sentinella) nell'ordine in cui sono stati letti. Per eseguire questo compito il programma deve usare un array M di k interi (allocato dinamicamente) . Se siamo nel caso (i), M è ovviamente sufficiente a contenere i valori letti prima della sentinella. Se invece siamo nel caso (ii), chiaramente le k posizioni di M sono sufficienti per contenere gli ultimi k valori letti. Il problema è che man mano che si leggono i valori, non si sa quali siano gli ultimi k e quindi, dopo aver letto in M i primi k valori, il k+1-esimo va messo al posto del primo letto (che sicuramente non è tra gli ultimi k letti e quindi può venire dimenticato), il k+2-esimo letto va messo al posto del secondo (per lo stesso motivo di prima) e così via. Insomma M conterrà in ogni momento gli ultimi k letti in quel momento, ma in generale non li conterrà a partire dalla prima posizione in poi. La gestione di M è in un certo senso "circolare" come illustra il seguente esempio.

Esempio: *Consideriamo innanzitutto il caso (i). Supponiamo che cin contenga 1 2 -2 e che k sia 4. Allora il programma inserisce in M i due interi che precedono -2 , cioè M sarà [1,2,_,_] e poi stamperà 1 e 2.*

Consideriamo ora il caso (ii). Supponiamo che cin contenga [3, 1, 2, 4, 0,-2], e che k sia 2. Allora il programma da realizzare legge uno alla volta i valori inserendoli in M (che ha ha k=2 elementi), quindi M diventa dopo ogni lettura:

[3,_], [3,1], [2,1],[2,4],[0,4] e infine viene letto -2 e il programma termina stampando 4 e 0 che sono gli ultimi 2 valori letti. Se invece k fosse uguale a 3, i valori di M sarebbero:

[3,_,_], [3,1,_],[3,1,2],[4,1,2],[4,0,2] e verrebbe stampato 2, 4 e 0.

Quindi il punto essenziale è la gestione di M che deve venire usato in modo "circolare". In particolare, in M sono ammessi solo inserimenti di nuovi valori e non sono ammessi spostamenti di valori già presenti.

Parte iterativa:

1) Realizzare una funzione iterativa G con prototipo void G(int k) che realizza l'operazione descritta in precedenza. G deve creare dinamicamente un array M di k interi ed usarlo nel modo circolare descritto prima. G può usare altre funzioni iterative. Per esempio una funzione per calcolare l' indice di M in cui inserire il prossimo intero letto ed una funzione per stampare il contenuto di M nell'ordine richiesto (cioè nell'ordine in cui i valori sono stati letti da cin).

2) specificare la PRE ed la POST di G;

3) descrivere l'invariante del principale ciclo di G.

Parte ricorsiva:

1) Realizzare una funzione ricorsiva Gric con prototipo void Gric(int k, int*M, int index, int count) che realizzi l'operazione descritta prima. Il parametro formale M è l'array di k elementi da utilizzare nel modo circolare descritto prima, index è la posizione di M in cui si deve inserire il prossimo intero letto (se diverso da -2) e count conta quanti interi (diversi da -2) sono stati letti, in modo da distinguere tra i casi (i) ed (ii). Gric può usare altre funzioni simili a quelle suggerite per la parte iterativa, ma ricorsive (qualora contengano delle iterazioni)

2) Descrivere la PRE e la POST di Gric e dimostrare induttivamente che Gric è corretta rispetto ad esse.