Esercizi Liste

Esercizio (tde 9-6-2009)

- Si considerino le seguenti definizioni di funzioni

```
void fun(Lista lis1,Lista lis2) {
        if (lis1==NULL)
                return;
        else if (lis1->next != NULL)
                fun(lis1->next,lis2);
        else
                lis1->next=lis2;
}
void foo(Lista lis1, Lista lis2) {
        fun(fun(lis1,lis2),fun(lis1,lis2));
}
```

- Si spieghi brevemente cosa fanno le funzioni fun e foo.

- Errore di compilazione

  void fun

  …

  fun(fun(lis1,lis2), fun(lis1,lis2))

Esercizio (continua)

- Si considerino le seguenti definizioni di funzioni

```
Lista fun(Lista lis1,Lista lis2) {
        if (lis1==NULL)
                return lis1;
        else if (lis1->next != NULL)
                lis1=fun(lis1->next,lis2);
        else
                lis1->next=lis2;
        return lis1;
}
void foo(Lista lis1, Lista lis2) {
        fun(fun(lis1,lis2),fun(lis1,lis2));
}
```

- Si spieghi brevemente cosa fanno le funzioni fun e foo.

- La prima chiamata annidata a fun concatena lis2 in coda a lis1, la seconda concatena lis2 in coda alla nuova lis1, ma visto che i nodi di lis2 sono sempre gli stessi abbiamo una lista che contiene un ciclo (l'ultimo elemento punta a quello che è il primo elemento di lis2)
- La chiamata a fun che contiene le altre non termina

Esercizio (tdeB 24-2-2012)

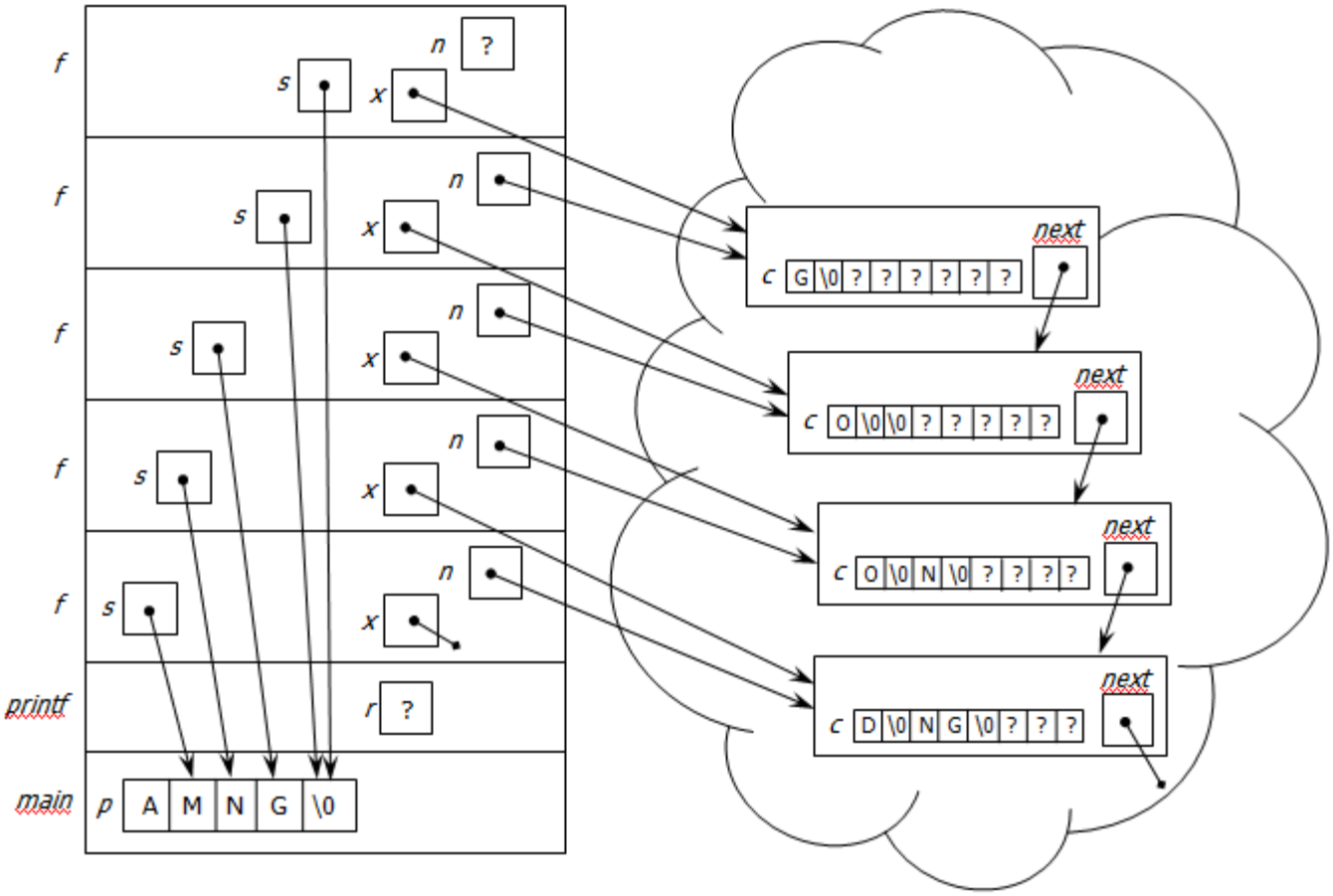
- Si disegnino lo **stack** dei record di attivazione e la memoria allocata dal programma nello **heap** nell'istante in cui la funzione f() inizia a eseguire l'istruzione indicata dalla freccia. Si rappresentino **tutte** le variabili (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi)
- Si calcoli la dimensione in byte della memoria allocata sullo stack (var. statiche e automatiche, tralasciando gli indirizzi di ritorno) e di quella allocata nello heap (sizeof(int)=sizeof(void*)=4)
- Si indichi la linea stampata dal programma sullo standard output

```
typedef struct n { char c[8]; struct n *next; } nodo;

nodo * f( nodo * x, char * s ) {
  nodo * n;
  if( strlen(s) > 0 ) {
    n = (nodo *) malloc(sizeof(nodo));
    n->next = x;
    strcpy( n->c, s++ );
    n->c[1] = '\0';
    *(n->c) += strlen(s);
    return f( n, s );
  }
  return x;
}

void printr( nodo * r ) {
  if( r ) {
    printf("%s", r->c);
    printr( r->next );
  }
}

int main() {
  char p[] = "AMNG" ;
  printr( f( NULL, p ) );
  return 0;
}
```



Heap: 4 x sizeof(nodo) = 4 x (8+4) Byte = 48 Byte
Stack = sizeof(p) + sizeof(r) + 5 x sizeof(r.d.a. di f) = 5 Byte + 4 Byte + 5 x (4+4+4) Byte = 69 Byte

GOOD

Esercizio (tdeB 20-2-2006)

- Sia data una lista concatenata semplice **non ordinata** di interi **senza valori duplicati**. La struttura è:

  typedef struct Elem { int dato;

  struct Elem * next; } Nodo;

  typedef Nodo * Lista;
- La lista rappresenta un insieme di numeri. Una funzione **incrocia()** riceve come **parametri**: la lista, un vettore dinamico di interi (anch'esso senza duplicati, allocato dal programma chiamante) e la lunghezza di tale vettore dinamico. La funzione **rimuove** dalla lista originaria (deallocandoli) tutti i valori contenuti nel vettore (se presenti) e **aggiunge in coda** tutti i valori contenuti nel vettore e non nella lista originaria.
- Si definisca opportunamente il prototipo della funzione **incrocia()** e si descriva sinteticamente (ma in modo preciso) come opera un algoritmo che la implementa. In particolare, si badi a evitare che un valore presente nella lista e non nel vettore sia prima aggiunto e poi rimosso (o viceversa)
- Si codifichi poi in C la funzione secondo l'algoritmo precedentemente dettagliato

La lista deve poter essere modificata, quindi è passata per indirizzo. Il vettore è dinamico, e non deve essere modificato, dunque è sufficiente un puntatore a intero. Il prototipo è pertanto

void incrocia (Lista * lis, int * v, int lun)

Per effettuare correttamente l'incrocio, la funzione può operare come segue:

per ogni valore v[i] del vettore,

effettua una scansione completa della lista, in cui, per ogni *nodo corrente*

- se il nodo corrente della lista contiene v[i], lo dealloca e termina la scansione
- se si è raggiunta la fine della lista, alloca un nuovo nodo col valore v[i]

È imporante ri-scandire *tutta* la lista per ogni elemento del vettore, poiché i valori non sono ordinati. L'assenza di valori duplicati nel vettore garantisce che uno stesso valore non sia prima aggiunto e poi tolto alla lista o viceversa; l'assenza di duplicati nella lista permette di interrompere la scansione.

Occorre anche trattare a parte l'inserimento in testa e la cancellazione nel caso in cui la lista sia vuota:

per ogni valore v[i] del vettore,

se la lista è vuota, alloca un nuovo nodo col valore v[i]

altrimenti, se il primo nodo contiene il valore v[i], dealloca il nodo e termina la scansione

altrimenti, effettua una scansione completa della lista, in cui, per ogni *nodo corrente*

- se il nodo corrente della lista contiene v[i], lo dealloca e termina la scansione
- se si è raggiunta la fine della lista, alloca un nuovo nodo col valore v[i]

