mercoledì 15 febbraio 2012

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

main()
{
int A[5]={1,10,20,30,40};
 A[2]=f(A[3],A);
cout<<A[0]<<A[1]<<A[2]<<A[3]<<A[4]<<endl;
}
```

Programmazione) Scrivere una funzione int F(char T[][10][20], char*P, int dim_P, int n_ele) che rispetta la seguente coppia di PRE e POST-condizione.
PRE=(T ha solo i primi n_ele elementi definiti, n_ele>0, char P[dim_P] è definito)
POST=(F restituisce un indice I t.c. lo strato T[I] ha un numero di righe in cui esiste un match con P maggiore o uguale a quello degli altri strati di T (considerando solo gli elementi definiti)).
F può usare anche altre funzioni. Per ognuna specificare la PRE e POST. Scegliere il ciclo principale di F, associargli un invariante e dimostrare che è effettivamente invariante del ciclo.

Idee:

0) Organizzare bene il lavoro tra varie funzioni
1) calcolare un intero per ogni strato, che conta quante righe hanno un match
2) fare attenzione che l'ultimo strato potrebbe essere non pieno
3) serve una funzione che determini se un array mono-dimensionale ha match o no
4) serve una funzione che determini se in un array mono-dimensionale c'è un match che parte dal primo elemento

partiamo da (4)

PRE=(T ha almeno dim_P elementi, P[0..dim_P-1] definito)

bool match(char* T, char*P, int dim_P)

POST=(restituisce ok t.c. (ok sse T[0..dim_P-1] = P[0..dim_P-1]))

```
bool match(char*T, char*P, int dim_P)
{
   bool ok=true;
   for(int i=0; i<dim_P && ok; i++) // R
      if(P[i]!=T[i])  //attenzione
         ok=false;
   return ok;
}
```

R=(0<=i<=dim_p && ok sse T[0..i-1]=P[0..i-1])

POST=(restituisce ok t.c. (ok sse T[0..dim_P-1] = P[0..dim_P-1]))

funzione (3) che determina se c'è almeno un match su un array mono-dimensionale di lim elementi

PRE=(T[0..lim-1] definita, lim>0, P[0..dim_P-1] definita e dim_P>0)

bool matchR(char*T, int lim, char*P, int dim_P)

POST=(restituisce si t.c. si sse esiste un a in [0..lim-1] a partire dal quale T[a..a+dim_P-1] = P[0..dim_P-1])

```
bool matchR(char*T, int lim, char*P, int dim_P)
{
   bool si=false;
   for(int i=0; i<lim-dimP+1 && !si; i++)  //R
      if(match(T+i,P,dim_P))
               si=true;
   return si;
}
```

R=(si sse esiste a in [0..i-1] t.c. T[a..a+dim_P-1] = P[0..dim_P-1])

(1)e (2) trattamento di uno strato: conviene poter variare il numero di righe dello strato in modo che funzioni anche per l'ultimo strato

PRE=(T[0..righe-1] [20] è definito, righe>=0, P[0..dim_P-1] è definito con dim_P>0)

int matchS(char T[][20], int righe, char*P, int dim_P)

POST=(restituisce conta=n.righe in T[0..righe-1] che contengono un match con P[0..dim_P-1])

```
int matchS(char T[][20], int righe, char* P, int dim_P)
{
  int conta=0;
  for(int i=0; i<righe; i++)
    if(matchR(*(T+i), 20, P, dim_P)) //R
        conta++;
  return conta;
}
```

R=(conta = n. righe in T[0..i-1] in cui c'è match con P)

funzione principale che usa tutte le altre:

```
int F(char T[][10][20], char*P, int dim_P, int n_ele)
{
   int nsp=n_ele/200, neu=n_ele%200, best=-1,
        indice=-1;
    for(int i=0; i<nsp; i++) //R1
      {  int x=matchS(T[i], 10, P, dim_P);
         if(x> best)
           {best=x; indice=i;}
       }
// ultimo strato nella prox slide
} R1=(i=0 => best=indice=-1) && (i>0 =>
T[indice] strato migliore tra strati 0..i-1)
```

```
 int nr=neu/20, ne=neu%20;
 int x= matchS(T[nsp],nr, P, dim_P)+
 (matchR(T[nsp][nr], ne, P, dim_P) ? 1 : 0);
  if(x>best)
     indice=nsp;
  return indice;
}
```