Nome File : Liste.h	Funzioni dipendenti dal tipo definite al momento dell'uso

```
typedef char* Elem;
typedef struct nodo* ListeP;

/* premettiamo le funzioni che dipendono dal tipo e il cui
codice sarà introdotto nel file di utilizzo della lista*/

int confr(Elem x, Elem y);
/* postc: restituisce un valore  <0 se x precede y, 0 se  x=y e un valore
>0 altrimenti*/

Elem allCpy(Elem x);
/*postc:restituisce una copia di x, allocandone la memoria*/

void libera(Elem x);
/*libera la memoria allocata per x*/

ListeP costrLista();
 /*postc: restituisce un puntatore di tipo ListeP inizializzato a NULL */

void distrLista(ListeP L);
/*  svuota la lista
/*postc: libera la memoria impegnata da L */
```

Nome File : Liste.h	Funzioni dipendenti dal tipo definite al momento dell'uso

```
int cerca(ListeP L, Elem el);
/* verifica se l'elemento el occorre in L
 *postc: restituisce 1 se el è presente, 0 altrimenti o se la lista e' vuota */

int addElem(ListeP *L, Elem  el );
/* Aggiunge l'elemento el, allocandone la memoria, alla lista *L
*prec: L!= NULL;
postc: restituisce 1 se l'inserimento in testa ha avuto succcesso, oppure 0
se l'elemento è già presente e -1 se non c'è    memoria sufficiente */

int remEl(ListeP *L, Elem el);
/*rimuove el dalla lista  *L
* prec: L!=NULL;
 *postc: restituisce 1 se la rimozione è avvenuta, 0 altrimenti*/
```

Nome File : Liste.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "Liste.h”
struct nodo {
  Elem elem;
 struct nodo * next };


ListeP costrLista()
/*post: restituisce un puntatore a una lista, NULL se non c'è memoria*/
{ ListeP L = calloc(1,sizeof(struct nodo));
if (!L) printf("memoria esaurita per costrLista\n");
return L;}

void distrLista(ListeP L)
/*  svuota la lista
/*postc:  libera la memoria impegnata da L */
{ListeP appP;
while (L != NULL)
       {appP = L;
        L = L -> next;
        libera(appP->elem);
        free(appP); }
}
```

Nome File : Liste.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
int cerca(ListeP L, Elem el)
/* verifica se l'elemento el occorre in L
 *postc: restituisce 1 se el è presente, 0 altrimenti o se la lista e' vuota */
{if (!L) return 0;
 if (confr( L ->elem,el) == 0) return 1;
 return cerca(L-> next,el);
}

int  addElem(ListeP *L, Elem el )
/* Aggiunge l'elemento el alla lista *L
 *postc: restituisce 1 se l'inserimento in testa ha avuto succcesso
 *0 se già presente e -1 se non c'è memoria sufficiente per un nuovo nodo. */
{ ListeP nuovo;
  assert(L);
  if (cerca(*L,el) return 0;
  nuovo = malloc(sizeof(struct nodo));
  if (!nuovo) {printf("manca memoria per addElem"); return -1;}
 nuovo -> elem = allCpy(el);
 nuovo -> next  = *L;
 *L = nuovo
 return 1;
}
```

Nome File : Liste.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
int  remEl(ListeP *L, Elem el )
/*rimuove el dalla lista  *L
* prec: L!=NULL;
 *postc: restituisce 1 se la rimozione
è avvenuta, 0 altrimenti*/
{ListeP tPtr;
assert(L);
if (!(*L)) return 0;
if (confr((*L) ->elem, el) == 0)
    {tPtr = *L;
     (*L) = (*L)->next;
     libera(tPtr ->elem);
     free(tPtr);
     return 1;
    }
 else
return (remEl(&(*L)->next, el));
}
```

Nome File : TabHash.h	Funzioni dipendenti dal tipo definite al momento dell'uso

