ESERCIZIO D'ESAME SULLE LISTE

Un file di testo di nome ESPR.TXT contiene, una per riga, stringhe costituite dai caratteri {`a`, `b`, `c`, `*`, `+`}. Ogni riga è lunga al più 20 caratteri. Si realizzi un programma C che:

a) riconosca le stringhe del file uguali a "`a*b+c`" oppure a "`a+b*c`"

b) inserisca i numeri di riga delle stringhe accettate in una lista L1;

c) produca, a partire dalla lista L1 sopra generata, un file di testo di nome UNICHE.TXT in cui ogni stringa accettata compaia una sola volta e sia seguita, sulla stessa riga, dalle sue posizioni originali nel file ESPR.TXT.

ESEMPIO

ESPR.TXT	UNICHE.TXT
a*b+c	a*b+c 1 4
a**b+	a+b*c 5
a*a*c+b+b	
a*b+c	
a+b*c	

Strutture dati

Occorre definire strutture dati per memorizzare:

- la coppia (stringa, posizione) → `element` (in `element.h`)
- il nodo della lista e il tipo lista → `item` e `list` (in `list.h`)

```
typedef struct {
     char stringa[20];  int  pos;
} element;
```

```
typedef struct nodo {
     element  value;   struct nodo *next;
} item;
```

```
typedef item* list;
```

Struttura del programma

All'inizio, la lista L1 è vuota e il contatore di riga vale 0.

Domande a) e b)

Occorre un ciclo di lettura delle stringhe dal file ESPR.TXT. Per ciascuna stringa letta:

- si incrementa il valore del contatore di riga
- si controlla se la stringa letta è una di quelle richieste (Domanda a) e se sì (Domanda b) si inserisce la coppia (*stringa*, *numeroRiga*) nella lista L1.

NB: anche se non richiesto, un inserimento ordinato sul campo *stringa* renderebbe contigue nella lista gli elementi contenenti la medesima *stringa*: ciò potrebbe semplificare le operazioni successive.

Struttura del programma (segue)

Domanda c)

Al termine del ciclo di lettura, si produce dalla lista L1 il file UNICHE.TXT.

NB: se L1 è stata creata come lista *ordinata* sul campo *stringa* basta una sola scansione di L1, altrimenti occorrono tante scansioni di L1 quante le stringhe accettabili.

File utilizzati:

`main.c`	programma principale
`element.h`	file header per l'ADT `element`
`element.c`	implementazione dell'ADT `element`
`list.h`	file header per l'ADT `lista` (ordinata)
`list.c`	implementazione dell'ADT `lista` (ordinata)

Il progetto deve quindi contenere *tutti i file* ***.c*** utilizzati.

ATTENZIONE alle inclusioni multiple!

Sfruttare le direttive al preprocessore per evitare guai!

IL CODICE COMPLETO DELLA SOLUZIONE

MAIN FILE (`main.c`)

```
#include "element.h"
#include "list.h"
#include <stdio.h>
#include <string.h>
#define N 30

main () {
   list  Laux,  L1 = emptyList();
   element el;
   char  s[N];
   int  pos;
   FILE  *f, *g;

   f = fopen("ESPR.TXT", "rt");
   /* DOMANDA a)
      CICLO DI SCANSIONE DEL FILE */
   pos=0;
   while (fgets(s,N,f)) {
      pos++;
      if ( strcmp(s,"a*b+c")==0 ||
           strcmp(s,"a+b*c")==0 ) {
            el = creaEl(s,pos);
            /* DOMANDA  b)
              INSERIMENTO NELLA LISTA */
            L1 = insord(el,L1);
      }
   }

   fclose(f);

      /* ---- segue retro ---- */
```

IL CODICE COMPLETO DELLA SOLUZIONE (II)

MAIN FILE (`main.c`)

```
      /* DOMANDA  c)  */

   g = fopen("UNICHE.TXT", "wt");

   Laux=L1;
   while (!empty(L1)) {
     fprintf(g,"%s",head(L1).stringa);
     fprintf(g,"\t%d", head(L1).pos);
     Laux = tail(L1);
     while (!empty(Laux)  &&
            strcmp(head(Laux).stringa,
                   head(L1).stringa)==0) {
        fprintf(g,"\t%d",Laux->value.pos);
        Laux=tail(Laux);
     }
     fprintf(g,"\n");
     L1=Laux;
   }
   fclose(g);
}
```


IL CODICE COMPLETO DELLA SOLUZIONE (III)

HEADER FILE DEL TIPO `element` (`element.h`)

```
#ifndef ELEMENT_H
#define ELEMENT_H

typedef  struct {
      char stringa[20];  int pos;
} element;

boolean isEqual(element, element);
boolean isLess(element, element);
element creaEl (char s[], int);
void    printElement(element);

#endif
```