Considera il prossimo elemento del vettore

```
void incrocia ( Lista * lis, int * v, int lun ) {
Lista cur, prec; int i, found;
for ( i=0; i<lun; i++) {
  if ( *lis == NULL ) {  /* Se la lista è vuota       */
    *lis = (Lista) malloc( sizeof(Nodo) );
    (*lis)->dato = v[i];   (*lis)->next = NULL;
  }
  else if ( (*lis)->dato == v[i] ) {       /* Se il primo valore è v[i] */
    cur = *lis;     *lis = (*lis)->next;   free(cur);
  }
  else { /* Altrimenti, saltando il primo nodo */
    cur = (*lis)->next;  prec = *lis;    found = 0;
    while ( !found && cur != NULL ) {
      if ( cur->dato == v[i] ) { prec->next = cur->next; free(cur);  found = 1; }
      else { prec = cur; cur = cur->next; }
    }
    if ( !found ) {   /* Ins. in coda, se !trovato */
      prec->next = (Lista) malloc( sizeof(Nodo) );
      prec->next->dato = v[i]; prec->next->next = NULL;
    }
  }
}
return;
}
```

```
Lista InsInFondo( Lista lista, int elem );
int VerificaPresenza(Lista lista, int elem);
Lista Cancella( Lista lista, int elem );
Lista incrocia ( Lista lis, int * v, int lun ) {
    if(lun==0)
        return lis;
    if(VerificaPresenza(lis,v[0]))
        lis=Cancella(lis, v[0]);
    else
        lis=InsInFondo(lis, v[0]);
    if(lun==1)
        return lis;
    lis=incrocia(lis, &v[1],lun-1);
    return lis;
}
```

```
Lista InsInFondo( Lista lista, int elem );
int VerificaPresenza(Lista lista, int elem);
Lista Cancella( Lista lista, int elem );
Lista incrocia ( Lista lis, int * v, int lun ) {
   for(i=0;i<lun;i++)
      if(VerificaPresenza(lis,v[i]))
           lis=Cancella(lis,v[i]);
      else
           lis=InsInFondo(lis,v[i]);

   return lis;
}
```

```
Lista InsInFondo( Lista lista, int elem ) {
  Lista punt, cur = lista;
  punt = (Lista) malloc( sizeof(Nodo) );
  punt–>next = NULL;
  punt–>dato = elem;
  if ( lista==NULL )
       return  punt;
  else {
       while( cur –>next != NULL )
          cur = cur–>next;
       cur –>next = punt;
  }
  return lista;
}

int VerificaPresenza( Lista lista, int elem ) {
   if(lista==NULL)
      return 0;
   if( lista–>dato == elem ) {
         return 1;

   return Trova( lista–>next, elem );
}
```

```
Lista Cancella( Lista lista, int elem ) {
   Lista puntTemp;
   if( lista!=NULL)
      if( lista–>dato == elem ) {
         puntTemp = lista–>next;
         free( lista );
         return puntTemp;
      } else
         lista–>next = Cancella( lista–>next, elem
     );
   return lista;
}
```

Esercizio (tdeB 21-7-2006)

- Il dispositivo di cronometraggio di una gara a tappe genera, all'arrivo di ogni tappa, una lista dinamica con i *numeri di maglia* e i *tempi di tappa* dei vari concorrenti. Un'altra lista conserva la *classifica generale* della gara, con i numeri di maglia e il *tempo totale* di percorrenza. I dati hanno la seguente struttura:
  typedef struct { int ore; int min; int sec; int cent; } Tempo;
  typedef struct Elem { int concorrente; Tempo t_tappa; struct Elem * next; } Arrivo;
  typedef Arrivo * Tappa;
  typedef struct Nodo { int concorrente; Tempo t_totale; struct Nodo *next; } Posizione;
  typedef Posizione * Classifica;
- Il dispositivo di cronometraggio genera la lista **in ordine di arrivo** (e quindi anche in ordine crescente di tempo) aggiungendo in coda nuovi elementi man mano che arrivano i concorrenti.
- Si implementino, dopo averne definito precisamente i prototipi, le funzioni
  … **sum(Tempo t1, Tempo t2, …);** *restituisce il tempo somma t = t1 + t2*
  … **cmp(Tempo t1, Tempo t2, …);** *restituisce 0 se t1=t2, un intero positivo se t1>t2, negativo altrimenti*
- **Suggerimento**: si implementino e utilizzino le due funzioni **long int TempoCent(Tempo t)** e **Tempo CentTempo(long int t)** per convertire un **Tempo** in centesimi di secondo e viceversa.

La funzione sum() può restituire il tempo somma tramite return, così non sono necessari altri parametri.
Per cmp() si può restituire direttamente il valore in millisecondi (con segno) della differenza tra t1 e t2.
Le funzioni di conversione sfruttando troncamento e resto della divisione tra interi, senza difficoltà.

```
long int TempoCent( Tempo t ) {
  long int tmp = t.cent + 100 * t.sec  + (60*100) * t.min + (60*60*100) * t.ore;
  return tmp;
}

Tempo CentTempo( long int t ) {
  Tempo tmp;
  tmp.ore  = t / (60*60*100);     t = t % (60*60*100);
  tmp.min  = t / (60*100);     t = t % (60*100);
  tmp.sec  = t / 100;     tmp.cent = t % 100;
  return tmp;
}

Tempo sum( Tempo t1, Tempo t2 ) {
  return CentTempo( TempoCent(t1) + TempoCent(t2) );
}

long int cmp( Tempo t1, Tempo t2 ) {
  long int tmp1 = TempoCent(t1)
  long int tmp2 = TempoCent(t2);
  return temp1 – temp2;
}
```

typedef struct { int ore; int min; int sec; int cent; } Tempo;

typedef struct Elem { int concorrente; Tempo t_tappa; struct Elem * next; } Arrivo;

typedef Arrivo * Tappa;

typedef struct Nodo { int concorrente; Tempo t_totale; struct Nodo *next; } Posizione;

typedef Posizione * Classifica;

Supponendo di disporre di: (1) le funzioni di ordinamento **Tappa ordTappaM(Tappa t)** e **Classifica ordClassM(Classifica c)**, che ordinano le liste **t** e **c** *in ordine crescente di numero di maglia*; (2) la funzione **Classifica ordClassT(Classifica c)**, che restituisce **c** *ordinata in base al tempo totale crescente* (campo **t_totale**); (3) le funzioni definite al punto (a), si progetti la funzione … **aggiorna(…)** che aggiorna la classifica generale operando come segue:

riceve come parametri la lista con gli arrivi dell'ultima tappa, così come è generata dal dispositivo di cronometraggio, e la lista che rappresenta la classifica generale;

aggiunge al tempo totale di ogni concorrente il tempo conseguito nell'ultima tappa;

elimina dalla classifica generale, **deallocandoli**, gli eventuali concorrenti non arrivati in fondo alla tappa (ritirati), agendo direttamente sulla lista originale (si perdono, quindi, i loro dati);

riordina la classifica generale in base ai nuovi tempi totali.

Si definisca precisamente la funzione **Classifica** aggiorna(Tappa t, Classifica c)

Suggerimento: conviene ordinare le liste in base allo stesso criterio per poi scandirle in parallelo.

Ordina le liste ricevute in ingresso in base al numero di maglia dei concorrenti – così non occorre alcuna funzione di ricerca nella lista per trovare i concorrenti

Gestisce l'eventuale cancellazione del primo nodo della classifica, deallocando i nodi della classifica finché non si raggiunge il primo concorrente arrivato nella tappa.

Scandisce in parallelo le parti restanti delle due liste

- segnalando un errore se è arrivato nella tappa un concorrente non presente in classifica generale
- avanzando in entrambe le liste e sommando i tempi se il numero di maglia corrisponde
- deallocando il concorrente ritirato e avanzando solo nella classifica se

Ordina la classifica generale in base al tempo

```
Classifica aggiorna( Tappa t, Classifica c ) {
  Classifica cur, tmp;
  if(c==NULL  || t==NULL)
    return NULL;
  t  = ordTappaM( t );
  c = ordClassM( c );
  while ( c->concorrente < t->concorrente ) {
    c = cancella(c,c->concorrente);
  }
 cur = c;
 while ( cur != NULL && t != NULL ) {
    if ( cur->concorrente == t->concorrente ) {
      cur->t_totale = sum( cur->t_totale, t->t_totale );
      cur = cur->next;   t = t->next;
    }
    else { /* dealloco il nodo se il conc. non è arrivato  */
      tmp=cur->next; c = cancella(c,cur->concorrente); cur=tmp;
    }
  }
  while ( cur != NULL) {/* dealloco gli eventuali ultimi non arrivati   */
      tmp=cur->next; c = cancella(c,cur->concorrente); cur=tmp;
  }
  c = ordClassT( c );       /* La classifica è riordinata per tempo totale  */
  return c;
}
```

Esercizio (tdeB 18-9-2006)

- Le seguenti dichiarazioni definiscono una struttura dati che rappresenta una lista di *acronimi* (un acronimo è una sigla in cui ogni lettera è l'iniziale di una parola, come **ATM** = **A**zienda **T**rasporti **M**ilanesi).
  typedef char Word[100];

  typedef struct WNode { Word parola; struct WNode * next; } WordNode;

  typedef WordNode * WordList;

  typedef struct Acro { Word sigla; WordList listaparole; } Acronym;

  typedef struct ANode { Acronym acronimo; struct ANode * next; } AcroNode;

  typedef AcroNode * AcroList;

Si dichiari (tramite opportuno prototipo) la funzione … **acronimogiusto(…)** che riceve come parametro una struttura di tipo **Acronym** e restituisce **1** se le iniziali delle parole della lista di parole corrispondono esattamente alle lettere della sigla, **0** altrimenti. Si descriva brevemente (ma in modo preciso) un algoritmo per implementarla e la si codifichi in C.

ATTENZIONE:

Gli acronimi possono essere sbagliati per diversi motivi. Ecco tre esempi di acronimi sbagliati:

AT**M** = Azienda Trasporti **R**omani,

AT**M** = Azienda Tessile,

ATM = Associazione Turistica Milano **M**arittima.

Se la sigla è una stringa vuota o la lista di parole è una lista vuota, l'acronimo è (convenzionalmente) giusto se e solo se anche l'altro componente dell'acronimo è vuoto.