```
#include "Liste.h"
typedef ListeP * TabHash;

/* premettiamo la funzione che dipende dal tipo e il cui
codice sarà introdotto nel file di utilizzo della tabella hash*/

unsigned long hash(Elem s);
/* calcola l'indirizzo associato a s
postc: restituisce l'indirizzo nella tabella hash per s */

TabHash costTabHash(unsigned int dim);
/* inizializza e restituisce un puntatore a una tabella hash
postc: se non c'è memoria restituisce NULL*/

void distrTabHash(TabHash t);
/* libera la memoria impegnata per t
postc: restituisce NULL, dopo aver liberato la memoria impegnata per
t*/

int cercaTab(TabHash tab,Elem el);
/* cerca un elemento nella tabella hash
Postc: restituisce 1 se l'elemento è presente, 0 altrimenti*/
```

Nome File : TabHash.h	Funzioni dipendenti dal tipo definite al momento dell'uso

```
int insTab(TabHash tab, Elem s);
/* prec: tab! NULL
postc: restituisce 1 se s è stato inserito s nella tabella hash, 0 altrimenti */

int cancTab(TabHash tab,  Elem s);
/* cancella un elemento  dalla tabella hash
 prec: tab! NULL
postc: restituisce 1 se s era presente nella tabella ed è stato cancellato, 0 altrimenti */
```

Nome File : TabHash.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "TabHash.h"

extern unsigned int DIM_TAB; /* la parola chiave extern informa il compilatore che
                                la definizione di questa variabile si trova altrove*/

TabHash costTabHash(unsigned int dim)
/* restituisce un puntatore a una tabella hash
prec: dim >0
postc: se non c'è memoria restituisce NULL*/
{TabHash t;
assert(dim>0);
t = calloc(dim,sizeof(ListeP));
if (!t) printf("manca memoria per costTabHash di dimensione dim =%d \n",dim);
return t;}
```

Nome File : TabHash.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
void distrTabHash(TabHash t)
/* libera la memoria impegnata per t
postc:  libera la memoria impegnata per t, e pone t a NULL*/
{int i=0;
assert(t);
while (i<DIM_TAB)
        distrLista(t[i]);
free(t);
}

int insTab(TabHash tab, Elem s)
/* prec: tab! NULL
postc: restituisce 1 se s è stato inserito s nella tabella hash, 0 altrimenti */
{assert(tab);
return  addElem(&tab[hash(s)],s);
}
```

Nome File : TabHash.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
int cercaTab(TabHash tab,  Elem s )
/* cerca un elemento  nella tabella hash
prec: tab! NULL
*postc: restituisce 1 se s è presente nella tabella, 0 altrimenti */
{assert(tab);
return cerca(tab[hash(s)],s);
}

int cancTab(TabHash tab,  Elem s)
/* cancella un elemento  dalla tabella hash
 prec: tab! NULL
postc: restituisce 1 se s era presente nella tabella ed è stato cancellato,
0 altrimenti */
 {unsigned long m;
  assert(tab);
  m = hash(s);
  if ( tab[m] != NULL) return remEl(&tab[m],s);}
```

