gis.h

```
/* Author Giuseppe Quartarone mat.: 408661
 * Implementazine di un grafo rappresentate un sistema di trasporto pubblico.
 */
#include "list.h"
#include "tree.h"
#include "pqueue.h"
#define INFINITO 65535

/* Definizione di un orario formato da ore e minuti */
typedef struct orario{
   int ore;
   int min;
}orario;

/* Rappresentazione di arco */
typedef struct gis_arch{
   char *name, *src, *dst;     /* nome dell'arco, sorgente, destinazione */
   int keysrc,keydst,keyservice;    /* chiavi della sorgente, della destinazione, del servizio */
   int cost;
   orario orap;                /* ora di partenza in minuti */
   orario orad;                /* ora di arrivo in minuti */
   struct gis_arch *nextfs;         /* puntatore al prossimo arco della stella uscente cui appartiene */
   struct gis_arch *next;      /* puntatore al prozzimo arco della lista */
} service;

/* Descrizione di un nodo */
typedef struct gis_node{
   char *name;
   int key;
   service *day[7];            /* identifica i giorni della settimana, e rappresenta la stella uscente del
nodo */
   struct gis_node *next;      /* puntatore al prossimo nodo nella lista */
} station;

/* Rappresentazione di un grafo */
```

```
typedef struct gis{
   station *stationlist;       /* lista di stazioni */
   service *servicelist;       /* lista di servizi */
   int sizestation;            /* dimensione lista stazioni */
   int sizeservice;            /* dimensione lista servizi */
   node_t *sthead;             /* albero binario contenente i puntatori alle stazioni */
} gis_t;

/* Struttura per rappresentare un predecessore */
typedef struct predecessore{
   station *pred;              /* puntatore alla stazione predecessore */
   station *node;              /* puntatore al nodo stesso */
   service *type;              /* puntatore al servizio che lega il nodo al suo predecessore */
   int etichetta;
} pred_t;

/* Stama l'intero grafo */
void printgis(gis_t *g);

/* Stampa la lista delle tazioni */
void printStationList(gis_t *g);

/* Legge il file di configurazione e crea il grafo */
int readconf(FILE *fp, gis_t *g);

/* Inizzializza il grafo */
gis_t *initializegis();

/* calcola il cammino dalla sorgente alla destinazione */
int cammino(gis_t *gis, int orap, int g, char *src, char *dest);
```

gis.c

```
/* Author Giuseppe Quartarone mat.: 408661
 * Implementazine di un grafo rappresentate un sistema di trasporto pubblico.
 */
```

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "gis.h"

#define MAXNAME 100
#define MAX_ERR_BUFF 100
#define MAX_SOLUZIONI 5
#define TEMPO_CAMBIO  10
#define CHECK(x,y,z,a) if(x != y){ fprintf(stderr,"%s\n",ERROR_3); exit(EXIT_FAILURE);}
#define     ERROR_3  "Errore nel file di configurazione"
#define     ERROR_1     "out of memory on malloc()"
#define     MACRO(x,y,z,t) if( x == y ){ fprintf(stderr,"%s\n",z); return t;}

char errbuff[MAX_ERR_BUFF];

/* Inizializza il gis ai valori di default, restituisce il puntatore alla struttura, NULL altrimenti*/
gis_t *initializegis(){
   gis_t *t;
   int i;
   t=(gis_t *)malloc(sizeof(gis_t));
   MACRO(t,NULL,ERROR_1,NULL)
   t->sizestation=0;
   t->sizeservice=0;
   t->stationlist=NULL;
   t->servicelist=NULL;
   return t;
}

/* Stampa la lista delle stazioni */
void printStationList(gis_t *g){
   station *st;
   st=g->stationlist;
   while(st != NULL){
      printf("%d) %s\n",st->key,st->name);
```