Un possibile algoritmo consiste nello scandire in parallelo in uno stesso ciclo le lettere della sigla e la lista delle parole, confrontando man mano la lettera e l'iniziale della parola corrispondente. Alla prima discrepanza, si restituisce 0. La condizione di permanenza nel ciclo è che né la sigla né la lista siano ancora terminate. Al termine del ciclo occorre verificare che non restino né altre lettere nella sigla né altre parole nella lista (le lunghezze, cioè, devono corrispondersi):

```
int acronimogiusto( Acronym a ) {
  int i = 0, lung = strlen(a.sigla);
  WordList tmp = a.listaparole;
  while ( i < lung && tmp != NULL ) {
    if ( a.sigla[i] != tmp->parola[0] )
      return 0;
    i = i + 1;
    tmp = tmp->next;
  }
  if ( i == lung && tmp == NULL )
     return 1;
  else
     return 0;
}
```

```
int acronimogiusto(Acronym a) {
    return aux(a.sigla,a.listaparole);
}

int aux(char c[], WordList lp){
     if(c[0]=='\0')
          if(lp==NULL)
               return 1;
          else return 0;
     else if(lp!=NULL && lp->parola[0]==c[0])
               return aux(&c[1],lp->next);
          else return 0;
}
```

Si codifichi in C una funzione che effettui la deallocazione di una lista di acronimi, secondo il prototipo:

void freeAcroList(AcroList)

Si badi a **non** deallocare **solamente** gli elementi di tipo AcroNode che compongono la lista di acronimi, **ma anche** le liste di parole contenute in ciascun acronimo. Eventuali funzioni ausiliarie devono essere anch'esse codificate.

```
void freeWordList( WordList list ) {
  if ( list != NULL ) {
    WordList tmp = list->next;
    free( list );    freeWordList( tmp );
  }
  return;
}

void freeAcroList( AcroList list ) {
  if ( list != NULL ) {
    AcroList tmp = list->next;
    freeWordList( (list->acronimo).listaparole );
    free( list );    freeAcroList( tmp );
  }
  return;
}
```

Esercizio (tdeB 19-2-2007)

- Si consideri la seguente definizione:

  typedef struct Elem { char * parola;

  struct Elem * next; } Nodo;

  typedef Nodo * Lista;
- Due parole si dicono *simili* se hanno al più due caratteri diversi. Una catena di parole si dice *compatibile col telefono senza fili* (*cctsf*) se ogni parola è *simile* alle adiacenti.
- La funzione **int simili (char *s1, char *s2);** restituisce 1 se s1 e s2 sono simili, 0 altrimenti.
- Usando la funzione **simili(…)** (senza codificarla), si codifichi in C una funzione **f(…)**, preferibilmente ricorsiva, che riceve come parametro una lista dinamica di parole (secondo la definizione soprastante) e restituisce **1** se la lista rappresenta una catena *cctsf*, **0** altrimenti.

```
int f( Lista a ) {   /* Versione ricorsiva */
  if ( a == NULL || a->next == NULL )    return 1;
  if ( !simili(a->parola, a->next->parola) )    return 0;
  return f( a->next );
}

int f( Lista a ) {  /* Versione iterativa */
  while ( a != NULL && a->next != NULL ) {
    if ( !simili(a->parola, a->next->parola) )    return 0;
    a = a->next;
  }
  return 1;
}
```

```
int f(Lista lis){
    if(lis==NULL || lis->next==NULL)
        return 1;
    if(simili(lis->parola, lis->next->parola)
        return f(lis->next);
    else return 0;
}
```

```
int f( Lista a ) {  /* Versione "provocatoria" */
  return ( a==NULL || a->next==NULL || simili(a->parola, a->next->parola) && f(a->next) );
}
```

Si consideri poi la definizione di una lista di catene di parole (da ogni **NodoTesta** inizia una **Lista**).

```
typedef struct Elem2 { Lista catena;
                       struct Elem2 * next;  } NodoTesta;
typedef NodoTesta * ListaDiListe;
```

Si codifichi (preferibilmente in modo ricorsivo: bonus **+1**) una funzione che riceve una lista di liste così definita e dealloca interamente dalla lista di liste le sequenze di parole che non sono cctsf.

Attenzione: si ipotizzi che nelle catene non ci siano parole allocate staticamente, e si badi a deallocare *tutta* la memoria dinamica. [***3 punti, +1*** se in versione ricorsiva]

```
void deallocaCatena( Lista a ) {
  if ( a != NULL ) {
    deallocaCatena( a->next );
    free(a->parola);          /* Bisogna deallocare anche il vettore dinamico */
    free( a );
  }
}

ListaDiListe sfrondaListaDiListe( ListaDiListe b ) {
  if ( b != NULL ) {
    ListaDiListe tmp = sfrondaListaDiListe( b->next );
    if ( !f( b->catena ) ) {
      deallocaCatena( b->catena );  free( b );  return tmp;
    }
    else b->next = tmp;
  }
  return b;
}
```

Esercizio (tdeB 18-7-2007)

- Si consideri la seguente definizione:
  typedef struct Elem { int x; int y; struct Elem * next; } Punto;
  typedef Punto * Linea;
- Una linea spezzata è rappresentata da una lista di punti.
- Definiamo una spezzata *aperta non degenere* (*AND*) se i suoi punti sono almeno due e tutti diversi tra loro. Consideriamo solo spezzate AND. La *lunghezza* di una spezzata è la somma delle distanze euclidee tra punti consecutivi. Date due spezzate A e B, diciamo che A e B sono *disgiunte* se non hanno punti in comune, che B è una *scorciatoia* di A se A e B hanno gli stessi estremi ma B ha lunghezza minore, che A *contiene* B se B è una sottosequenza di A , e che A *estende* B se A contiene B e hanno l'ultimo punto in comune, e infine definiamo la *concatenazione* di due liste disgiunte A e B come la linea C = A●B ottenuta aggiungendo B in coda ad A. Se A e B non sono disgiunte, A●B = ⊥.

- Si progettino e codifichino opportunamente le funzioni seguenti.

  int scorciatoia(Linea A, Linea B)

  restituisce 1 se A è scorciatoia di B, 0 altrimenti

  int estende(Linea A, Linea B)

  restituisce 1 se A estende B, 0 altrimenti

  Linea concatena(Linea A, Linea B)

  restituisce C=A•B riusando i nodi di A e B.

  float tortuosità(Linea A)

  restituisce il rapporto tra la lunghezza di A e la distanza tra i suoi estremi

- È **indispensabile** definire, codificare e usare altre opportune funzioni di supporto per definire in modo più compatto e leggibile le funzioni richieste.

```
int uguali( Punto a, Punto b ) {
 /* Punti uguali se coordinate uguali */
  return ( (a.x == b.x) && (a.y == b.y) );
}

int numNodi( Linea A ) {
 /* Conteggio ricorsivo standard dei nodi */
  if ( A == NULL )
     return 0;
  return 1 + numNodi(A->next);
}
```

```
int AND( Linea A ) {
  Linea tmp;
  if ( numNodi(A) < 2 ) return 0;
     /* Se non è almeno un segmento, non è AND */
  while( A != NULL ) {        /* Per ogni punto */
    tmp = A->next;
    while(tmp != NULL) { /*se ce n'è un altro uguale */
      if ( uguali(*A, *tmp) ) return 0;
      tmp = tmp->next;
    }
    A = A->next;
  }
  return 1;   /* Se arriviamo in fondo, è AND */
}
```

Definita la funzione AND(),ipotizziamo senza ledere la generalità che le linee passate come parametro alle altre funzioni siano AND, a cura dei rispettivi programmi chiamanti (che possono verificare i parametri prima di procedure all'invocazione). Ciò è coerente con l'indicazione della traccia "consideriamo solo spezzate AND".

```
Punto * last( Linea A ) {
  /* Un puntatore all'ultimo punto   */
 if ( A == NULL || A->next == NULL )
    return A;
 return last(A->next);
}
```

```
float dist( Punto p, Punto q ) {
 return sqrt( (p.x-q.x)*(p.x-q.x)+(p.y-q.y)*(p.y-q.y) );
}

float lung( Linea A ) {
 if ( A == NULL || A->next == NULL )   return 0.0;
 return dist(*A, *(A->next)) + lung(A->next);
}

int lineeUguali( Linea a, Linea b ) {
 if ( a == NULL && b == NULL )  return 1;
 if ( a == NULL || b == NULL )  return 0;
 return uguali(*a,*b)&&lineeUguali(a->next,b->next);
}
```

Passiamo quindi a definire le funzioni richieste:

```
int scorciatoia( Linea A, Linea B ) {
  if ( uguali(*A, *B) && uguali(*last(A), *last(B)) )
     return lung(A) < lung (B);
  return 0;
}

int estende( Linea A, Linea B ) {  /* N.B.: B è AND per ipotesi */
  if ( A == NULL )
     return 0;
  if ( uguali(*A, *B) )
     return lineeUguali(A, B);
  return estende(A->next, B);
}
```

```
Linea concatena( Linea A, Linea B ) {
    (last(A))->next = B;
    return A;
}

float tortuosità( Linea A ) {
    return lung(A) / dist(*A, *last(A));
}
```

```
// NON CHIESTA DAL TDE E NON SERVE PER TDE
int contiene(Linea A, Linea B) {
   while(A!=NULL && !uguali(*A,*B))
        A=A->next;
   if(A==NULL)
      return 0;

   while(A!=NULL && B!=NULL && uguali(*A,*B))
        A=A->next;B=B->next;
   if(B==NULL)
      return 1;
   else return 0;
}
```

Esercizio (tde 27-2-2013)

- Si progetti e codifichi una funzione che riceve in ingresso un intero k e una lista di matrici così definita

```
#define N 100
typedef struct NodeM { int numeri[N][N];
                       struct NodeM * next; } NodoM;
typedef NodoM * ListaM;
```

e restituisce una lista di interi del tipo

```
typedef struct NodeI { int numero;
                       struct NodeI * next; } NodoI;
typedef NodoI * ListaI;
```

contenente il più piccolo intero maggiore di k contenuto in ognuna della matrici (se esiste).

```
int cerca(int **numeri, int k) {
    int cur = k, i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (numeri[i][j] > k) {
                if (cur == k) {
                    cur = numeri[i][j];
                }
                else {
                    if (numeri[i][j] < cur) {
                        cur = numeri[i][j];
                    }
                }
            }
        }
    }

    return cur;
}
```

```
ListaI f(ListaM lista, int k) {
    ListaI head = NULL, tmp;
    ListaM iteratore = lista;
    int i;

    while (iteratore != NULL) {
        i = cerca(iteratore->numeri, k);
        if (i > k) {
            tmp = malloc(sizeof(NodoI));
            if (tmp == NULL) {
                // gestione dell'errore
                return NULL;
            }
            tmp->numero = i;
            tmp->next = head;
            head = tmp;
        }
        iteratore = iteratore->next;
    }

    return head;
}
```

Esercizio (tde 27-2-2013)

- Si considerino due liste concatenate semplici di dati, una contenente dati di studenti e una contenente dati di esami. La struttura della liste è:

```
typedef struct Date { int giorno; int mese; int anno; } Data;
typedef struct Item { int matricola;
                      char * cognome, nome, corsoDiStudi;
                      int numEsamiSuperati;
                      float media;
                      struct Item * next } Studente;
typedef Studente * ListaDiStudenti;
typedef struct Node { int matricola;
                      char * corso;
                      int voto;
                      Data data;
                      struct Node * next; } Esame;
typedef Esame * ListaDiEsami;
```