Nome File : TestTabHash.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "TabHash.h"
 #define NUM_EL  1000          /* il numero di elementi che vogliamo inserire in una tabella*/
#define MAX_LUN   20        /*la massima lunghezza delle stringhe */
unsigned int DIM_TAB = 101;
/*DIM_TAB dovrebbe essere un primo.  Una regola empirica suggerisce di prendere un valore
di circa un decimo il numero delle chiavi che si prevede di inserire, così che ciascuna lista
dovrebbe contenere  non più di dieci elementi. Con più chiavi del previsto si ha solo un
tempo  maggiore di ricerca, con meno chiavi si impegnerà più spazio del necessario. */
unsigned long hash(char *s)
/* prec: s !=NULL
postc: restituisce l'indirizzo associato a s*/
{unsigned long hasfVal = 0;
assert(s)
while *s !='\0')
    {hashVal = hashVal * 31 + (*s); s++;}
return hashVal;
}
/*31 è una costante dipendente dall'implementazione  correlata alla dimensione dell'alfabeto
utilizzato, hashVal = l'intero associato al prefisso di s fino a qui considerato */
```

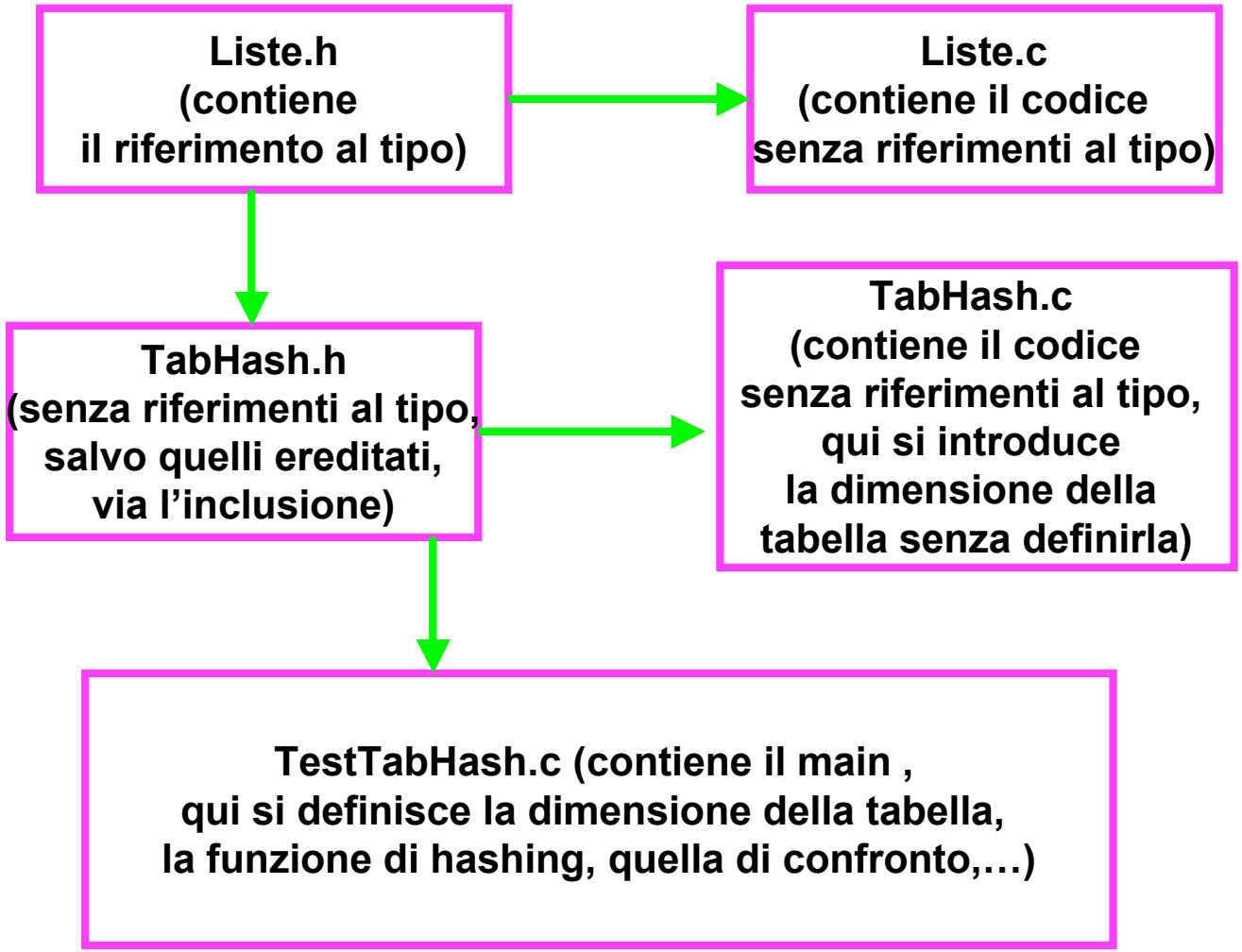
Nome File : TestTabHash.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
 int confr(Elem x,Elem y)
/* postc: restituisce un valore  <0 se x precede y, 0 se  x=y e un valore >0 altrimenti*/
{assert(x);
assert(y);
return strcmp(x,y);}

char* strAllCpy(char* par);

il codice di strAllCpy è omesso.

char* allCpy(char* x) {assert(x);return   strAllCpy(x);}
/*postc:restituisce una copia di x, allocandone opportunamente la memoria*/

void libera(Elem x) {free(x);}
/ *libera la memoria allocata per x*/
```