```
      st = st->next;
   }
}

/* Stampa tutta la struttura */
void printgis(gis_t *g){
   station *st;
   service *next;
   st=g->stationlist;
   while(st != NULL){
      printf("%d) Stazione: %s\n",st->key,st->name);
      int i;
      printf(" Servizi: \n");
      for(i=0; i<7; i++){
       printf("GIORNO: %d) ",i+1);
       if(st->day[i] != NULL){
          next=st->day[i];
          while(next != NULL){
             printf("[%d] %s %d) %s --> %d) %s P: %02d:%02d A: %02d:%02d COSTO: %d|\n           ",next-
>keyservice,next->name,next->keysrc,next->src,next->keydst,next->dst,next->orap.ore,next->orap.min,next-
>orad.ore,next->orad.min,next->cost);
             next=next->nextfs;
          }
       }
       printf("\n");
       }
      st=st->next;
   }
}
/* Aggiunge una relazione, in particolare aggiunge un nodo ed il suo relativo arco della stella uscente.
 * restituisce il puntatore all'arco, NULL in caso di errore.
 * Param: g gis, tos nome del servizio, day giorni di effettuazione,puntatore all'arco precedente,sorgente
 * orari di partenza e destinazione
 */
service *addRel(gis_t *g, char *tos, char *day, service *prec, char *sorgente, orario orap, orario orad){
   if(g == NULL )
      return 0;
```

```
int i=0;
int c;
station *src;
service *new;
new = NULL;
addStation(g,sorgente, &src);
if(orap.ore >=0 && orap.min >= 0){
   new=(service *)malloc(sizeof(service));
   MACRO(new,NULL,ERROR_1,NULL)
   new->name=(char *)malloc(sizeof(char)*strlen(tos)+1);
   MACRO(new,NULL,ERROR_1,NULL)
   new->name=strcpy(new->name,tos);
   new->src=(char *)malloc(sizeof(char)*strlen(src->name)+1);
   MACRO(new,NULL,ERROR_1,NULL)
   new->src=strcpy(new->src,src->name);
   new->keysrc=src->key;
   new->orap.ore=orap.ore;
   new->orap.min=orap.min;
   new->orad.ore=orad.ore;
   new->orad.min=orad.min;
   new->next=g->servicelist;
   g->servicelist=new;
   g->sizeservice++;
   new->keyservice = g->sizeservice;
   while(day[i] != '\0'){
    c=day[i] -49 ;
    new->nextfs=src->day[c];
    src->day[c]=new;
    i++;
   }
}
if(prec != NULL){
   prec->dst=(char *)malloc(sizeof(char)*strlen(src->name)+1);
   MACRO(prec->dst,NULL,ERROR_1,NULL)
   prec->dst=strcpy(prec->dst,src->name);
   prec->keydst=src->key;
   prec->orad.ore=orad.ore;
```

```
      prec->orad.min=orad.min;
      prec->cost=(((prec->orad.ore*60)+prec->orad.min)-(prec->orap.ore*60)-(prec->orap.min));
      //printf("COSTO: %d\n",prec->cost);
   }
   return new;
}


/* Legge il file di configurazione, creando man mano le stazioni e gli archi necessari, termina con exit in
caso di errore di lettura
 * scrivendo un messaggio di errore sullo standar error, restituisce 1 in cas positivo.
 */
int readconf(FILE *fp, gis_t *g){
   char tos[MAXNAME], day[8], src[MAXNAME], dest[MAXNAME];
   char tmp[MAXNAME], buff[MAXNAME*3];
   orario orap, orad;
   service *prec;
   int i,j;
   j=0;
   while((fgets(buff, sizeof(buff), fp) != NULL)){
      j++;
      if(isalpha(buff[0])){      // letura TOS
      i=sscanf(buff,"%[^|]|%s\n",tos,day);
      day[7]='\0';
      CHECK(2,i,buff,j)
      }else if(buff[0] == '-'){  //lettura SRC
      i=sscanf(buff,"-|%[^|]|%02d:%02d\n",src,&(orap.ore),&(orap.min));
      CHECK(i,3,buff,j)
      orad.ore=-1;
      orad.min=-1;
      prec=addRel(g,tos,day,NULL,src,orap,orad);
      }else if(buff[0] != '\n'){ //LETTURA DEST
      i=sscanf(buff,"%d:%d|%[^|]|%s\n",&(orad.ore),&(orad.min),dest,tmp);
      CHECK(i,4,buff,j)
      if(isdigit(tmp[0])){
         i=sscanf(tmp,"%02d:%02d",&(orap.ore),&(orap.min));
         CHECK(2,i,buff,j)
```

```
      }
      else{
         orap.ore=-1;
         orap.min=-1;
      }
      if(prec == NULL )
         fprintf(stderr,"%s\n","ERRORE GRAVE IN ADDREL");
      prec=addRel(g,tos,day,prec,dest,orap,orad);
      }
   }
}

