mercoledì 8/2
l compitino

Turno 1

Teoria) Data la dichiarazione double X[4][10][10][20];

1) Specificare il tipo di **(*(X-2) +2) e le dimensioni dell'oggetto puntato da X;

Sia L il valore di X. Si osservi che L è semplicemente un intero (che è anche l'indirizzo RAM del primo byte del primo elemento di X).

2) Usando L, specificare il valore dell'espressione (*X-2)+3. Specificare anche il tipo di (*X-2)+3 e le dimensioni dell'oggetto puntato da (*X-2)+3.

1)essendoci tre stelle, di **(*(X-2) +2) ha tipo double* quindi punta ad oggetto di dimensioni 8 byte
2) (*X-2)+3 è lo stesso di *X+1. *X ha tipo double (*)[10][20] e quindi punta ad un oggetto di dimensioni (10*20*8) byte e quindi *X+1 ha valore L+ (10*20*8)
*X+1 ha lo stesso tipo di *X.

Programmazione) Scrivere un programma costituito da un main che includa la seguente dichiarazione: int C[100], B[100]. Senza occuparsi di mettere valori dentro C, il main deve riempire B in modo tale che alla fine del programma valga la seguente POST-condizione: **POST=(B[0..99] contiene solo valori 1 o 0, ∀ valore v ∈C[0..99], esiste un solo b∈[0..99], tale che C[b]=v e B[b]= 1)**. La PRE-condizione è vuota.

Esempio: C=[3,2,3,2,4], B deve diventare B[1,1,0,0,1].

Insomma B individua i valori diversi contenuti in C, mettendo a 0 le ripetizioni.

La pre-condizione è vuota. Specificare come commento, **scritto dopo il programma**, invariante e post-condizione per ogni ciclo. Considerando il ciclo più interno (più annidato) del vostro programma, delineare le 3 parti che costituiscono la prova del ciclo.

Nota: l'inizializzazione di B è importante.

PRE=()

programma da fare

POST=(B[0..99] contiene solo valori 1 o 0, ∀ valore v ∈C[0..99], esiste un solo b∈[0..99], tale che C[b]=v e B[b]= 1).

L'idea è di scorrere C e di mettere 1 nella posizione di un valore di C che viene "visto" per la prima volta e poi di azzerare tutte le successive occorrenze in C di quel valore. E' necessario rendere visibili in B le posizioni in C di valori non ancora visti. Poiché 1 in B indica un valore visto per la prima volta e 0 un valore che è già stato visto. Per un valore non ancora visto possiamo usare qualsiasi valore diverso da 0 e 1. Per esempio -1.

Inizialmente quindi B viene riempito con -1.
Un ciclo scorre C con indice i crescente e se B[i]=-1 deve segnare in B che l'elemento ora è stato "visto", B[i]=1 e andare avanti su C (secondo ciclo) per marcare con 0 le eventuali altre occorrenze di C[i].

conviene introdurre la seguente notazione:
POST(k) = (0<=k<100) &&(POST con k al posto di 99)
osserva che POST(k) è come applicare la ricetta di indicizzazione

R1=(0<=i<=100) && POST(i-1) &&
($\forall$ a$\in$[i..99], B[a]=-1/0 &&
(B[a]=-1 => C[a] !$\in$C[0..i-1]) &&
(B[a]=0 => C[a] $\in$C[0..i-1])
)

quindi

```
for(int i=0; i<100; i++) //R1
{
  qui si deve mettere il giusto valore in B[i] e ci sono vari casi:
-se B[i]=0 => niente da fare
-se B[i]=-1=> prima volta, va trattato lui e le successive occorrenze
-se B[i]=1 => impossibile
}
```

```
for(int i=0; i<100; i++) //R1
{
  if(B[i]==-1)
   {
     B[i]=1; //B=B
     for(int j=i+1; j<100; j++)//R2
        if(C[j]==C[i])
           B[j]=0;
     }//POST2
}
```

R2=($\forall a \in [i+1..j-1]$ if C[a]=C[i] then B[a]=0 else B[a]=$\underline{B}$[a])

POST2=($\forall a \in [i+1..99]$ if C[a]=C[i] then B[a]=0 else B[a]=$\underline{B}$[a])

Ricetta !! e quindi prova caso d'uscita facile. Anche invarianza è facile

Teoria) Data la dichiarazione double Y[6][8][10][20];
1) Specificare il tipo di *(*(Y-2) +2) e le dimensioni dell'oggetto puntato da *(*(Y-2) +2);
Sia L il valore di Y. Si osservi che L è semplicemente un intero (che è anche l'indirizzo RAM del primo byte del primo elemento di Y).
2) Usando L, specificare il valore dell'espressione (*Y-2)+3. Specificare anche il tipo di (*Y-2)+3 e le dimensioni dell'oggetto puntato da (*Y-2)+3.