- La lista di esami è ordinata per data.
- Si codifichi una funzione che, ricevuta in ingresso una lista di studenti e una lista di esami, proceda a assegnare i valori corretti ai campi “numEsameSuperati” e “media” degli studenti in base ai dati contenuti nella lista degli esami

```
void f1(ListaDiStudenti studenti, ListaDiEsami esami) {
    ListaDiStudenti itStudenti = studenti;
    ListaDiEsami itEsami;
    int voto = 0;

    while (itStudenti != NULL) {
        itStudenti->numEsamiSuperati = 0;
        itEsami = esami;
        while (itEsami != NULL) {
            if (itEsami->matricola = itStudenti->matricola && itEsami->voto >= 18) {
                voto += itEsami->voto;
                itStudenti->numEsamiSuperati += 1;
            }
            itEsami = itEsami -> next;
        }
        if (itStudenti->numEsamiSuperati > 0) {
            itStudenti->media = voto / itStudenti->numEsamiSuperati;
        }
        itStudenti = itStudenti -> next;
    }
}
```

Continua (tde 27-2-2013)

- Si codifichi una funzione che, ricevuta in ingresso una lista di studenti e una lista di esami, proceda a eliminare dalle due liste gli studenti che non hanno superato esami dopo il 31-12-2003

```
int cerca(int matricola, ListaDiEsami esami) {
    ListaDiEsami itEsami = esami;

    while (itEsami != NULL) {
        if (itEsami->data.anno > 2003) {
            return 1;
        }
        itEsami = itEsami->next;
    }

    return 0;
}
```

```
void elimina(int matricola, ListaDiEsami *esami) {
    ListaDiEsami prev = NULL, itEsami = *esami, tmp;

    while (itEsami != NULL) {
        if (matricola == itEsami->matricola) {
            tmp = itEsami;
            if (prev == NULL) {
                itEsami = tmp->next;
                esami = &itEsami;
            } else {
                prev->next = tmp->next;
                itEsami = tmp->next;
            }
            free(tmp);
        } else {
            prev = itEsami;
            itEsami = itEsami->next;
        }
    }
}
```

```
void f2(ListaDiStudenti *studenti, ListaDiEsami *esami) {
    ListaDiStudenti itStudenti = *studenti, prev = NULL, tmp;

    while (itStudenti != NULL) {
        if (cerca(itStudenti->matricola, *esami)) {
                elimina(itStudenti->matricola, esami);
                tmp = itStudenti;
                if (prev == NULL) {
                        itStudenti = tmp->next;
                        studenti = &itStudenti;
                }
                else {
                        prev->next = tmp->next;
                        itStudenti = tmp->next;
                }
                free(tmp);
        }
        else {
                prev = itStudenti;
                itStudenti = itStudenti->next;
        }
    }
}
```

Esercizio (tdeB 29-1-2009)

- La funzione … ds(…) calcola la *differenza simmetrica* degli elementi di due liste ordinate in senso crescente e prive di duplicati, restituendola come una *nuova* lista (allocata allo scopo), anch'essa ordinata. La differenza simmetrica è costituita dagli elementi che appartengono a una delle due liste ma non all'altra lista (contiene cioè tutti gli elementi che non sono in comune alle due liste).
- Si usi questa definizione:

  typedef struct N { int valore; struct N * next; } Nodo;

  typedef Nodo * Lista;
- Si codifichi la funzione in C.
- Si consiglia l'uso di funzioni ausiliarie.

Se entrambe le liste sono vuote, C è il risultato finale (C è una lista inizialmente vuota).

Se una delle due liste è vuota:

inserisci *in coda* a C una copia dell'elemento corrente della lista non vuota;

avanti ricorsivamente sulla lista vuota e la coda della lista non vuota.

Se entrambe le liste sono non vuote (possiamo confrontare i valori):

se i valori in testa alle due liste sono uguali,

avanti ricorsivamente sul next di entrambe le liste, senza modificare C

se i valori sono diversi,

inserisci in coda a C una copia dell'elemento con il valore *minore*;

avanti ricorsivamente, avanzando sul next della sola lista che aveva il valore minore

```
Lista insCoda( Lista L, int n) {
 if( L == NULL ) {
   L = (Lista) malloc(sizeof(Nodo));
   L->next = NULL;
   L->valore = n;
 } else
   L->next = insCoda( L->next, n );
 return L;
}
```

```
Lista ds( Lista A, Lista B, Lista C ) {  // Una versione "analitica"
  if( A == NULL && B == NULL )
    return C;
  if( A == NULL ) {
    C = insCoda( C, B->valore );
    return ds(A, B->next, C);
  }
  if( B == NULL ) {
    C = insCoda( C, A->valore );
    return ds(A->next, B, C);
  }
  if( A->valore < B->valore ) {
    C = insCoda( C, A->valore );
    return ds(A->next, B, C);
  }
  if( A->valore > B->valore ) {
    C = insCoda( C, B->valore );
    return ds(A, B->next, C);
  }
  return ds(A->next, B->next, C);
}
```

```
// Una versione equivalente ma meno prolissa
Lista ds( Lista A, Lista B, Lista C ) {
if( A == NULL && B == NULL ) return C;
  if( A == NULL || ( B != NULL && A->valore > B->valore ) ) {
    C = insCoda( C, B->valore );
    return ds(A, B->next, C);
  }
  if( B == NULL || ( A != NULL && A->valore < B->valore ) ) {
    C = insCoda( C, A->valore );
    return ds(A->next, B, C);
  }
  return ds(A->next, B->next, C);
}
```

```
// Soluzione alternativa
Lista creanodo( int val ) {
  Lista tmp = (Lista) malloc(sizeof(Nodo));
  tmp->next = NULL;
  tmp->valore = val;
  return tmp;
}

Lista Copia( Lista L ) {  // Alloca e restituisce una copia di L
  Lista tmp = L;
  if( tmp != NULL )
    tmp = creaNodo( L->valore );
    tmp->next = Copia( L->next );
  }
  return tmp;
}
```

```
Lista ds( Lista A, Lista B ) {
  Lista tmp = NULL;
  if( A == NULL )
    return Copia( B );
  if( B == NULL )
    return Copia( A );
  if( A->valore < B->valore ) {
    tmp = creanodo( A->valore );
    tmp->next = ds( A->next, B );
  }
  else if( A->valore > B->valore ) {
    tmp = creanodo( B->valore );
    tmp->next = ds( A, B->next );
  }
  else
    tmp = ds( A-next, B->next );
  return tmp;
}
```

Si noti che non si effettua inserimento in coda e non si deve quindi scorrere più volte alcuna lista.
Si noti che non è necessario propagare come parametro un riferimento alla lista in costruzione.
Si noti anche come la scomposizione in funzioni chiarisce bene il ruolo delle aprti e aiuta a rendere la soluzione globalmente ben comprensibile.
Si noti infine che le differenze simmetriche restituite dalle chiamate ricorsive alla funzione ds vengono agganciate "in coda" all'elemento di volta in volta costruito dalle singole attivazioni.

//Soluzione ricorsiva

Se entrambe le liste sono vuote,

la differenza simmetrica (d.s.) è vuota.

Se i due valori in testa sono uguali,

tale valore non sarà nella d.s.: non si alloca memoria, ma si propaga il calcolo e se ne restituisce il risultato

In ogni altro caso occorre allocare un nodo, e dunque lo si fa subito:

Se B è vuota oppure il valore in testa a B è maggiore di quello in testa a A

Il nuovo nodo riceve A->valore (che fa parte della d.s.)

L'analisi continuerà su B e la coda di A

altrimenti (cioè se A è vuota oppure il valore in testa a B è minore di quello in testa a A)

Il nuovo nodo riceve B->valore (che fa parte della d.s.)

L'analisi continuerà su A e la coda di B

SI EFFETTUA LA CHIAMATA RICORSVA

La lista restituita da tale chiamata viene attaccata in coda al nuovo elemento, che è il primo della d.s.

Come nella seconda soluzione, il nuovo nodo viene allocato prima della chiamata ricorsiva, ma viene agganciato alla lista solo quando l'esecuzione della chiamata ricorsiva è terminata. In questo modo è possibile eseguire "in testa" l'inserimento che nella prima versione avviene "in coda", sfruttando la "fase discendente" della ricorsione – quella cioè in cui si deallocano i record di attivazione.

```
Lista ds( Lista A, Lista B ) {
  Lista tmp;
  if( A == NULL && B == NULL )
    return NULL;
  if( B != NULL && B != NULL && A->valore == B->valore )
    return ds(A->next, B->next);
  tmp = (Lista) malloc(sizeof(Nodo));
  if ( B == NULL || (A != NULL && A->valore < B->valore) ) {
    tmp->valore = A->valore;
    A = A->next;
  } else {        // qui è garantito che  A == NULL || B != NULL && A->valore > B->valore
    tmp->valore = B->valore;
    B = B->next;
  }
  tmp->next = ds(A, B);
  return tmp;
}
```

Esercizio (tde 1-2-2008)

- Si progetti e codifichi una funzione C che riceve in ingresso un vettore di liste definite

  typedef struct Node { int numero;

  struct Node * next; } Nodo;

  typedef Nodo * Lista;

  e un intero N rappresentante la dimensione del vettore. La funzione deve verificare se esistono numeri presenti in tutte le liste del vettore tranne una e in caso positivo stamparli a video. Si utilizzino opportune funzioni ausiliarie per dividere il problema in sottoproblemi più semplici. La funzione sia così definita:
- **void stampaQuasiComuni(Lista lis[], int N);**

```
int cerca(Lista lis, int n) {
    for(; lis!=NULL;lis=lis->next)
          if(lis->numero==n)
                return 1;
     return 0;
}
```

```
int conta(Lista lis[], int N, int k) {
    int cont=0,i;
     for(i=0;i<N;i++)
         if(cerca(lis[i],k))
               cont++;
     return cont;
}
```

```
void stampaQuasiComuni(Lista lis[], int N) {
   Lista aux=NULL;
   for(aux=lis[0];aux!=NULL; aux=aux->next)
      if(conta(lis,N,aux->numero)==N-1
            && !cerca(aux->next,aux->numero) ) // evita di stampare più volte stesso
                                                // numero
        printf("%d ", aux->numero);

   for(aux=lis[1];aux!=NULL; aux=aux->next)
      if(conta(lis,N,aux->numero)==N-1
            && !cerca(lis[0], aux->numero) // non l'ho già trovato nella prima lista
            && !cerca(aux->next,aux->numero) )
        printf("%d ", aux->numero);
}
```

Esercizio (tde 26-1-2009)

- Si consideri una lista concatenata semplice di dati su alberghi caratterizzati dal loro nome, dalla partita IVA e dal numero di camere doppie e singole. La struttura della lista è:

```
typedef struct Node { int    partitaIVA;
                      char * nome;
                      int    numSingole;
                      int    numDoppie;
                      struct Node * next;  } Nodo;
typedef Nodo * Lista;
```