/* Aggiunge una stazione, e completa la configurazine di quella precedente rappresentata da s,
 * restituisce -1 in caso di errore, 1 altrimenti.
 * Param: g gis, name nome stazione, s stazione precedente da completare.
 */
int addStation(gis_t *g,char *name, station **s){
   if(g == NULL || name == NULL || name == '\0' || strlen(name)>MAXNAME)
      return -1;
   station *st;
   node_t *node;
   node=NULL;
   st=(station *)malloc(sizeof(station));
   MACRO(st,NULL,ERROR_1,-1)
   addNode(&(g->sthead),name,st,&node);
   if(node == NULL){
      st->name=(char *)malloc(sizeof(char)*strlen(name)+1);
      MACRO(st,NULL,ERROR_1,-1)
      st->name=strcpy(st->name,name);
      st->key=++(g->sizestation);
      st->next=g->stationlist;
      g->stationlist=st;
      *s=st;
      return 1;
   }
   *s=(station *)node->data;
   return 1;
```

```
}


/* Calcola un cammino dalla stazione di partenza fino alla destinazione, restituisce il puntatore all
struttura
 * predecessore.
 * Param: gis gis, g giorno della settimana (1-7), orap orario di partenza in minuti, pred puntatore alla
struttura predecessore,
 * dst stazione di arrivo, minserv puntatore ad un albero binario contenente gli archi da scartare, puntatore
alla coda
 * utilizzata per la visita del grafo.
 */
pred_t *spt(gis_t *gis,int g, int orap, pred_t *pred, station *src,station *dst, node_t *minserv,pqueue *q){
   int partenza,attesa,oldorap;
   node *qnode;
   service *serv,*servtmp;
   station *s;
   node_t *tmp;
   char str[MAXKEYLEN], oldnameserv[MAXNAME];
   oldorap = orap;
   attesa=0;
   partenza=0;
   s = gis->stationlist;
   while(s != NULL){
      pred[s->key-1].etichetta=INFINITO;
      pred[s->key-1].pred=src;
      pred[s->key-1].node=s;
      pred[s->key-1].type=NULL;
      s=s->next;
   }
   pred[src->key-1].etichetta=0;
   pred[src->key-1].pred=NULL;
   initpqueue(&q,5);
   enqueue(q,src->key,pred[src->key-1].etichetta);
   while((qnode=dequeue(q)) != NULL){
      s=pred[qnode->key-1].node;
      serv=s->day[g-1];
```

```
      while(serv != NULL){
       sprintf(str,"%d",serv->keyservice);
       if(((tmp=findNode(minserv,str)) == NULL) ){
          partenza=(serv->orap.ore*60)+serv->orap.min;
          if(partenza >= oldorap){
             if(pred[serv->keysrc-1].type != NULL){
              attesa= partenza - ((pred[serv->keysrc-1].type->orad.ore*60) + pred[serv->keysrc-1].type-
>orad.min);
              strcpy(oldnameserv,pred[serv->keysrc-1].type->name);
              oldorap = 0;
             }
             if(( (attesa >= 0) && (pred[s->key-1].etichetta + serv->cost + attesa) < pred[serv-
>keydst-1].etichetta) ){
              if(!((attesa < TEMPO_CAMBIO) && (pred[serv->keysrc-1].type != NULL) && (strcmp(serv-
>name,oldnameserv) != 0))){
                 pred[serv->keydst-1].etichetta = pred[s->key-1].etichetta + serv->cost + attesa;
                 pred[serv->keydst-1].pred=s;
                 pred[serv->keydst-1].type=serv;
                 enqueue(q,serv->keydst,pred[serv->keydst-1].etichetta);
                 if(serv->keydst == dst->key)
                  return pred;
              }
             }
          }
       }
       serv=serv->nextfs;
      }
   }
   return pred;
}

/* Stampa ricorsivamente il percosro dalla sorgente alla destinazine. Di volta in volta aggiunge
 *  a mincost l'arco usato dal primo nodo per raggiungere la destinazine.
 * Param: p puntatore alla struttura predecessore, i indice della destinazione, key chiave della sorgente,
 * mincost struttura utilizzata per contenere gli archi da scartare nel calcolo delle succesive soluzioni
 */
void stampasol(pred_t *p,int i, int key, node_t **mincost ){
```

