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

//gestione di una coda Q composto da N elementi in cui il primo elemento è quello letto più recentemente.Il programma legge da input N=lunghezza lista, poi entra in un ciclo in cui all'inizio di ogni iterazione legge un intero e : e=0 terminazione; e=1 legge un intero x e aggiorna la coda nel seguente modo: cerca x in Q,se x \in Q allora si sposta in testa, se x \notin Q allora lo aggiunge in testa e rimuove eventualmente la coda di Q se la lunghezza supera N; e=2 stampa Q


struct Lista{ int info;
 struct Lista *next;};

typedef struct Lista lista;

lista* inserisci_in_testa(lista *l, int m);
int trovato(lista *l, int x);//restituisce 1 se x appartiene alla lista e 0 altrimenti
lista* cancella_ultimo(lista *l);
void stampa(lista *l);
int lunghezza(lista *l);//restituisce la lunghezza della lista
lista* cancella_primo(lista *l);
lista* cancellax(lista *l, int x); //cancella la prima occorrenza di x in l



int main()
{
int N,e,x;
lista *Q;
scanf("%d", &N);
Q=malloc(sizeof(lista));
Q=NULL;
scanf("%d",&e);
while(e!=0)
{
if(e==1){
scanf("%d",&x);
if(trovato(Q,x)==0 && lunghezza(Q)<N){Q=inserisci_in_testa(Q,x);}
else if(trovato(Q,x)==0 && lunghezza(Q)>=N)
{Q=inserisci_in_testa(Q,x); Q=cancella_ultimo(Q);}
else //trovato(Q,x)==1
  {
  Q=cancellax(Q,x);
  Q=inserisci_in_testa(Q,x);
  }
```

```
 }
else //e==2
{stampa(Q);}

scanf("%d", &e);

}



return 0;
}




lista* inserisci_in_testa(lista *l, int m)
{
lista *temp;
temp=malloc(sizeof(lista));
temp->info=m;
temp->next=l;
return temp;
}


lista* cancella_ultimo(lista *l)
{
if(l!=NULL)
{
if(l->next==NULL)
{l=cancella_primo(l); return l;}
else {l->next=cancella_ultimo(l->next); return l;}
}
}

lista* cancella_primo(lista *l)
{
lista *temp;
temp=malloc(sizeof(lista));
temp=l;
l=l->next;
free(temp);
return l;
}

int trovato(lista *l, int x)
{
int found=0;
```

```
if (l!=NULL)
{ if(l->info==x) {found=1; return found;}
else return trovato(l->next,x);
}
}


void stampa(lista *l)
{ if (l!=NULL)
{printf("\n%d\n", l->info);
stampa(l->next);}
 }