- Si implementi la funzione di prototipo Lista estraiInOrdine(Lista lis) che riceve una lista non ordinata e restituisce una lista contenente solo gli alberghi con più doppie che singole ordinati per partitaIVA. Si utilizzino opportune funzioni ausiliarie per dividere il problema in sottoproblemi più semplici.
- Si ricorda che la funzione int strcmp(const char *s1, const char *s2) restituisce un intero minore, uguale o maggiore di 0 a seconda che s1 sia (o i primi n caratteri siano) rispettivamente alfabeticamente minore, uguale o maggiore di s2.

```
ListaDiElem InsInOrd(Lista lis, int pi,char * n,int ns, int nd) {
      Lista Punt, PuntCor=lis, PuntPrec=NULL;
      while ( PuntCor != NULL  &&  pi > PuntCor–>partitaIva ) {
              PuntPrec = PuntCor; PuntCor = PuntCor->Prox;
      }
      Punt = malloc(sizeof(Nodo));
      Punt–>partitaIva = pi;
      strcpy(Punt–>nome,n);
      Punt–>numeroSingole = ns; Punt–>numeroDoppie = nd;
      Punt–>Prox = PuntCor;
      if (PuntPrec != NULL ) {   /* Inserimento interno alla lista */
              PuntPrec–>Prox = Punt;
              return lis;
      }
      else return Punt;          /* Inserimento in testa alla lista */
}
Lista f(Lista lis) {
 Lista lisnew=NULL;
 while(lis!=NULL) {
   if(lis->numeroDoppie >lis->numeroSingole)
     lisnew=insInOrd(lisnew,lis->partitaIva,lis->nome,lis->numSingole,lis->numDoppie)
     lis=lis->next;
 }
 return lisnew;
}
```

Esercizio (tde 26-1-2009)

- La funzione di prototipo Lista f(Lista A, Lista B) riceve due liste di interi e restituisce una nuova lista (allocata allo scopo) costituita da tutti gli interi della lista A (nell'ordine in cui appaiono in A) che non sono nella lista B seguiti da tutti gli interi della lista B (nell'ordine in cui appaiono in B) che non sono nella lista A.
- Si noti che le liste A e B possono contenere dei duplicati; di conseguenza anche la lista risultante può contenere duplicati.
- Ad esempio, se A contiene gli interi 1,3,2,5,3,4,5 (in quest'ordine) e B contiene gli interi 4,2,6,6,4 (in quest'ordine) allora la lista risultante conterrà gli interi 1,3,5,3,5,6,6 (in quest'ordine).
- Si usi questa definizione di lista:

  typedef struct N { int valore; struct N * next; } Nodo;

  typedef Nodo * Lista;
- Si dia una codifica ricorsiva della funzione f. Si consiglia e si apprezza il ricorso ad eventuali funzioni ausiliarie.

```
#include <stdio.h>
#include<stdlib.h>

typedef struct N {
        int valore;
        struct N * next; } Nodo;
typedef Nodo * Lista;

Lista eliminaPrimo (Lista Lista, int Elem);
int VerificaPresenza(Lista  lis, int Elem);
Lista eliminaTutti (Lista Lista, int Elem);
Lista f2(Lista A,Lista B, Lista C);
Lista f(Lista A,Lista B);
Lista InsInFondo( Lista lista, int elem );
Lista InsInFondoLista(Lista lis,Lista elem);
void VisualizzaLista(Lista lis);
```

```
int main(){
   Lista lista1 = NULL,lista2=NULL;
   int val;

   printf("\nInserisci valori della prima lista:\n");
   scanf("%d",&val); printf("\n");
   while(val!=-1){
      lista1 = InsInFondo(lista1,val);
      scanf("%d",&val); printf("\n");
   }
   VisualizzaLista(lista1);

   printf("\nInserisci valori della seconda lista:\n");
   scanf("%d",&val); printf("\n");
   while(val!=-1){
      lista2 = InsInFondo(lista2,val);
      scanf("%d",&val); printf("\n");
   }
   VisualizzaLista(lista2);

   printf("\nVisualizza risultato:\n");
   VisualizzaLista(f(lista1,lista2));
   return 1;
}
```

```
Lista f(Lista A, Lista B){
   Lista C = NULL;
   return f2(A,B,C);
}

Lista f2(Lista A,Lista B,Lista C){
   if(A==NULL){
      return B;
   }
   if(B==NULL){
      return InsInFondoLista(A,C);
   }
   if(VerificaPresenza(A,B->valore)==1){
      A=eliminaTutti(A,B->valore);
      B=eliminaTutti(B,B->valore);
   }else{
      C=InsInFondo(C,B->valore);
      B=eliminaPrimo(B,B->valore);
   }
   return f2(A,B,C);
}
```

```
int VerificaPresenza(Lista  lis, int Elem){
   if(lis==NULL)
      return 0;
   if(lis->valore==Elem){
      return 1;
   }
   return VerificaPresenza(lis->next,Elem);
}

Lista eliminaPrimo (Lista lis, int Elem) {
   Lista PuntTemp;
   if (lis!=NULL)
      if (lis->valore==Elem) {
         PuntTemp = lis->next;
         free(lis);
         return PuntTemp;
      } else {
         lis->next=eliminaPrimo(lis->next,Elem);
         return lis;
      } else
         return lis;
}
```

```
Lista eliminaTutti (Lista lis, int Elem) {
   Lista PuntTemp;
   if (lis!=NULL){
      if (lis->valore == Elem ) {
         PuntTemp = lis->next;
         free(lis);
         return eliminaTutti(PuntTemp,Elem);
      } else
         lis->next = eliminaTutti (lis->next, Elem);
      }
   return lis;
}

Lista InsInFondo( Lista lista, int elem ) {
   Lista punt, cur = lista;
   punt=(Lista)malloc(sizeof(Nodo));
   punt->next=NULL;
   punt->valore=elem;                  /* Crea il nuovo nodo */
   if ( lista==NULL )
      return  punt;                    /* => punt è la nuova lista */
   else {
      while(cur->next!=NULL)
         cur=cur->next;
      cur->next=punt;                  /* Aggancio all'ultimo nodo */
   }
   return lista;
}
```

```
Lista InsInFondoLista( Lista lis, Lista elem ) {
   Lista punt = elem, cur = lis;      /* Crea il nuovo nodo */
   if ( lis==NULL )
      return  punt;                   /* => punt è la nuova lista */
   else {
      while(cur->next!=NULL)
         cur=cur->next;
      cur->next=punt;                 /* Aggancio all'ultimo nodo */
   }
   return lis;
}

void VisualizzaLista(Lista lis){
   if(lis==NULL)
      printf(" --|\n");
   else{
      printf("%d --> ",lis->valore);
      VisualizzaLista(lis->next);
   }
}
```

Esercizio (tde 9-6-2009)

- Si consideri la seguente struttura dati:

```
typedef struct DataT { int giorno; int mese; int anno; } Data;
typedef struct InterpreteT { char * cognome;
                             char * nome;
                             char * cittadinanza; } Interprete;
typedef Interprete * ListaInterpreti;
typedef struct ConcertoT { ListaInterpreti Lista,
                           Data data,
                           int prezzoBiglietto,
                           struct Node * next; } Concerto;
typedef Concerto * ListaConcerti;
```

- Si codifichi una funzione che, ricevuta in ingresso una lista di concerti restituisce la somma dei prezzi dei biglietti di tutti i concerti che hanno tra gli interpreti almeno un italiano (un italiano ha il campo Cittadinanza=”ITA”).
- Si consiglia fortemente l’utilizzo di funzioni ausiliarie.

```
int isIta(ListaInterpreti c) {
    if(strcmp("ITA",c->cittadinanza)==0)
        return 1;
    return 0;
}

int hasIta(ListaInterpreti c){
    if(c==NULL)
        return 0;
    if(isIta(c))
        return 1;
    return hasIta(c->next);
}
```

```
int totIta(ListaConcerti lc){
    int totDopo;
    if(lc==NULL)
        return 0;
    totDopo=totIta(lc->next);
    if(hasIta(lc->Lista)
        return totDopo+lc->prezzoBiglietto
    else
        return totDopo;
}
```

Esercizio (tde 1-2-2008)

- Si consideri una lista concatenata semplice di dati su persone caratterizzate dal loro cognome, dal loro nome e dal codice di iscrizione. La struttura della lista è:

  typedef struct Node { int codice;
  char * cognome;
  char * nome;
  struct Node * next; } Nodo;

  typedef Nodo * Lista;

  Si implementi la funzione di prototipo Lista generaListaOrdinata(Lista lis); che riceve una lista non ordinata e restituisce una lista contenente gli stessi elementi ma ordinati per cognome e, a parità di cognome, per nome. Si utilizzino opportune funzioni ausiliarie per dividere il problema in sottoproblemi più semplici.

  Si ricorda che la funzione int strcmp(const char *s1, const char *s2) restituisce un intero minore, uguale o maggiore di 0 a seconda che s1 sia (o i primi n caratteri siano) rispettivamente minore, uguale o maggiore di s2.

Esercizio (tde 2-25-2008)

- Si progetti e codifichi una funzione C che riceve in ingresso una lista definita

  typedef struct Node { int numero;

  struct Node * next; } Nodo;

  typedef Nodo * Lista;
- La funzione deve verificare se l'andamento della lista è ondulatorio, cioè se non capita mai che tre numeri consecutivi siano in ordine crescente o decrescente. La funzione restituisce 1 se l'andamento è ondulatorio, 0 altrimenti. La funzione sia così definita:

  int ondulatoria(Lista lis);
- Una lista è ondulatoria crescente se mai tre numeri consecutivi sono in ordine crescente o decrescente ma sempre il primo e il terzo di tre consecutivi sono in ordine crescente. Definire una funzione che accetta come parametri una lista e restituisce 1 se la condizione è verificata, 0 altrimenti. La funzione sia così definita:

  int ondulatoriaCrescente(Lista lis);

```
int ondulatoria(Lista lista) {
    Lista iteratore = lista,prec1, prec2;
    int contatore = 0;

    while (iteratore != NULL) {
        contatore++;
        if (contatore == 1) {
            prec1 = iteratore;
        } else if (contatore == 2) {
            prec2 = iteratore;
        }
        else {
            if ((prec1->numero > prec2->numero && prec2->numero > iteratore->numero) ||
                (prec1->numero < prec2->numero && prec2->numero < iteratore->numero)) {
                return 0;
             } else {
                prec1 = prec2;
                prec2 = iteratore;
             }
        }
        iteratore = iteratore->next;
    }
    return 1;
}
```

```
int ondulatoriaCrescente(Lista lista) {
  Lista iteratore = lista, prec1, prec2;
  int contatore = 0;

  while (iteratore != NULL) {
     contatore++;
     if (contatore == 1) {
              prec1 = iteratore;
     } else if (contatore == 2) {
              prec2 = iteratore;
     } else if ((prec1->numero > prec2->numero && prec2->numero > iteratore->numero) ||
                (prec1->numero < prec2->numero && prec2->numero < iteratore->numero)) {
              return 0;
     } else if (prec1->numero >= iteratore->numero) {
                                        return 0;
     } else {
              prec1 = prec2;
              prec2 = iteratore;
     }
     iteratore = iteratore->next;
  }
  return 1;
}
```

Esercizio (tde 2-25-2008)

- Definiamo prefisso di lunghezza p di una lista la lista ottenuta prendendo i primi p elementi della lista nello stesso ordine della lista originaria.