```
   service *serv;
   if((p[i].pred != NULL) && (p[i].pred->key != key)){
      stampasol(p,p[i].pred->key-1,key,mincost);
   }
   serv = p[i].type;
   if(p[i].pred->key == key){
      char str[MAXKEYLEN];
      node_t *tmp;
      sprintf(str,"%d",serv->keyservice);
      addNode(mincost,str,serv,&tmp);
   }
   serv = p[i].type;
   printf("%s | %s | %02d:%02d | %s | %02d:%02d\n",serv->name,serv->src,serv->orap.ore,serv->orap.min,serv-
>dst,serv->orad.ore,serv->orad.min);
   return;


}

/* Ricerca un cammino sul grafo e ne esegue la stampa durante l'esecuzione. Si puo` impostare la macro
MAX_SOLUZIONI
 * che intica il numero di riesecuzioni dell'algoritmo per il calcolo di cammini alternativi. Restituisce 1
in caso positivo
 * -1 altrimenti ed in questo caso scrive un messaggio di errore su errbuff.
 * Param: gis gis, orap orario di partenza in minuti, g giono della settimana (1-7), sorgente, destinazione
 */
int cammino(gis_t *gis, int orap, int g, char *sorgente, char *destinazione){
   pqueue *q;
   pred_t *pred;
   node_t *st1,*st2;
   station *src,*dst;
   service *serv, *servtmp;
   node_t *mincost,*tmp;
   int min = INFINITO;
   char *str,keyserv[100];
   int k;
   int j,i,y;
```

```
   j=0; i=0;
   str=" ";
   mincost = NULL;
   printf("Ricerca cammino %d \n",g);
   if(((st1 = findNode(gis->sthead,sorgente)) == NULL) || ((st2 = findNode(gis->sthead,destinazione)) ==
NULL)){
      strcpy(errbuff,"Stazione non trovata\n");
      return -1;
   }
   pred = (pred_t *)malloc(sizeof(pred_t)*gis->sizestation);
   src = (station *)st1->data;
   dst = (station *)st2->data;
   while( j <= MAX_SOLUZIONI ){
      pred = spt(gis,g,orap,pred,src,dst,mincost,q);
      k=dst->key-1;
      if(pred[k].etichetta >= INFINITO){
       printf("fine\n");
       break;
      }
      printf("Opzione %d | %02d:%02d\n",j+1,(pred[k].etichetta/60),(pred[k].etichetta%60));
      stampasol(pred,k,src->key,&mincost);
      printf("\n");
      j++;
   }
   return 1;
}
```

testGis.c

```
/* Author Giuseppe Quartarone mat.: 408661
 */
#include <stdio.h>
#include <stdlib.h>
#include "gis.h"
```

```
#define ABROT_1   "Impossibile aprire il file "
#define ABROT_2   "Errore nella lettura della richiesta "
#define MACRO(x,y,z) if((x) == y){fprintf(stderr,"%s \n",z); exit(EXIT_FAILURE);}

int main(int argc, char *argv[]){
   int i;
   FILE *fp;
   gis_t *gis;
   int giorno,mese,anno,ore,min, gsettim;
   char stpart[25], starrivo[25],buff[100];
   struct tm *t;
   time_t tmp;


   gis = initializegis();
   MACRO(gis,NULL,"initializegis\n");
   printf("Caricamento dei file in corso...\n");
   MACRO(argc,1,"Argomenti mancatani")
   for(i=1; i < argc; i++){
      if((fp=fopen((argv[i]),"r")) == NULL)
      MACRO(fp,NULL,ABROT_1);
      readconf(fp,gis);
      fclose(fp);
   }

   printf("Caricamento completato.\n");
   printf("Inserisci la tua richiesta:\n gg/mm/aa | hh:mm | stazione partenza | stazione arrivo\ndigita Q per
terminare.\n");

   while((fgets(buff,(sizeof(buff)-2),stdin) != NULL) && (buff[0] != 'Q')){
      if((i=sscanf(buff,"%d/%d/%d|%d:%d|%[^|]|%[^\n]",&giorno,&mese,&anno,&ore,&min,stpart,starrivo)) != 7){
      printf("Errore di sintassi nella richiesta, riprova:\ngg/mm/aa | hh:mm | stazione partenza | stazione
arrivo%d\n",i);
      continue;
      }
printf("%02d/%02d/%02d %02d:%02d ---%s---%s---\n",giorno,mese,anno,ore,min,stpart,starrivo);
      if(!( (giorno >=1) && (giorno <=31) && (mese >= 1) && (mese <=12) && (min >= 0) && (ore >=0) )){
```