int lunghezza(lista *l)
{
int k=0;
while(l!=NULL)
{k++; l=l->next; }

return k;
}


lista* cancellax(lista *l, int x)
{
if(l!=NULL)
{if(l->info==x) {l=cancella_primo(l); return l; }
else l->next=cancellax(l->next,x); return l; }
}
```

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

//operazioni su liste

struct Lista{ int info;
 struct Lista *next;};

typedef struct Lista lista;

lista* inserisci_in_testa(lista *l, int m);
lista* cancella_ultimo(lista *l);
void stampa(lista *l);
int lunghezza(lista *l);//restituisce la lunghezza della lista
lista* cancella_primo(lista *l);
int trovato(lista *l, int x);//1 se x appartiene alla lista e 0
altrimenti
lista* cancellax(lista *, int x);//cancella la prima occorrenza di
x



int main()
{
lista *l;
int m,x,i=0;
l=malloc(sizeof(lista));
l=NULL;
while (i<5)
{
printf("\n\ninserisci un intero\n\n");
scanf("%d",&m);
l=inserisci_in_testa(l,m);
i++;
}
printf("\n\n\n %d\n\n", lunghezza(l));
l=cancella_ultimo(l);
stampa(l);
printf("\ninserisci un intero da ricercare nella lista\n\n");
scanf("%d", &x);
printf("\n\n%d\n\n", trovato(l,x));
l=cancellax(l,x);
stampa(l);


return 0;
}

void stampa(lista *l)
{ if (l!=NULL)
```

```
{printf("\n%d\n", l->info);
stampa(l->next);    }
 }

lista* inserisci_in_testa(lista *l, int m)
{
lista *temp;
temp=malloc(sizeof(lista));
temp->info=m;
temp->next=l;
return temp;
}

int lunghezza(lista *l)
{
int k=0;
while(l!=NULL)
{k++; l=l->next; }

return k;
}


lista* cancella_ultimo(lista *l)
{
if(l!=NULL)
{
if(l->next==NULL)
{l=cancella_primo(l); return l;}
else {l->next=cancella_ultimo(l->next); return l;}
}
}

lista* cancella_primo(lista *l)
{
lista *temp;
temp=malloc(sizeof(lista));
temp=l;
l=l->next;
free(temp);
return l;
}

lista* cancellax(lista *l, int x)
{
if(l!=NULL)
{if(l->info==x) {l=cancella_primo(l); return l; }
else l->next=cancellax(l->next,x); return l; }
}
```

```
int trovato(lista *l, int x)
{
int found=0;
if (l!=NULL)
{ if(l->info==x) {found=1; return found;}
else return trovato(l->next,x);
}
}
```

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

//restituisce il nodo più profondo  comune a x e y in un albero binario tale che sul ramo *right ci sono oggetto >= chiave e sul ramo *left

struct tree{
 int chiave;
 struct tree *left;
 struct tree *right;
 };

typedef struct tree albero;

int lca(albero *a , int x, int y); //lowest common ancestor
albero* inserisci(albero *a, int k);

int main()
{
 int n,i,x,y,k;
 albero *a;
a=malloc(sizeof(albero));
a=NULL;
scanf("%d", &n);
for(i=1;i<=n;i++)
{
scanf("%d", &k);
a=inserisci(a,k);
}
scanf("%d",&x);
scanf("%d",&y);
printf("%d\n", lca(a,x,y));
}



albero* inserisci(albero *a,int k)
{
if(a==NULL)

{
 a=malloc(sizeof(albero));
 a->chiave=k;
 a->left=NULL;
 a->right=NULL;
return a;
 }
else if (k>=a->chiave)
{a->right=inserisci(a->right,k); return a;}
```

```
else {a->left=inserisci(a->left,k); return a;}

}



int lca(albero *a , int x, int y) //restituisce la chiave più
profonda comune a x e y

{
int nodo,max,min;
if(a!=NULL)
{
 if(x>y){max=x; min=y;}
 else if(x<y) {max=y; min=x;}
 else {return x; //sono sullo stesso ramo e sono uguali
}

if(max>=a->chiave && min<a->chiave) //sono su rami distinti
{return a->chiave;}

else if(min>=a->chiave) // sono entrambi sul ramo destro
 { return lca(a->right,x,y); }

else //sono entrambi sul ramo sinistro
{return lca(a->left,x,y);}

 }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//Ordinamento alfabetico di stringhe in numero arbitrario con sentinella

struct s{ char nome[dim]; };

typedef struct s T;

struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;



void stampa(coda *q);
coda* inserisci_in_testa(coda *l, char *s);
int min(coda *q, char *s);//restituisce 1 se s è il minimo di q e 0 altrimenti
coda* ordina(coda *l,char *m); //inserzione ordinata


int main()
{
coda *q;
int k,n=0,b=1;
int e=1; //sentinella
T *v;
v=(T*)malloc(sizeof(T));
while (e!=0)
 {
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 fgets(v[n].nome,dim,stdin);
 b++;
 scanf("%d", &e);
 }
printf("\n\n\n");

q=(coda*)malloc(sizeof(coda)); //creo una lista a partire dall'array dato
q=NULL;
for(k=1;k<=b+1;k++)
```