  Ad esempio data la lista -> 5 -> 6 -> 9 -> 2 -> 4 -> 1 il prefisso di lunghezza 3 è -> 5 -> 6 -> 9
- Avendo a disposizione il tipo

  typedef struct Node { int numero;

  struct Node * next; } Nodo;

  typedef Nodo * Lista;

  typedef struct NodeList { Lista lis;

  struct NodeList * next; } NodoLista;

  typedef NodoLista * ListaDiListe;
- Definire una funzione **ListaDiListe generaListaPrefissi(Lista lis, int k);** che ricevuta in input una lista e un intero k restituisce la lista delle k liste costruite prendendo i k prefissi di lunghezza compresa tra 1 e k.

```
void freeLista(Lista lista) {
    Lista iteratore = lista, corrente;

    while (iteratore != NULL) {
        corrente = iteratore;
        iteratore = iteratore->next;
        free(corrente);
    }
}
```

```
Lista prefisso(Lista lista, int n) {
    Lista iteratore = lista, pref = NULL, corrente = NULL, coda = NULL;
    int i;

    for (i = 0; i < n && iteratore != NULL; i++, iteratore = iteratore->next) {
            corrente = malloc(sizeof(Nodo));
            if (corrente == NULL) {
                    fprintf(stderr, "Impossibile allocare\n");
                    return pref;
            }
            corrente->numero = iteratore->numero;
            if (coda == NULL) {
                    pref = corrente;
                    coda = corrente;
            } else {
                    coda->next = corrente;
                    coda = corrente;
            }
    }
    if (i < n && iteratore == NULL) {
            freeLista(pref);
            return NULL;
    }
    return pref;
}
```

```
ListaDiListe generaListaPrefissi(Lista lista, int k) {
    int i;
    ListaDiListe risultato = NULL, nuovalista = NULL;

    if (lista == NULL || k <= 0) {
        return NULL;
    }
    for (i = 1; i <= k; i++) {
        nuovalista = malloc(sizeof(NodoLista));
        if (nuovalista == NULL) {
            fprintf(stderr, "Impossibile allocare\n");
            return risultato;
        }
        nuovalista->lis = prefisso(lista, i);
        if (nuovalista->lis == NULL) {
            free(nuovalista);
            return risultato;
        }
        nuovalista->next = risultato;
        risultato = nuovalista;
    }
    return risultato;
}
```

Esercizio (tde 18-2-2009)

- Si progetti e codifichi una funzione C che riceve in ingresso una lista definita

  typedef struct Node { int numero;

  struct Node * next; } Nodo;

  typedef Nodo * Lista;
- La funzione deve verificare se l'andamento della lista è a "crescita lenta", cioè se tutti gli elementi sono in ordine crescente, ma ogni elemento non è più grande del predecessore più di quanto il predecessore fosse più grande del suo predecessore. La funzione restituisce 1 se l'andamento è a "crescita lenta", 0 altrimenti.
- Esempi:

  7 45 67 78 → ok

  7 9 11 13 → ok

  7 45 38 47 → no (38 > 45)

  7 45 234 247 → no (234 rispetto a 45 è cresciuto più di 45 rispetto a 7)

```
int crescitaLenta(Lista lista) {
  Lista iteratore = lista, prec1, prec2;
  int contatore = 0;

  while (iteratore != NULL) {
     contatore++;
     if (contatore == 1) {
              prec1 = iteratore;
     }
     else if (contatore == 2) {
              prec2 = iteratore;
              if (prec2->numero <= prec1->numero) {
                       return 0;
              }
     } else if (!(prec1->numero <= prec2->numero && prec2->numero <= iteratore->numero) ||
                   (prec2->numero - prec1->numero < iteratore->numero - prec2->numero)) {
              return 0;
     } else {
              prec1 = prec2;
              prec2 = iteratore;
     }
     iteratore = iteratore->next;
  }
  return 1;
}
```

Esercizio (tde 18-2-2009)

- Si considerino due liste concatenate semplici di dati, una contenente dati di dipendenti pubblici e il loro Cartellino, una contenente dati di assenze dal lavoro. La struttura della liste è:

  typedef struct Date { int giorno, mese, anno; } Data;

  typedef struct Item { int matricola;

  char * cognome, * nome, * ruolo;

  int bonus;

  struct Item * next } Cartellino;

  typedef Cartellino * ListaDiCartellini;

  typedef struct Node { int matricola; Data data; struct Node * next; } Assenza;

  typedef Assenza * ListaDiAssenze;La lista di assenze è ordinata per data.
- Si supponga di avere a disposizione una funzione (che non deve essere codificata, ma solo usata) che riceve in input due date e restituisce la loro distanza in giorni, definita: **int distanza(Data d1, Data d2);**
- Si ricorda inoltre che la funzione **int strcmp(char *s1, char *s2);** restituisce un intero minore, uguale o maggiore di 0 a seconda che s1 sia rispettivamente minore, uguale o maggiore di s2.
- Si codifichi una funzione che, ricevuta in ingresso una lista di Cartellini e una lista di Assenze, proceda a modificare il campo bonus dei Cartellini aumentando di 1000 il bonus a chi negli ultimi 30 giorni non ha assenze e diminuendolo di 1000 a chi negli ultimi 30 giorni ha più di due assenze. Il bonus non può però in nessun caso diventare negativo.

```
int contaAssenze(int matricola, ListaDiAssenze assenze, Data oggi) {
    ListaDiAssenze iteratore = assenze;
    int contatore = 0;

    while (iteratore != NULL) {
        if (iteratore->matricola == matricola && distanza(oggi, iteratore->data) <= 30) {
                contatore++;
        }
        iteratore = iteratore->next;
    }

    return contatore;
}
```

```
int contaAssenze2(int matricola, ListaDiAssenze assenze, Data oggi) {
        ListaDiAssenze iteratore = assenze;
        int contatore = 0, vecchio = 0;

        while (iteratore != NULL && !vecchio) {
                if (distanza(oggi, iteratore->data) > 30) {
                        vecchio = 1;
                }
                else if (iteratore->matricola == matricola) {
                        contatore++;
                }
                iteratore = iteratore->next;
        }

        return contatore;
}
```

```
void verifica(ListaDiCartellini cartellini, ListaDiAssenze assenze, Data oggi) {
        ListaDiCartellini iteratore = cartellini;
        int numAssenze;

        while (iteratore != NULL) {
                numAssenze = contaAssenze(iteratore->matricola, assenze, oggi);
                if (numAssenze == 0) {
                        iteratore->bonus += 1000;
                }
                else if (numAssenze > 2) {
                        if (iteratore->bonus >= 1000) {
                                iteratore->bonus -= 1000;
                        }
                        else {
                                iteratore->bonus = 0;
                        }
                }
                iteratore = iteratore->next;
        }
}
```

Esercizio (tde 7-9-2009)

- Si progetti e codifichi una funzione C che riceve in ingresso una lista definita

  typedef struct Node { int numero;

  struct Node * next; } Nodo;

  typedef Nodo * Lista;

  che contiene solo valori positivi.
- Definiamo **linziana** una lista se ogni valore pari è seguito da un valore che è esattamente la sua metà.
- Ad esempio la lista

  4 2 1 36 18 9 23 87 34 17 64 32 16 8 4 2 1

  è linziana, mentre la lista

  4 2 36 18 9 23 87 34 17 64 32 16 8 4 2 1

  non lo è perché il 2 è seguito dal 36.
- Si progetti e codifichi una funzione C che riceve in ingresso una lista definita come sopra e restituisce 1 se la lista è linziana, 0 altrimenti.

Esercizio (tde 7-9-2009)

- Si consideri la seguente struttura dati:

```
typedef struct DataT { int giorno; int mese; int anno; } Data;
typedef struct InterpreteT { char * cognome;
                             char * nome;
                             char * cittadinanza;
                             struct InterpreteT * next; } Interprete;
typedef Interprete * ListaInterpreti;
typedef struct ConcertoT { ListaInterpreti Lista,
                           Data data,
                           int prezzoBiglietto,
                           struct Node * next; } Concerto;
typedef Concerto * ListaConcerti;
```

- Si codifichi una funzione che, ricevuta in ingresso una lista di concerti restituisce una lista contenente tutti gli interpreti di concerti del febbraio 2009 con un prezzo inferiore o uguale ai 50 euro. Si consiglia fortemente l'utilizzo di funzioni ausiliarie.

Esercizio (tde 9-2-2010)

- PoliTunes è un software che gestisce musica. PoliTunes gestisce una lista di canzoni, detta Libreria, di dimensione variabile e non limitata. Ogni canzone è caratterizzata dal titolo, dal nome dell'album di appartenenza, dal nome dell'artista, dal nome del compositore e dall'anno di produzione del disco.

  typedef struct canz {

  char titolo[100],album[100],artista[200],compositore[100];

  int anno;

  } Canzone;

  typedef struct ncanz { Canzone c; ncanz * next; } NodoCanzone;

  typedef nodoCanzone * Libreria;

- Si scrivano due funzioni. La prima aggiunge una canzone alla lista.

  Libreria aggiungi(Canzone c, Libreria lib);

- La seconda restituisce una **nuova** lista di canzoni dei "Queen" (artista), scegliendo tutte le canzoni che sono state scritte tra il 1978 e il 1982 da "Roger Taylor" (Compositore). Non viene modificata la lista di partenza.

  Libreria QueenAscoltabili(Libreria lib);

- Si ricorda che la funzione int strcmp(const char *s1, const char *s2) restituisce un intero minore, uguale o maggiore di 0 a seconda che s1 sia (o i primi n caratteri siano) rispettivamente alfabeticamente minore, uguale o maggiore di s2.

