Scritto di programmazione 23 giugno 2009

E' indispensabile consegnare un testo leggibile. Evitare di spezzare una funzione su più pagine. In caso serva si può usare il foglio di protocollo aperto a giornale. Elaborati non leggibili non verranno corretti.

Chi copia perde il diritto di accedere al prossimo appello scritto.

Gli esercizi riguardano le liste concatenate ed in concreto si chiede di considerare un tipo struttura L2E (Lista a 2 Entrate) come segue: struct L2E{nodo* I, *F;};

In un valore di tipo L2E, i 2 campi puntano a 2 nodi di una stessa lista concatenata: il campo I (Inizio) punta al primo nodo della lista ed il campo F (Fine) punta all'ultimo nodo della stessa lista. Quindi un valore L2E permettere di accedere direttamente, oltre che all'inizio di una lista, anche alla fine della lista.

Parte Iterativa:

1) Si chiede di scrivere le seguenti funzioni :

-void addI(L2E & X, nodo* n)

-void addF(L2E& X, nodo* n)

che, rispettivamente, aggiungono il nodo n all'inizio ed alla fine della lista gestita da X.

2) Data una lista concatenata L, i cui nodi hanno tipo struct nodo{int info; nodo* next;}; , e dato un array int P[dimP], si chiede di fare match degli elementi di P (in ordine) con i campi info dei nodi di L, scomponendo L in 2 liste: quella dei nodi in cui si è trovato il match e quella degli altri nodi di L. Il seguente esempio dovrebbe chiarire l'enunciato.

Esempio: sia L=6->3->1->5->2 e P=[3,1,7]. La funzione richiesta deve produrre la coppia di liste : (6->5->2, 3->1). Quindi non si richiede che il match di P sia completo.

Visto che la funzione deve restituire una coppia di liste, è conveniente usare il seguente tipo struttura: struct coppiaL{nodo* a,*b;}; La funzione richiesta deve avere il seguente prototipo:

coppiaL F(nodo*L, int*P, int dimP)

F deve essere iterativa e non deve in nessun caso creare nuovi nodi, ma deve distribuire i nodi della lista data L sulle 2 liste da costruire e da restituire con il return. La struttura L2E e la funzione addF dell'esercizio (1) dovrebbero venire usate per semplificare la realizzazione di F. Specificare la pre e la post condizione di F e l'invariante del ciclo che essa contiene.

Parte Ricorsiva:

3) Si tratta di risolvere lo stesso esercizio (2), ma con una funzione ricorsiva F_ric con lo stesso prototipo di prima. Di nuovo F_ric non deve in alcun caso creare nuovi nodi. Specificare pre e post condizione di F_ric.

SOLUZIONE:

```
#include <iostream>
using namespace std;

struct nodo{int info; nodo* next;
  nodo(int a=0, nodo* b=0){info=a; next=b;}
};


struct L2E{nodo* I,*F; L2E(nodo*x,nodo*y){I=x;F=y;}
};

struct coppiaL{nodo* a,*b; coppiaL(nodo*x,nodo*y){a=x;b=y;}
};

// domanda (1)

void addI(L2E&amp; X, nodo*a)
{
  if(!X.I)
    {
      X.I=a;
      X.F=a;
      a-&gt;next=0;
    }
  else
    {
      a-&gt;next=(X.I);
      X.I=a;
    }
}

void addF(L2E&amp; X, nodo*a)
{
  if(!X.I)
    {
      X.I=a;
      X.F=a;
      a-&gt;next=0;
    }
  else
    {
      a-&gt;next=0;
      X.F-&gt;next=a;
      X.F=a;
    }
}


// domanda (2) pattern match con filtraggio iterativo e ricorsivo usando L2E
```

```
coppiaL PM_i(nodo* x, char*P, int dimP)
{
  nodo *y=x,*t=0;
  L2E X,Y;
  int ip=0;
  while(y &amp;&amp; ip &lt; dimP)
    {
      t=y;
      y=y-&gt;next; // importante ricordare y-&gt;next prima di invocare addF che lo mette a 0
      if(P[ip]==t-&gt;info)
            {
              addF(X,t);
              ip++;
            }
      else
            {
              addF(Y,t);

            }
    }
  if(ok)// attenzione a non dimenticare il resto della lista y dopo che il match e' completato
    if(Y.F) // puo' essere che Y sia ancora vuota
      Y.F-&gt;next=y;
    else
      Y.I=y; // ci interessa solo Y.I, vedi la prossima istruzione
  return coppiaL(X.I,Y.I); // invoca costruttore di coppiaL
}


// domanda (3) ricorsiva
coppiaL PM(nodo*x, char*P, dimP)
{
  if(!x)
    return coppiaL(0,0);
  if(!dimp)
    return coppiaL(0,x); //
  if(x-&gt;info==*P);
  {
    coppiaL y=PM(x-&gt;next,P+1,dimP-1);
    x-&gt;next=y.a;
    y.a=x;
    return y;
  }
  else
    {
      coppiaL y=PM(x-&gt;next,P, dimP);
      x-&gt;next=y.b;
      y.b=x;
      return y;
    }
}
```