```
{
q=ordina(q,v[k].nome);
 }
stampa(q);


}



coda* inserisci_in_testa(coda *l, char *s)
 {
 coda *temp;
 temp=malloc(sizeof(coda));
 strcpy(temp->nome,s);
 temp->next=l;
return temp;
 }



int min(coda *q, char *s)
{
int m=1;
if(q!=NULL)
{
 if (strcmp(s, q->nome)>0)
 {
  m=0;
 }
else m=min(q->next,s);
 }
return m;
}


coda* ordina(coda *l,char *m) //inserzione ordinata
{
if(l!=NULL)
 {
 if(min(l,m)==1) l=inserisci_in_testa(l,m);
 else l->next=ordina(l->next,m);
  }
else l=inserisci_in_testa(l,m);
return l;
 }
```

```
void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//Ordinamento alfabetico di b stringhe

struct s{ char nome[dim]; };

typedef struct s T;

struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;




void stampa(coda *q);
coda* inserisci_in_testa(coda *l, char *s);
int min(coda *q, char *s);//restituisce 1 se s è il minimo di q e
0 altrimenti
coda* ordina(coda *l,char *m); //inserzione ordinata


int main()
{
coda *q;
int k,n=0,b;
int e;
T *v;
v=(T*)malloc(sizeof(T));
printf("\ninserisci la dimensione dell'array\n");
scanf("%d", &b);
for(e=0;e<=b;e++)
 {
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 fgets(v[n].nome,dim,stdin);
 }
printf("\n\n\n");

q=(coda*)malloc(sizeof(coda)); //creo una lista a partire
dall'array dato
q=NULL;
for(e=1;e<=b+1;e++)
{
```

```
q=ordina(q,v[e].nome);
 }
stampa(q);


}



coda* inserisci_in_testa(coda *l, char *s)
 {
 coda *temp;
 temp=malloc(sizeof(coda));
 strcpy(temp->nome,s);
 temp->next=l;
return temp;
 }



int min(coda *q, char *s)
{
int m=1;
if(q!=NULL)
{
 if (strcmp(s, q->nome)>0)
 {
  m=0;
 }
else m=min(q->next,s);
 }
return m;
}


coda* ordina(coda *l,char *m) //inserzione ordinata
{
if(l!=NULL)
 {
 if(min(l,m)==1) l=inserisci_in_testa(l,m);
 else l->next=ordina(l->next,m);
  }
else l=inserisci_in_testa(l,m);
return l;
 }
```