1) *(*(Y-2) +2) ha tipo double (*)[20] quindi punta a 20*8 bytes
2) (*Y-2)+3 è uguale a *Y+1 che ha valore L+(10*20*8) in quanto *Y punta a 10*20*8 bytes

Programmazione) Scrivere un programma costituito da un main che includa la seguente dichiarazione: int C[100], B[100]. Senza occuparsi di mettere valori dentro C, il main deve riempire B in modo tale che alla fine del programma valga la seguente post-condizione: **POST=(∀a∈[0..99], B[a] >= 0, ∀ valore v ∈C[0..99], esiste un solo b∈[0..99], tale che C[b]=v e B[b]>0 e B[b] è il numero di occorrenze di v in C).**

Esempio: C=[5,4,4,4,5,9], B può essere [2,3,0,0,0,1] (o anche[0,0,3,0,2,1] e ci sono anche altri array corretti rispetto alla POST). In ogni caso, B individua i diversi valori contenuti in C (si trovano nelle posizioni 0,1 e 5 di C) e specifica la loro numerosità: B[0]=2 dice che in C ci sono 2 valori C[0]=5, B[1]=3 che ci sono 3 valori C[1]=4 e B[5]=1 che c'è un solo C[5]=9.

La PRE-condizione è vuota. Specificare come commento, **scritto dopo il programma**, invariante e post-condizione per ogni ciclo. Considerando il ciclo più interno (più annidato) del programma, delineare le 3 parti che costituiscono la prova del ciclo.

Nota: l'inizializzazione di B è importante.

PRE=()

programma da fare

POST=(∀a∈[0..99], B[a] >= 0, ∀ valore v ∈C[0..99], esiste un solo b∈[0..99], tale che C[b]=v e B[b]>0 e B[b] è il numero di occorrenze di v in C).

L'idea è di scorrere C (indice i) e quando si incontra un valore non ancora "visto", si conta quante volte occorre nel seguito di C, sostituendo questo valore in B[i], si deve anche "neutralizzare" le altre occorrenze in C[j] mettendo B[j]=0. Poiché in B usiamo 0 e un valore >0 per indicare valori già "visti", useremo -1 per indicare quelli non ancora "visti". B va quindi inizializzato a -1.

scorriamo C con un ciclo (indice i) e se B[i]=-1, cioè è visto per la prima volta, allora andiamo con un secondo ciclo (indice j) a cercare le successive occorrenze in C e le neutralizziamo mettendo a 0 B[j] e aumentando B[i] di 1 ogni volta.

Introduciamo una notazione:
POST(k)=(0<=k<100) && (POST con k al posto di 99)=($\forall a \in [0..k]$, B[a] >= 0, $\forall$ valore v $\in$C[0..k], esiste un solo b$\in$[0..k], tale che C[b]=v e B[b]>0 e B[b] è il numero di occorrenze di v in C).

R1=(0<=i<=100) && POST(i-1) && ($\forall a \in [i..99]$, B[a]=-1/0 && B[a]=-1 => C[a] !$\in$C[0..i-1] && B[a]=0=>C[a] $\in$C[0..i-1].

```
for(int i=0; i<100; i++) //R1
{
  if(B[i]==-1) //prima volta
   {
     B[i]=1;  //B=B
     for(int j=i+1; j<100;j++) //R2
        if(C[j]==C[i])
            {B[j]=0; B[i]++;}
     //POST2
    }
}//POST1
```

R2=($\forall a \in [i+1..j-1]$ if C[a]=C[i] then B[a]=0 else B[a]=B[a])
&& (B[i] =n. occorrenze di C[i] in C[0..j-1])

POST2=($\forall a \in [i+1..99]$ if C[a]=C[i] then B[a]=0 else
B[a]=B[a]) &&(B[i] =n. occorrenze di C[i] in C[0..99])