```
      printf("Errore di sintassi nella richiesta, riprova");
      continue;
      }
      t = localtime(&tmp);
      t->tm_year = (anno-1900);
      t->tm_mon = (mese-1);
      t->tm_mday = giorno;
      t->tm_hour = ore;
      t->tm_min = min;
      t->tm_sec = 00;
      t->tm_isdst = -1;
      tmp = mktime(t);
      t = localtime(&tmp);
      gsettim = t->tm_wday+1;
      min += (ore*60);
      cammino(gis,min,gsettim,stpart,starrivo);
      printf("Inserisci la tua richiesta:\n gg/mm/aa | hh:mm | stazione partenza | stazione arrivo\ndigita Q
per terminare.\n");
   }
}
```

pqueue.h

```
/* Author Giuseppe Quartarone mat.: 408661
 * Implementazione di un heap di minimo
 */

#define MAX_INT 65535
#define MAX_STR   200
#define LEFT(x)   ((2*x)+1)
#define PARENT(x) ((x-1)/2)
#define SWAP(x,y) tmp=x; x=y; y=tmp

/* Rappresentazione di un nodo dello heap */
typedef struct heap_node{
   int key;
```

```
   int etichetta;
}node;

/* Rappresentazione di un heap di dimensione dim e contenente size elementi*/
typedef struct priority_queue{
   int size, dim;
   node **elements;
}pqueue;

/* nodo di supprto per effettuare lo scambio tra due nodi */
node *tmp;

/* Inizzializza la struttura, restituisce -1 in caso di errore 1 altrimenti */
int initpqueue(pqueue **p, int dim);

/* Inserisce un nuovo elemento alla struttra, i nodi sono ordinati per etichetta
 * e garantisce il bilanciamento a sinistra dello heap, restituisce -1 in caso di errore 1 altrimenti */
int enqueue(pqueue *p, int key, int etichetta);

/* Estrae l'elemento in testa (minimo) restituisce il puntatore al nodo estratto NULL in caso di errore */
node *dequeue(pqueue *p);

/* Restituisce il puntatore all'elemento in testa, ma non effettua la sua estrazione, in caso di errore
restituisce NULL */
node *first(pqueue *p);

/* Stampa gli elementi della lista */
void printElements(pqueue *p);
```

pqueue.c