```
void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}
```

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

//gestione di una coda Q composto da N elementi in cui il primo elemento è quello letto più recentemente.Il programma legge da input N=lunghezza lista, poi entra in un ciclo in cui all'inizio di ogni iterazione legge un intero e : e=0 terminazione; e=1 legge un intero x e aggiorna la coda nel seguente modo: cerca x in Q,se x \in Q allora si sposta in testa, se x \notin Q allora lo aggiunge in testa e rimuove eventualmente la coda di Q se la lunghezza supera N; e=2 stampa Q con output esempio 6 3 $


struct Lista{ int info;
 struct Lista *next;};

typedef struct Lista lista;

lista* inserisci_in_testa(lista *l, int m);
int trovato(lista *l, int x);//restituisce 1 se x appartiene alla lista e 0 altrimenti
lista* cancella_ultimo(lista *l);
void stampa(lista *l);
int lunghezza(lista *l);//restituisce la lunghezza della lista
lista* cancella_primo(lista *l);
lista* cancellax(lista *l, int x); //cancella la prima occorrenza di x in l



int main()
{
int N,e,x;
lista *Q;
scanf("%d", &N);
Q=malloc(sizeof(lista));
Q=NULL;
scanf("%d",&e);
while(e!=0)
{
if(e==1){
scanf("%d",&x);
if(trovato(Q,x)==0 && lunghezza(Q)<N){Q=inserisci_in_testa(Q,x);}
else if(trovato(Q,x)==0 && lunghezza(Q)>=N)
{Q=inserisci_in_testa(Q,x); Q=cancella_ultimo(Q);}
else //trovato(Q,x)==1
  {
  Q=cancellax(Q,x);
  Q=inserisci_in_testa(Q,x);
  }
```

```
 }
else //e==2
{stampa(Q); putchar('$');}

scanf("%d", &e);

}



return 0;
}




lista* inserisci_in_testa(lista *l, int m)
{
lista *temp;
temp=malloc(sizeof(lista));
temp->info=m;
temp->next=l;
return temp;
}


lista* cancella_ultimo(lista *l)
{
if(l!=NULL)
{
if(l->next==NULL)
{l=cancella_primo(l); return l;}
else {l->next=cancella_ultimo(l->next); return l;}
}
}

lista* cancella_primo(lista *l)
{
lista *temp;
temp=malloc(sizeof(lista));
temp=l;
l=l->next;
free(temp);
return l;
}

int trovato(lista *l, int x)
{
int found=0;
```

```
if (l!=NULL)
{ if(l->info==x) {found=1; return found;}
else return trovato(l->next,x);
}
}


void stampa(lista *l)
{ if (l!=NULL)
{printf("%d", l->info); putchar(' ');
stampa(l->next);}
 }


int lunghezza(lista *l)
{
int k=0;
while(l!=NULL)
{k++; l=l->next; }

return k;
}


lista* cancellax(lista *l, int x)
{
if(l!=NULL)
{if(l->info==x) {l=cancella_primo(l); return l; }
else l->next=cancellax(l->next,x); return l; }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 100

//N coppie chiave e valore. le N chiavi sono stringhe di al più
100 caratteri per le quali si fa un albero di ricerca non
bilanciato i valori sono int positivi. prende in input una stringa
s che è presente nell'albero. u nodo dell'albero avente chiave s.
albero secondo le stringhe


struct albero{ char chiave[dim]; int valore; struct albero *left;
struct albero *right; };
typedef struct albero tree;

tree* add(tree *t, char *s, int n); //aggiunge un nodo all'albero
tree* ricerca(tree *t, char *s); //restituisce sottoalbero che
parte da s
int max(tree *t); //massimo intero nell'albero t
int trovato(tree *t, int m);//restituisce 0 se m è il massimo
intero presente in t e >0 altrimenti
int massimo(int a,int b);



int main()
{
int N,k,n;
char s[dim], u[dim];
tree *t;
t=malloc(sizeof(tree));
t=NULL;
scanf("%d", &N);
for(k=0;k<N;k++)
{ scanf("%s", s); scanf("%d", &n); t=add(t,s,n);  }

scanf("%s", u);
t=ricerca(t,u);
printf("\n");
printf("%d", max(t));
printf("\n");
return 0;

}

tree* add(tree *t, char *s, int n)
{
if(t==NULL)
{t=malloc(sizeof(tree)); strcpy(t->chiave, s); t->valore=n;
```

```
 t->left=NULL; t->right=NULL; return t; }

else if(strcmp(s,t->chiave)>=0) {t->right=add(t->right,s,n);
return t; }

else {t->left=add(t->left,s,n); return t;}
}


tree* ricerca(tree *t, char *s)
{
if(t!=NULL)
 {