Esercizio (tde 25-2-2010)

- Un antivirus è un software che cerca sequenze speciali di byte che sono contenute nei virus. Il database dei virus è quindi una lista di strutture, ogni struttura contiene la sequenza di byte (di lunghezza massima 100), la lunghezza reale delle sequenza e una stringa contenente il nome del virus che contiene la sequenza.

  typedef struct v { char sequenza[100], nome[50]; int lunghezza; struct v * next; } Nodo;

  typedef Nodo * Lista;
- Implementare la seguente funzione che, ricevuta in ingresso una stringa che rappresenta un programma, verifica che non contenga virus:

  void verifica(char programma[] , int lunProg, Lista virusLis);
- Qualora dovesse contenere virus, la funzione stampa a video quali sono i virus da cui è affetto il programma. Se una sequenza è presente più di una volta, la funzione la notifica una sola volta.
- Implementare la seguente funzione che aggiorna il database delle definizioni dei virus. La funzione

  void aggiorna(Lista virusLis, Lista virusScaricati);

  riceve in input la lista dei virus già noti e una lista di virus recenti appena scaricato e verifica che questi virus non siano già catalogati nell'antivirus. Per i virus già presenti nel database corrente, la funzione non deve fare niente, per quelli mancanti, deve aggiungerli in testa al database corrente.

Esercizio (tde 9-7-2010)

- Si consideri la seguente definizione di una lista:

  typedef struct EL { int dato;

  struct EL * next,

  } nodo;

  typedef nodo * lista;

- Implementare una funzione che, ricevuto in ingresso una lista e una matrice NxN di interi (si supponga N una costante predefinita), restituisce 1 se la matrice contiene tutti gli elementi della lista e se in tutte le righe della matrice ci sono almeno due degli elementi contenuti nei nodi della lista (non è necessario controllare se lo stesso elemento è contenuto più di una volta). Si consiglia di implementare funzioni di supporto.

Esercizio (tde 10-9-2010)

- Si consideri la seguente definizione di una lista:

  typedef struct EL { int dato;

  struct EL * next; } nodo;

  typedef nodo * lista;

- Implementare una funzione che, ricevute in ingresso due liste di interi, restituisce la lista che contiene tutti gli elementi presenti in una delle due liste ma non in entrambe. Per semplicità si faccia l'ipotesi che le due liste non contengono duplicati. Si consiglia di implementare funzioni di supporto.

Esercizio (tde 8-2-2011)

- Descrivere il comportamento della funzione m.
- Determinare cosa verrebbe stampato invocando m(12) e m(14560)

```
#define M 16
…
void m(int k) {
    int* h = (int*) malloc(sizeof(int) * M);
    f(h + M - 1, k, 0);
    g(h, M);
    if (h)
                free(h);
}
void f(int * a, int b, int c) {
    if (c == M)
                return;
    c++;
    *(a) = b % 2;
    a--;
    f(a, b/2, c);
}
void g(int* d, int e) {
    if (e) {
                printf("%d ",*d);
        d++;
        g(d, e - 1);
    } else
        printf("\n");
}
```

Esercizio (tde 8-2-2011)

- Un archivio musicale è organizzato nel seguente modo: gli artisti sono disposti in una lista e sono ordinati per nome; ogni artista ha associata una lista di album organizzati per anno e a parità di anno per ordine alfabetico; ogni album consta di una lista di canzoni ordinate per posizione all'interno dell'album.
- Le strutture dati utilizzate sono le seguenti:

  typedef struct Song { char * titolo; int durata, pos; struct Song * next; } Canzone;

  typedef Canzone * ListaCanzoni;

  typedef struct Album { char * titolo; int n_canzoni, anno;
  ListaCanzoni canzoni; struct Album * next; } Disco;

  typedef Disco * ListaDischi;

  typedef struct Singer { char * nome; int n_dischi;
  ListaDischi dischi; struct Singer * next; } Artista;

  typedef Artista * ListaArtisti;
- Si codifichi in C la seguente funzione:

  int discoPiuLungo(ListaArtisti artisti, int anno)

  che restituisce la durata del disco più lungo in un determinato anno

Esercizio (continua)

- Si codifichi in C la seguente funzione:

  int inserisciCanzone(ListaArtisti Lis, char *artista, char *disc,
  int anno, char *canzone, int durata, int posizione)

  che inserisce la canzone qualora l'artista esista e restituisce 0 in caso di esito positivo, -1 in caso negativo. In particolare, occorre aggiungere il disco qualora questo non esista o aggiornare la lista di canzoni del disco qualora esista. Conseguentemente, vanno opportunamente aggiornati i contatori del numero di canzoni e di dischi. Infine, la funzione ha esito negativo qualora esista un album con medesimo titolo e anno diverso, canzone con medesimo titolo all'interno dell'album o un'altra canzone nella medesima posizione.

Esercizio (tde 24-2-2011)

- Una società gestisce la distribuzione di prodotti distribuiti da contadini. I dati della società sono organizzati nel seguente modo: i contadini sono disposti in una lista, ogni contadino ha associata una lista di prodotti.
- Le strutture dati utilizzate sono le seguenti:

```
typedef struct Product { char * nome, categoria;
                         int prezzoAlKg;
                         struct Product * next; } Prodotto;
typedef Prodotto * ListaProdotti;
typedef struct Peasant { char * cognome, nome;
                         ListaProdotti prodotti;
                         struct Peasant * next; } Contadino;
typedef Contadino * ListaContadini;
```

- Si codifichi in C la seguente funzione:

  float mediaPrezziProdotto(ListaContadini contadini, char * prodotto)

  che riceve in input la lista dei contadini e il nome di un prodotto e restituisce la media dei prezzi praticata dai contadini per quel prodotto.

Esercizio (continua)

```
typedef struct Product { char * nome, categoria;
                         int prezzoAlKg;
                         struct Product * next; } Prodotto;
typedef Prodotto * ListaProdotti;
typedef struct Peasant { char * cognome, nome;
                         ListaProdotti prodotti;
                         struct Peasant * next; } Contadino;
typedef Contadino * ListaContadini;
```

- Si codifichi in C la seguente funzione:

  int eliminaProdotto(ListaContadini contadini, char * prodotto)

  che elimina un prodotto dalle liste di tutti i contadini qualora questo esista e restituisce il numero di contadini da cui è stato eliminato. Qualora per un contadino quel prodotto fosse l'unico anche il contadino dev'essere eliminato dalla lista dei contadini.

Esercizio (tde 8-7-2011)

- Una società gestisce una linea di autobus in una grande città. I dati degli autisti e degli autobus sono organizzati in liste. Le strutture dati utilizzate sono le seguenti:

  typedef struct Driver { char * codice, * nome, * patente;
  int annoDiNascita,anniDiEsperienza;
  struct Driver * next; } Autista;

  typedef Autista * Autisti;

  typedef struct Bus { char * targa; int NumeroPosti; struct Bus * next; } Autobus;

  typedef Autobus * Veicoli;

  typedef struct R {char *targaBus, *codiceAutista, tipoCorsa; struct R * next;} Corsa;

  typedef Corsa * Corse;
- Il campo tipoCorsa può assumere i valori 'U' ("Urbano"), 'S' ("Servizio scuole"), 'E' ("Extraurbano"),
- Si codifichi la seguente funzione: int AutistiEsperti(Corse C, Autisti A) che riceve la lista delle corse e degli autisti e restituisce 1 se tutti gli autisti che si occupano di corse "Servizio scuola" hanno almeno 10 anni di esperienza, 0 altrimenti
- Si codifichi in C la seguente funzione: int eliminaCorsa(Corse C, char * targa) che elimina tutte le corse che utilizzano l'autobus con la targa passata come parametro.

Esercizio (tde 12-9-2011)

- Una società gestisce una catena di alberghi. I dati degli alberghi e delle rispettive camere sono organizzati in liste. Le strutture dati utilizzate sono le seguenti:

  typedef struct Room { Albergo * albergo; int numero;
  char doppiaOSingola; /* 'd' se doppia, 's' se singola */
  struct Room * next; } Stanza;

  typedef Stanza * Stanze;

  typedef struct Hotel { char * nome, * indirizzo, * citta; int numSingole, numDoppie;
  Stanze s; /* lista delle stanze dell'hotel */
  struct Hotel * next; } Albergo;

  typedef Albergo * Alberghi;
- Si codifichi in C la seguente funzione: int HotelCapienti(Alberghi A, int N)

  che riceve in input la lista degli alberghi A e l'intero N e restituisce il numero di alberghi della catena che possono ospitare almeno N ospiti
- Si codifichi in C la seguente funzione: int correggiDatiAlberghi(Alberghi A)

  che per ogni albergo verifica se i valori "numSingole" e "numDoppie" sono corretti (rispetto alla lista delle stanze di ogni albergo), corregge quelli errati e restituisce il numero di alberghi per cui è stato necessario effettuare almeno una correzione

Esercizio

- Si consideri la seguente definizione:

```
typedef struct Elem {   char * parola;
                        struct Elem * next; } Nodo;
typedef Nodo * Lista;
typedef struct Elem2 { Lista catena;
                        struct Elem2 * next; } NodoTesta;
typedef NodoTesta * ListaDiListe;
```

- Si scriva una funzione che rimuove dalla ListaDiListe le catene contenenti almeno due parole palindrome.
- È consigliabile definire funzioni ausiliarie.

Esercizio (tdeB 21-9-2007)

- Si consideri la seguente definizione di una lista di liste (catene) di parole:

  typedef struct Elem { char * word; struct Elem * next; } Node;

  typedef Node * List;

  typedef struct Elem2 { List chain; struct Elem2 * next; } HeadNode;

  typedef HeadNode * ListOfLists;

- Si codifichi una funzione **clean(…)** che rimuove (dealloca) da una lista di tipo ListOfLists tutte le catene che contengono parole ripetute. Attenzione: si deallochi tutta la catena che contiene ripetizioni, non solo le parole ripetute, e si deallochi anche il corrispondente HeadNode
- Si consiglia l'uso di opportune funzioni di supporto

Esercizio

- Si progetti e codifichi una funzione C che riceve in ingresso una lista definita

  typedef struct Node { int numero; struct Node * next; } Nodo;

  typedef Nodo * Lista;
- La funzione deve verificare se l'andamento della lista è ondulatorio, cioè se non capita mai che tre numeri consecutivi siano in ordine crescente o decrescente. La funzione restituisce 1 se l'andamento è ondulatorio, 0 altrimenti. La funzione sia così definita:
- **int ondulatoria(Lista lis);**
- Una lista è ondulatoria crescente se mai tre numeri consecutivi sono in ordine crescente o decrescente ma sempre il primo e il terzo di tre consecutivi sono in ordine crescente. Definire una funzione che accetta come parametri una lista e restituisce 1 se la condizione è verificata, 0 altrimenti. La funzione sia così definita:
- **int ondulatoriaCrescente(Lista lis);**

Esercizio

- Definiamo prefisso di lunghezza p di una lista la lista ottenuta prendendo i primi p elementi della lista nello stesso ordine della lista originaria.
- Ad esempio data la lista -> 5 -> 6 -> 9 -> 2 -> 4 -> 1 il prefisso di lunghezza 3 è -> 5 -> 6 -> 9
- Avendo a disposizione il tipo

  typedef struct NodeList { Lista lis;

  struct NodeList * next; } NodoLista;

  typedef NodoLista * ListaDiListe;
- Definire una funzione **ListaDiListe generaListaPrefissi(Lista lis, int k);** che ricevuta in input una lista e un intero k restituisce la lista delle k liste costruite prendendo i k prefissi di lunghezza compresa tra 1 e k.

