Esercitazione 3: homework e liste

Irene Finocchi
finocchi@di.uniroma1.it

Homework 1: esercizio 1

Data una stringa s, calcolare una nuova stringa in cui tutte le lettere 'A' maiuscole sono rimpiazzate dalla 'a' minuscola:

```
char *rimpiazzaMaiuscole(char *s);
```

Soluzione con funzione ausiliaria

```
char *rimpiazzaMaiuscole(char *s) {
  char *sNew = (char*) malloc(strlen(s)+1);
  rimpiazzaMaiuscoleRec(s, sNew);
  return sNew;
}

void rimpiazzaMaiuscoleRec(char *s, char *sNew) {
  *sNew= (*s=='A') ? 'a' : *s;
  if (*s!='\0')
      rimpiazzaMaiuscoleRec(s+1, sNew +1);
}
```

Soluzione senza funzione ausiliaria

```
char *rimpiazzaMaiuscole(char *s) {
  char *s1 = (char *) malloc(strlen(s)+1);
  char *s2;
  s1[0]= (*s=='A') ? 'a' : *s;
  if (*s=='\0') return s1;
  s1[1]='\0';
  s2 = rimpiazzaMaiuscole(s+1);
  s1 = strcat(s1,s2);
  if (s2!=NULL) free(s2);
  return s1;
}
```

Homework 1: esercizio 2

Verificare se una stringa s1 è sottostringa di s2:

int sottostringa (char *s1, char *s2);

Una sottostringa è un insieme di caratteri in posizioni consecutive.

Esempio: sottostringa(io, pioggia)=1
sottostringa(io, pippo)=0

® Soluzione con funzione ausiliaria

```
int sottostringa (char *s1, char *s2)
  { return sottostringaRec(s1, s2, 0); }

int sottostringaRec (char *s1, char *s2, int flag) {
  if (*s1=='\0') return 1;
  if (*s2=='\0') return 0;
  if (flag==0) {
      if (*s1==*s2 &&
          sottostringaRec(s1+1,s2+1,1)==1) return 1;
      return sottostringaRec(s1, s2+1, 0);
  }
  if (*s1==*s2) return sottostringaRec(s1+1,s2+1,1);
  else return 0;
}
```

Soluzione senza funzione ausiliaria

```
int sottostringa (char *s1, char *s2) {
  if (*s1=='\0') return 1;
  if (*s2=='\0' || strlen(s2) < strlen(s1)) return 0;
  if (*s1==*s2) {
      int temp;
      char c = s2[strlen(s1)];
      s2[strlen(s1)] = '\0';
      temp = sottostringa(s1+1,s2+1);
      s2[strlen(s1)] = c;
      if (temp==1) return 1;
  }
  return sottostringa(s1,s2+1);
}
```

Esempio:
s1="io"
s2 = "pippo"

Homework 1: esercizio 3

Dati una lista L ed un intero k>0, rimuovere da L un elemento ogni k (ovvero, rimuovere gli elementi in posizione k, 2k, 3k, etc, assumendo che le posizioni siano numerate a partire da 1).

La lista L va modificata. E' ammesso fare deallocazioni, ma non allocazioni di nuova memoria.

void eliminaOgniK(ListPtr *L, int k);

Homework 1: esercizio 3

eliminaOgniK ("irenefinocchi", 1) = ""

eliminaOgniK ("irenefinocchi", 2) = "ieeioci"

eliminaOgniK ("irenefinocchi", 3) = "irneincci"

Il metodo deve usare una sottoprocedura ricorsiva con il seguente prototipo:

void eliminaOgniKRec(ListPtr *L, int h, int k);

La sottoprocedura elimina un elemento ogni k, esclusi i primi h elementi della lista.

® Una possibile soluzione

```
void eliminaOgniK(ListPtr *L, int k) {
  eliminaOgniKRec(L, k-1, k);
}
void eliminaOgniKRec(ListPtr *L, int h, int k) {
  if (*L==NULL) return;
  if (h==0) {
     ListPtr temp=*L;
     *L=(*L)->next;
     free(temp);
     eliminaOgniKRec(L,k-1,k);
  }
  else eliminaOgniKRec(&((*L)->next),h-1,k);
}
```

Esercizio 4 — Intersezione di due liste L1 e L2 non ordinate

Sia x il primo elemento di L1

1) Se x ∈ L2, restituisci x concatenato con l'intersezione tra L1 -> next e L2

2) Se x ∉ L2, restituisci l'intersezione tra L1 -> next e L2

® Esercizio 4 — Intersez. liste non ordinate

```
ListPtr intersezione(ListPtr L1, ListPtr L2) {
  ListPtr L, L3=L2;
  if (L1==NULL || L2==NULL) return NULL;
  while (L3!=NULL && L1->elem!=L3->elem) L3=L3->next;
  if (L3!=NULL) {
      L = malloc(sizeof(struct nodolista));
      if (!L) return NULL;
      L->elem = L1->elem;
      L->next = intersezione(L1->next,L2);
      return L;
  }
  else return intersezione(L1->next,L2);
}
```

Esercizio 5 Intersezione di due liste L1 e L2 ordinate

Sia x il primo elemento di L2
Trova il primo elemento y di L1 ≥ x

1) Se y esiste e y=x, restituisci y concatenato con l'intersezione tra L1 -> next e L2 -> next

2) Se y non esiste o y>x, restituisci l'intersezione tra L1 -> next e L2 -> next

® Esercizio 5 Intersezione di liste ordinate

```
/* Pre:  non ci sono elementi duplicati */
ListPtr intersezioneOrdinate(ListPtr L1, ListPtr L2) {
  if (L1==NULL || L2==NULL) return NULL;
  while (L1!=NULL && L1->elem < L2->elem) L1=L1->next;
  if (L1!=NULL && L1->elem == L2->elem) {
    ListPtr L = malloc(sizeof(struct nodolista));
    if (!L) return NULL;
    L->elem = L1->elem;
    L->next = intersezioneOrdinate(L1->next,L2->next);
    return L;
  }
  else return intersezioneOrdinate(L1,L2->next);
}
```