FILE SORGENTE DEL TIPO `element` (`element.c`)

```
#include "element.h"
#include <stdio.h>
#include <string.h>

boolean isEqual(element e1,element e2) {
  return (strcmp(e1.stringa, e2.stringa)==0); }

boolean isLess(element e1, element e2) {
  return (strcmp(e1.stringa, e2.stringa)<0); }

element creaEl (char s[], int  p) {
  element  el;
  strcpy(el.stringa, s);  el.pos = p;
  return el;
}

void printElement(element el){
      printf("%s\t%d", el.stringa, el.pos);
}
```


IL CODICE COMPLETO DELLA SOLUZIONE (IV)

HEADER FILE DEL TIPO `list` (`list.h`)

```
#ifndef LIST_H
#define LIST_H

#include "element.h"

typedef struct list_element {
      element  value;
      struct list_element  *next;
} item;

typedef  item* list;
      /* OPERAZIONI PRIMITIVE */
list emptyList(void);
boolean empty(list);
element head(list);
list tail(list);
list cons(element, list);
      /* OPERAZIONI NON PRIMITIVE */
void showList(list);
boolean memberIt(element, list);
boolean member(element, list);
int lengthIt(list);
int length(list);
list append(list, list);
list reverse(list);
list naiveReverse(list);
list copy(list);
list delete(element, list);
list insord(element, list);
list insordIt(element, list);
list inputOrdList(int);

#endif
```


IL CODICE COMPLETO DELLA SOLUZIONE (V)

FILE SORGENTE DEL TIPO `list` (`list.c`)

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

/* Prototipi funzioni ausiliarie */
list nRev2(list, list);
      /* OPERAZIONI PRIMITIVE */
list  emptyList(void) { return NULL; }

boolean  empty(list l) { return (l==NULL); }

element  head(list l) {
  if (empty(l)) abort(); else return l->value;
}

list  tail(list l) {
  if (empty(l)) abort(); else return l->next;
}

list  cons(element e, list l){
  list t;
  t=(list)malloc(sizeof(item));
  t->value=e;  t->next=l;
  return t;
}
      /* OPERAZIONI NON PRIMITIVE */
void showList(list l) {
  printf("[");
  while (!empty(l)) {
      showEl(head(l));  l=tail(l);
      if (!empty(l)) printf(", ");
  }
 printf("]\n");
}
```

IL CODICE COMPLETO DELLA SOLUZIONE (VI)

FILE SORGENTE DEL TIPO `list` (`list.c`) (segue)

```
boolean memberIt(element el, list l) {
  while (!empty(l)) {
      if (isEqual(el,head(l))) return true;
      else  l=tail(l);
  }
  return false;
}

boolean  member(element el, list l) {
  if (empty(l)) return false;
  else
  if (isEqual(el,head(l))) return true;
  else return member(el,tail(l));
}

int lengthIt(list l) {
  int n=0;
  while (!empty(l)) { n++; l=tail(l); }
  return n;
}

int length(list l) {
  if (empty(l)) return 0;
  else return (1 + length(tail(l)));
}

list append(list l1, list l2) {
  if (empty(l1)) return l2;
  else return cons(head(l1), append(tail(l1),l2));
}

list reverse(list l) {
  if (empty(l)) return emptyList();
  else return append(reverse(tail(l)),
                     cons(head(l),emptyList()));
}
```


IL CODICE COMPLETO DELLA SOLUZIONE (VII)

FILE SORGENTE DEL TIPO `list` (`list.c`) (segue)

```
list naiveReverse(list l) {
      return nRev2(emptyList(),l);
}

/* Funzione ausiliaria */

list nRev2(list l2, list l1) {
 if (empty(l1)) return l2;
 else return nRev2(cons(head(l1),l2), tail(l1));
}

list copy(list l){
  if (empty(l)) return l;
  else return cons(head(l), copy(tail(l)));
}

list delete(element el, list l){
  if (empty(l)) return emptyList();
  else
  if (isEqual(el,head(l))) return tail(l);
  else return cons(head(l), delete(el,tail(l)));
}

list insord_p(element el, list l) {
  list pprec,patt=l,paux;
  boolean trovato=0;
  while ( patt!=NULL && !trovato) {
      if (isLess(el,patt->value)) trovato=1;
      else { pprec= patt;  patt= patt->next; }
  }
  paux = (list) malloc(sizeof(item));
  paux->value = el;   paux->next = patt;
  if (patt==l) return paux;
  else { pprec->next=paux; return l; }
}
```


IL CODICE COMPLETO DELLA SOLUZIONE (VIII)

FILE SORGENTE DEL TIPO `list` (`list.c`) (segue)

```
list insord(element el, list l) {
/* ins. ordinato con possibili duplicazioni */
  if (empty(l)) return cons(el,l);
  else
  if (isLess(el,head(l))) return cons(el,l);
  else
  return cons(head(l),insord(el,tail(l)));
}

list inputOrdList(int n) {
/* creazione di una lista ordinata da input */
  element e;
  if (n<0) abort();
  else if (n==0) return emptyList();
  else {
         getElement(&e);
         return insord(e,inputOrdList(n-1));
  }
}
```