```
/* Author Giuseppe Quartarone mat.: 408661 */

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

```
#include "pqueue.h"
#define MAX_ERR_BUFF 100
#define DEB fprintf(stderr,"%s\n","eccomi");
#define    ABORT_1    "out of memory on malloc()"
#define    ABORT_2  "out of memory in realloc()"
#define    MACRO(x,y,z)   if( x == NULL ){ strcpy(errbuff,y); return z;}

char errbuff[MAX_ERR_BUFF];

/* Inizializza la coda, restituisce -1 in caso di errore, 1 altrimenti.
 * Param p coda da creare, dim dimensione.
 */
int initpqueue(pqueue **p, int dim){
   if(dim <= 0 || dim > MAX_INT)
      return -1;
   pqueue *new;
   new=(pqueue *)malloc(sizeof(pqueue));
   MACRO(new,ABORT_1,-1)
   new->dim = dim;
   new->elements=(node **)malloc(sizeof(node)*dim);
   MACRO(new->elements,ABORT_1,-1);
   new->size=0;
   *p=new;
   return 1;
}

/* Ripristina la proprieta` di heaptree per la cancellazione di un nuovo nodo */
void balance(pqueue *p, int i){
   int l, n;
   while((l=LEFT(i)) < p->size-1 && i !=(n=selectNode(p,i))){
      SWAP(p->elements[i],p->elements[n]);
      i=n;
   }
}

/* Seleziona il minimo tra il padre e i figli */
int selectNode(pqueue *p, int i){
```

```
   int j, k;
   j=LEFT(i);
   k=j;
   if(k+1 < p->size-1 ) k++;
   if(p->elements[k]->key < p->elements[j]->key)
      j=k;
   if(p->elements[i]->key < p->elements[j]->key)
      j=i;
   return j;
}

/* Ripristina la proprieta` di heaptree per l'inserimento di un nodo */
void riorganizza(pqueue *p, int i){
  while(i>0 && (p->elements[i]->etichetta < p->elements[PARENT(i)]->etichetta)){
      SWAP(p->elements[i],p->elements[PARENT(i)]);
      i=PARENT(i);
   }
}

/* Inserisce un nuovo nodo, nel caso in cui non ci sia spazio, provvede ad allocare
 * nuova memoira. In particolare se la coda e` piena, la sua dimensione verra` raddoppiata.
 * Restituisce -1 in caso di errore 1 altrimenti
 */
int enqueue(pqueue *p, int key, int etichetta){
   node *new;
   int i,j;
   if(p == NULL || key < 0 )
      return -1;
   if(p->size == p->dim){
      p->elements=realloc(p->elements, sizeof(node)*(p->dim*2));
      MACRO(p,ABORT_2,-1);
      p->dim=p->dim*2;
   }
   for(i=0; i<p->size;i++){
      if(p->elements[i]->key == key ){
       p->elements[i]->etichetta=etichetta;
       riorganizza(p,i);
```

```
      return 1;
      }
   }
   new=(node *)malloc(sizeof(node));
   MACRO(new,ABORT_1,-1);
   new->key=key;
   new->etichetta=etichetta;
   p->elements[p->size]=new;
   p->size++;
   i=p->size-1;
   riorganizza(p,i);
   return 1;
}
/* Estrae l'elemento in testa della coda, restituisce il puntatore a quest'ulitmo, NULL in caso di errore
 * se nella sista ci sono meno di size/4 elementi, la sua dimensione verra dimezzata */
node *dequeue(pqueue *p){
   node *min;
   if(p->size == 0)
     return NULL;
   if(p->size == (p->dim/4)){
      p->elements=realloc(p->elements, sizeof(node)*(p->dim/2));
      MACRO(p,ABORT_2,NULL)
      p->dim=p->dim/2;
   }
   min=p->elements[0];
   p->elements[0]=p->elements[p->size-1];
   p->size--;
   balance(p,0);
   return min;
}

/* Restituice il puntatore all'elemento in testa, non effettua l'estrazione */
node *first(pqueue *p){
   if(p->size > 0 )
      return p->elements[0];
   return NULL;
}
```

```
/* stampa gli elementi della coda */
void printElements(pqueue *p){
   int i=0;
   while(i < p->size){
      printf("key: %d -- etichetta: %d \n",p->elements[i]->key,p->elements[i]->etichetta);
      i++;
   }
   printf("\n");
}
```

tree.h

```
/* Author Giuseppe Quartarone mat.:408661
 * Implementazione di un albero binario
 */
#define MAXKEYLEN 100

/** Nodo dell'albero */
typedef struct tree_node{
   char *key;
   void *data;
   struct tree_node *left;
   struct tree_node *right;
} node_t;

/** Aggiunge un nodo all'albero */
int addNode(node_t **r, char *key, void *date, node_t **ret);

/** Cancella un nodo */
int delNode(node_t **r, char *key);

/** Cerca un nodo */
node_t *findNode(node_t *r, char *key);

/** stampa albero */
```

```
void printTree(node_t *r);
```