Nome File : TestTabHash.c	Funzioni dipendenti dal tipo definite al momento dell'uso

```
int main()
{char temp[MAX_LUN];
TabHash tabella;
int i,c;
tabella = costTabHash(DIM_TAB);
if (tabella)
 {printf("inserisci %d stringhe di caratteri, di lunghezza minore di %d \n",NUM_EL,MAX_LUN);
  for (i = 0;i < NUM_EL;i++)
    {printf("inserisci una stringa di caratteri\n");
     printf("?\n");
     scanf("%s",temp);
     c = insTab(tabella,temp);
     if (c==1) printf("inserimento riuscito! \n"); else printf("inserimento non riuscito! \n");
    }
 i = cercaTab(tabella, temp);
 if (i==1) {printf("i = %d,  quindi la stringa  %s e' presente \n",i,temp);
             i = cancTab(tabella, temp);
            if (i==1) printf("i = %d,  quindi la stringa  %s e' stata rimossa \n",i,temp);
           else printf("rimozione fallita!");}
 else printf("la stringa non è presente: errore!");

 }
 else  printf("tabella vuota!");
return 0;}
```

Funzioni dipendenti dal tipo definite al momento dell'uso



Esempio di cambio di tipo e utilizzo della tabella hash

Nel file liste.h si sostituisca la definizione di Elem:

```
typedef char * Elem;
```

con:

```
struct el_lista {
char* nome;
char* def;
};
typedef struct el_lista *
Elem;
```

Nel file testTabHash.c si sostituiscano le definizioni delle funzioni confr, allCpy e libera come segue

```
int confr(Elem x,Elem y) {
assert(x);
assert(y);
return  strcmp(x->nome,y ->nome);}

Elem allCpy(Elem x) {
Elem nuovo;
nuovo = (Elem) malloc(sizeof(struct el_lista));
if (nuovo)
{nuovo ->nome = strAllCpy(x->nome);
nuovo ->def = strAllCpy(x->def);}
return nuovo;}

void libera(Elem x)
{free(x->nome);
free(x->def);
free(x);}
```

Esempio di cambio di tipo e utilizzo della tabella hash

```
int main()
{char temp1[MAX_LUN], temp2[MAX_LUN];
TabHash tabella;
struct el_lista app = {temp1,temp2};
Elem appP = &app;
int i,c;
tabella = costTabHash(DIM_TAB);
if (tabella)
 {printf("inserisci %d coppie di stringhe di caratteri, di lunghezza minore di %d \n",NUM_EL,MAX_LUN
  for (i = 0;i < NUM_EL;i++)
    {printf("inserisci le stringhe di caratteri\n");
     printf("?\n");
     scanf("%s%s", appP ->nome ,appP -> def);
     c = insTab(tabella,appP);
     if (c==1) printf("inserimento riuscito! \n"); else printf("inserimento non riuscito! \n");
    }
 i = cercaTab(tabella, appP);
if (i==1) {printf("i = %d, le stringhe  %s e %s sono  presenti \n",i,appP->nome,appP->def);
            i = cancTab(tabella, appP);
           if (i==1) printf("i = %d, le stringhe  %s e %s sono  presenti \n",i,appP->nome,appP->def);
           else printf("rimozione fallita!");}
else printf("la stringa non è presente: errore!");
 }
else  printf("tabella vuota!");
return 0;}
```