Esercizio

- Si considerino due liste concatenate semplici di dati, una contenente dati di persone e relative patenti, una contenente dati di multe. La struttura della liste è:
- **typedef struct Date { int giorno; int mese; int anno; } Data;**

```
typedef struct Item { int          numeroDiPatente;
                      char *       cognome;
                      char *       nome;
                      char *       professione;
                      int          punti;
                      struct Item * next } Patente;
typedef Patente * ListaDiPatenti;
typedef struct Node { int          numeroDiPatente;
                      int          puntiTolti;
                      Data         data;
                      struct Node * next; } Multa;
typedef Multa * ListaDiMulte;
```

- La lista di multe è ordinata per data.

Esercizio (tdeB 21-7-2008)

- Una *lista di messaggi con priorità* è una struttura dinamica i cui elementi contengono un messaggio (stringa) e la sua priorità (intero da 1 a 9). I messaggi sono ordinati per priorità e per ordine di arrivo. Tutti i messaggi a pari priorità sono contigui, e a pari priorità troviamo prima quelli giunti da più tempo.
- Definiamo le operazioni di *inserimento* e *rimozione*. Nell'inserimento si specificano testo e priorità, e il nuovo messaggio è posto in coda a quelli che hanno la stessa priorità. La rimozione invece restituisce (e rimuove dalla lista) l'elemento più "vecchio" tra quelli a priorità più alta (o NULL, se la lista è vuota).
- Si definiscano tutte le strutture dati necessarie, si specifichi con precisione come si intendono definiti gli ordinamenti (crescente o decrescente dalla testa alla coda della lista per la priorità e l'anzianità), e si codifichino opportunamente in C le due funzioni **inserisci(...)** e **rimuovi(...)**.

- Si supponga di avere a disposizione una funzione (che non deve essere codificata, ma solo usata) che riceve in input due date e restituisce la loro distanza in giorni, definita: **int distanza(Data d1, Data d2);**
- Si ricorda inoltre che la funzione **int strcmp(char *s1, char *s2);** restituisce un intero minore, uguale o maggiore di 0 a seconda che s1 sia rispettivamente minore, uguale o maggiore di s2.

- Si codifichi una funzione che, ricevuta in ingresso una lista di patenti e una lista di multe, proceda a modificare il campo punti delle patenti dei multati togliendo i punti previsti dalla multa e, a quelle persone che già avevano preso una multa negli ultimi 30 giorni, tolga ulteriori due punti.

Esercizio

- Si consideri la definizione

  typedef struct Node { int numero;

  struct Node * next; } Nodo;

  typedef Nodo * Lista;

- Si codifichi in C la funzione **… spargidivisori(…)**, che riceve come parametri una lista di interi e una matrice NxN di interi, *tutti strettamente positivi*. La funzione deve cercare di copiare ogni valore **v** della lista nella matrice, inserendolo al posto di un valore che sia multiplo di **v**. *Se ci riesce, restituisce 1, e la matrice deve contenere tutti i valori modificati, se non ci riesce, però, oltre a restituire 0, deve lasciare* ***inalterata*** *la matrice*. **Attenzione**: (1) i valori v devono sempre essere confrontati *con la versione iniziale* della matrice, non con le versioni "intermedie" derivanti dalla sostituzione di alcuni valori, (2) se ci sono più multipli di v, se ne può sostituire uno a piacere (il primo che si incontra), (3) si badi a definire chiaramente e/o dichiarare eventuali opportune strutture dati di appoggio o funzioni ausiliarie.

Esercizio (tdeB 25-2-2010)

- Per cercare un valore in una lista concatenata semplice, anche se ordinata, occorre scandirla dall'inizio. Per velocizzare l'accesso, si può affiancare alla lista un vettore ausiliario V che indica dove iniziano e finiscono vari segmenti della struttura. In particolare, la struttura qui sotto rappresenta una lista di parole ordinata alfabeticamente, in cui ogni elemento del vettore ausiliario punta al primo e all'ultimo nodo del segmento di lista relativo alle parole che iniziano con una lettera particolare: la cella V[0] è relativa all'iniziale 'A', v[1] alla 'B', ... v[25] alla 'Z'. Si considerino per semplicità parole di sole lettere maiuscole. Se non vi sono parole con una particolare iniziale, i due puntatori sono posti a NULL.

  typedef struct Nd { char * word; struct Nd * next; } Nodo;

  typedef Nodo * Lista;

  typedef struct Blc { Lista primo, ultimo } Blocco;

  typedef Blocco Vettore[26]

- Si codifichi in C la funzione Lista inserisci(char * p, Lista L, Vettore V) che inserisce una nuova parola p in ordine alfabetico nella struttura dati aggiornando opportunamente il vettore ausiliario e restituendo la lista modificata
- Si implementi in C la funzione ... trasforma(...)che riceve in ingresso la lista e la converte in un albero ordinato con lo stesso criterio, e si confronti qualitativamente l'efficacia delle due strutture (in termini di efficienza misurata in numero medio o massimo di accessi necessari per trovare un particolare elemento)

Esercizio (tde 7-2-2012)

- Un archivio di offerte di lavoro è organizzato nel seguente modo: le offerte sono disposte in una lista e sono ordinate per codice.

  ```
  #define N 100
  typedef struct d { int giorno, mese, anno; } Data;
  typedef struct job { char codice[N], descrizione[N], titoloStudioRichiesto[N];
                       int stipendio;
                       Data dataInserimento;
                       struct job * next; } offertaDiLavoro;
  typedef offertaDiLavoro * ListaOfferte;
  ```

- Si codifichi in C la seguente funzione:

  ```
  int offertaMigliore(ListaOfferte offerte, char titolo[])
  ```

  che restituisce lo stipendio più alto offerto a chi ha un titolo di studio esattamente uguale a quello passato nel parametro titolo

- Si codifichi in C la seguente funzione:

  ```
  int eliminaOfferteVecchie(ListaOfferte Lis, Data oggi)
  ```

  che elimina le offerte di lavoro più vecchie di 90 giorni rispetto alla data passata nel parametro oggi

Esercizio (tde 23-2-2012)

- I dati degli esami degli studenti del Politecnico sono gestiti nel seguente modo:

```
#define N 100
typedef struct D { int giorno,mese,anno; } Data;
typedef struct Es { char codice[N], nome[N];
                    int voto, crediti;
                    Data d;
                    struct Es * next; } Esame;
typedef Esame * ListaEsami
typedef struct St { char nome[N], cognome[N];
                     int Media, totaleCrediti;
                     ListaEsami esami;
                     struct St * next; } Studente;
typedef Studente * ListaStudenti;
```

- Si codifichi in C la seguente funzione *void Calcola(ListaStudenti lis)* che riceve in input la lista degli studenti e aggiorna per ogni studente il valore del campo media con la media pesata degli esami sostenuti e il valore del campo totaleCrediti con la somma dei crediti degli esami sostenuti.
- Si codifichi in C la seguente funzione *ListaStudenti eliminaStudente(ListaStudenti lis)* che riceve in input la lista degli studenti e elimina gli studenti (e i relativi esami) che non hanno superato nessun esame dal 1 gennaio 2007.

Esercizio (tde 6-7-2012)

- Una società organizza serate composte da esibizioni di artisti. I dati della società sono organizzati nel seguente modo: le serate sono raccolte in una lista, ogni serata ha associata una lista di esibizioni.
- Le strutture dati utilizzate sono le seguenti:

```
typedef struct D { int giorno,mese,anno;} Data;
typedef struct Esib { char artista[50];
                      int costoBiglietto;
                      struct Esib * next; } Esibizione;
typedef Esibizione * ListaEsibizioni;

typedef struct Sera { char titolo[50];
                      Data data;
                      ListaEsibizioni esibizioni;
                      struct Sera * next; } Serata;
typedef Serata * ListaSerate;
```

- Si codifichi in C la seguente funzione *int totCostiSerate(ListaSerate serate, Data oggi)* che riceve in input la lista delle serate e una data e restituisce il costo totale di partecipare a tutte le esibizioni più care di tutte le serate in agenda posteriori alla data passata come parametro.
- Si codifichi in C la seguente funzione:

  *ListaSerata eliminaEsibizione(ListaSerata serate, char * artista, Data oggi)*

  che elimina tutte le esibizioni dell'artista passato come parametro da tutte le serate posteriori alla data terzo parametro della funzione. Qualora per una serata l'esibizione cancellata fosse l'unica esibizione anche la serata è eliminata dalla lista delle serate.

Esercizio (tde 5-2-2013)

- Un archivio di spettacoli è organizzato nel seguente modo: gli spettacoli sono contenuti in una lista e sono ordinati per codice. Le strutture dati utilizzate sono le seguenti:

```
#define N 100
typedef struct d { int giorno, mese, anno; } Data;
typedef struct spe { char codice[N], titolo [N], descrizione[N];
                     int costoBiglietto;
                     Data data;
                     struct spe * next; } spettacolo;
typedef spettacolo * ListaSpettacoli;
```

- Si codifichi in C la seguente funzione: int spettacoloMigliore(ListaSpettacoli spettacoli, char titolo[]) che restituisce il costo del biglietto meno caro tra gli spettacoli con titolo esattamente uguale a quello passato nel parametro titolo
- Si codifichi in C la seguente funzione:

```
ListaSpettacoli eliminaSpettacoliVecchi(ListaSpettacoli spettacoli, Data oggi)
```

che elimina gli spettacoli precedenti rispetto alla data passata nel parametro oggi