tree.c

```
/* Author Giuseppe Quartarone mat.: 408661
 * Implementazione di un albero binario
 */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "tree.h"
#define DEBUG fprintf(stderr,"%s\n","eccomi")
#define    ABORT_1    "out of memory on malloc()"
#define    MACRO(x,y)     if( x == NULL ){ fprintf(stderr,"%s\n",y); exit(EXIT_FAILURE);}
#define NewNode new=(node_t *)malloc(sizeof(node_t)); MACRO(new,ABORT_1)\
               new->key=(char *)malloc(sizeof(char)*strlen(key)+1); MACRO(new,ABORT_1)\
               strcpy(new->key, key); \
               new->data=data; \
               new->left = NULL; \
               new->right = NULL


/* Aggiunge un nodo all'albero, se il nodo esiste gia` assegna a ret il suo puntatore.
 * Restituisce -1 in caso di errore 1 altrimenti
 */
int addNode(node_t **r, char *key, void *data, node_t **ret ){
   node_t *new;
   node_t *att;
   att=*r;
   if(key == NULL || key =='\0')
      return -1;
   int i=0;
   while( att != NULL){
      i = strcmp(key, att->key);
      if(i>0){
```

```
      if(att->right != NULL )
         att=att->right;
      else{
         NewNode;
         att->right = new;
         return 1;
      }
      }
      else if(i<0){
         if(att->left != NULL)
            att=att->left;
         else{
            NewNode;
            att->left = new;
            return 1;
         }
      }
      else{
      *ret=att;
      return 1;
      }
   }
   NewNode;
   *r=new;
   return 1;
}


/* Elimina un nodo dall'albero, restituice -1 in caso di errore 1 altrimenti */
int delNode(node_t **r, char *key){
   if(r == NULL || key == NULL || key == '\0')
      return -1;
   node_t *canc, *prec, *att2, *prec2;
   int i;
   int type=0;
   canc=*r;
   prec=*r;
```

```
while(canc != NULL && (i=strncmp(key,canc->key,MAXKEYLEN)) != 0){
   if(i>0){
    prec=canc;
    canc=canc->right;
    type=1;
    }
    else{
    prec=canc;
    canc=canc->left;
    type=-1;
    }
}
if(canc == NULL)
   return -1;
if(canc->right == NULL && canc->left == NULL){     // un solo nodo
   free(canc->data);
   free(canc);
   if(prec == canc)
    *r=NULL;
   return -1;
}
att2=canc->right;
prec2=att2;
while(att2->left != NULL){
   prec2=att2;
   att2=att2->left;
}
att2->left=canc->left;
if(prec2 != att2){
   prec2->left = NULL;
   att2->right=canc->right;
}
if(canc = prec)
   *r=att2;
else{
   if(type == 1)
    prec->right = att2;
```

```
      else
       prec->left = att2;
   }
  free(canc->data);
  free(canc);
  return 1;
}
/* Ricerca un nodo e ne restituisce il suo puntatore, NULL in caso di errore */
node_t *findNode(node_t *r, char *key){
   if(r == NULL || key == NULL || key == '\0')
      return NULL;
   node_t *att;
   att=r;
   int i=0;
   while( att != NULL && (i=strncmp(key,att->key,MAXKEYLEN)) != 0 ){
      if(i>0)
       att=att->right;
      else
       att=att->left;
   }
   return att;
}
/* Stampa l'intero albero */
void printTree(node_t *r){
   if(r != NULL){
      printf("key: %s\n",r->key);
      printTree(r->left);
      printTree(r->right);
   }
}
```

list.h

```
/* Author Giuseppe Quartarone mat.: 408661
 * Implementzine di una lista
 */
```

```
#define MAXKEYLEN 100

/* Struttura rappresentate un elemento di una lista */
typedef struct list{
   int key;
   void *data;
   struct list *next;
}elem_t;

/* Aggiunge un elemento in testa alla lista */
elem_t *addTopElem(elem_t *l, int k, void *d);

/* Cerca un elemento nella lista */
elem_t *findElem(elem_t *l, int k);

/* Stampa l'intera lista */
void printList(elem_t *l);
```

list.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
#define    ABORT_1    "out of memory on malloc()"
#define    MACRO(x,y,z)   if( x == NULL ){ fprintf(stderr,"%s\n",y); return z;}



/* Aggiunge un elemento alla lista, e ne restituisce la testa, NULL in caso di errore */
elem_t *addTopElem(elem_t *l, int k, void *d){
   elem_t *new;
   new=(elem_t *)malloc(sizeof(elem_t));
   MACRO(new,ABORT_1,NULL)
   new->key=k;
   new->data=d;
```

```
   new->next=l;
   return new;
}

/* Ricerca un elemento e ne restituice il suo puntatore */
elem_t *findElem(elem_t *l, int k){
   if(l == NULL )
      return NULL;
   while(l != NULL && (k != l->key)){
      l=l->next;
   }
   return l;
}

/* Stampa la lista */
void printList(elem_t *l){
   if(l == NULL)
      printf("lista vuota");
   while(l != NULL){
      printf("key: %d\n",l->key);
      l=l->next;
   }
   printf("fine lista\n");
}
```