Un'implementazione ad hoc

```
/* nome file : TabHashStringhe.h, versione veloce
*contiene le dichiarazioni di tipo e i prototipi delle funzioni,   */
/*le definizioni delle operazioni di gestione di una tabella hash*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

struct nodo {
char* elem;
struct nodo * next;};

typedef struct nodo * ListeP;
typedef ListeP * TabHash;


 unsigned long hash(char * s);
/*  calcola l'indirizzo associato a s
postc: restituisce l'indirizzo nella tabella hash per s */
```

Un'implementazione ad hoc

```
TabHash costTabHash(int dim);
/* restituisce un puntatore a una tabella hash
prec: dim >0
postc: se non c'è memoria restituisce NULL*/

char* strAllCpy(char* par);
/* alloca memoria e restituisce una copia di par,
prec: par!=NULL && strlen(par)>0
postc: restituisce un puntatore a una copia di par o NULL se non c'e' memoria*/

int cercaTab(TabHash tab,  char *  s );
/* cerca un elemento  nella tabella hash
prec: tab! NULL
*postc: restituisce 1 se s è presente nella tabella, 0 altrimenti */

int insTab(TabHash tab,char *  s);
/* prec: tab! NULL
postc: restituisce 1 se s è stato inserito s nella tabella hash, 0 altrimenti */

int cancTab(TabHash tab,  char * s);
/* cancella un elemento  dalla tabella hash
 prec: tab! NULL
postc: restituisce 1 se s era presente nella tabella ed è stato cancellato, 0 altrimenti */
```

Un'implementazione ad hoc

```
int insTab(TabHash tab, char *  s)
/* prec: tab! NULL
postc: restituisce -1 se non c'è memoria per un nuovo nodo, altrimenti
1 se s è stato inserito s nella tabella hash, 0 altrimenti */
{unsigned long m;
ListeP nuovo;
assert(tab);
m = hash(s);
if (cercaTab( tab,s) ==1) {printf("elemento già presente!");return 0;}
nuovo  = calloc(1,sizeof(struct nodo));
            if (!nuovo)
                {printf("manca memoria per un nuovo nodo\n"); return -1;}
            nuovo -> elem = strAllCpy(s);  /* qui si alloca anche la memoria per la stringa*/
            nuovo -> next = tab[m];
            tab[m] = nuovo;
            return 1;}
```

Un'implementazione ad hoc

```
int cercaTab(TabHash tab,  char *  s )
/* cerca un elemento  nella tabella hash
prec: tab != NULL
*postc: restituisce 1 se s è presente nella tabella, 0 altrimenti */
{unsigned long m;
ListeP l;
assert(tab);
m = hash(s);
for (l=tab[m]; l!=NULL; l=l->next)
     {if (strcmp(s,l->elem)==0) return 1;}
return 0;
}
```

Un'implementazione ad hoc

```
int cancTab(TabHash tab, char *  s)
/* cancella un elemento  dalla tabella hash
 prec: tab! NULL
postc: restituisce 1 se s era presente nella tabella ed è stato cancellato, 0 altrimenti */
{unsigned long m;
ListeP app,temp,prec;
assert(tab);
m = hash(s);
if (!tab[m]) return 0;
if (strcmp(tab[m] ->elem, s) ==0) {tab[m] = tab[m]->next; return 1;}
temp = tab[m] ->next;
prec =tab[m];
while (temp && strcmp(temp ->elem, s) != 0)
 {prec  = temp;
 temp = temp ->next;}
 if (temp)
              {app=temp;
              prec->next = temp->next;
              free(temp ->elem);
              free(temp);
              return 1;}
return 0;}
```