   if(strcmp(s,t->chiave)==0) {return t;}
  else if(strcmp(s,t->chiave)>0)
{t=t->right;
return ricerca(t,s);}
  else {t=t->left; return ricerca(t,s); }
 }

}


int max(tree *t)
{
int m;
if(t!=NULL)
 {
 m=t->valore;
 if(trovato(t,m)==0) // m è il massimo in t
  { return m; }

 else if (trovato(t->left,m)==0) //m è più grande degli altri
valori in t->left
     { t=t->right; return max(t);  }

 else if (trovato(t->right,m)==0) //m è più grande degli altri
valori in t->right
     { t=t->left; return max(t);  }

 else //ci sono elementi più grandi di m sia in right che in left
    { return massimo(max(t->left), max(t->right));  }

 }
}
```

```c
int trovato(tree *t, int m)
{
int k;
k=0;
if(t!=NULL)
{
  if(t->valore > m) //m non è il massimo
 { k++; return k; }

  else
 { return trovato(t->left,m)+trovato(t->right,m); }


 }

}
```

```c
int massimo(int a,int b)

{
if(a>=b) return a;
else return b;
}
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define dim 15

//(tutte le stringhe sono senza spazi) vengono inserite in un
albero binario non bilanciato M coppie personaggio-ID, viene poi
creato un array di liste di dimensione N che contiene alla k-esima
posizione la lista di vittime del personaggio con id k. dopo aver
preso in input l'albero si prendono le N coppie di nomi da input,
il primo è il personaggio al quale si vuole associare il secondo
che è la sua vittima. ovvero si ricerca l'ID del primo nome
inserito e si inserisce il secondo nome in testa a l[ID]. alla
fine di questo secondo ciclo di chiede di inserire un nome e si
stampa la lista dei personaggi che ha ucciso



struct albero{ char personaggio[dim]; int ID; struct albero
*right; struct albero *left; };

struct Lista{ char info[dim]; struct Lista *next;   };

typedef struct Lista lista;
typedef struct albero tree;

tree* add(tree *t, char *s, int id); //aggiunge un nodo all'albero
lista* add1(lista *l, char *s);//aggiunge un nodo in testa alla
lista l
void stampa(tree *t);
void stampa1(lista *l);
int ricerca(tree *t, char *s);//restituisce l'ID del personaggio
di nome s presente nell'albero t


int main()
{
int x,M,m,id,k,N, num;
tree *t;
char s[dim],interr[dim], per[dim], vittima[dim];
lista **l;

t=malloc(sizeof(tree));
t=NULL;
scanf("%d", &M);

for(m=0;m<M;m++)
{ scanf("%s", s); scanf("%d", &id); t=add(t,s,id);    }

stampa(t);
```

```
printf("\n\n");

scanf("%d", &N);
l=malloc(N*sizeof(lista*));

for(k=0;k<N;k++)
{ l[k+1]=malloc(sizeof(lista));  }

for(k=0;k<N;k++)
{ scanf("%s", per); x=ricerca(t,per); scanf("%s", vittima);
l[x]=add1(l[x],vittima);  }

scanf("%s",interr);
num=ricerca(t,interr);
stampa1(l[num]);

return 0;
}



tree* add(tree *t, char *s, int id)
{
if (t==NULL) { t=malloc(sizeof(tree)); strcpy(t->personaggio,s);
t->ID=id; t->right=NULL; t->left=NULL; return t;}

else if (strcmp(s,t->personaggio)>=0)
{t->right= add(t->right,s,id); return t;}

else {t->left= add(t->left,s,id); return t;}
}


lista* add1(lista *l, char *s)
{ lista *temp;
  temp=malloc(sizeof(lista));
  strcpy(temp->info,s);
  temp->next=l;
  return temp;
  }


void stampa(tree *t)
{ if(t!=NULL)
 { printf("\n%s\n", t->personaggio); printf("%d", t->ID);
stampa(t->right); stampa(t->left);  }

 }
```

```c
void stampa1(lista *l)
{
 if(l!=NULL) { printf("\n%s\n", l->info); stampa1(l->next); }
}


int ricerca(tree *t, char *s)
{
 if(t!=NULL)
 {
 if(strcmp(s,t->personaggio)==0) { return t->ID;}
else if(strcmp(s,t->personaggio)>0) {return ricerca(t->right,s);}
else return ricerca(t->left,s);
 }
}
```

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


struct tree{
 int chiave;
 struct tree *left;
 struct tree *right;
 };

typedef struct tree albero;

int lca(albero *a , int x, int y); //lowest common ancestor
albero* inserisci(albero *a, int k);
int trovato(albero *a, int m);

int main()
{
 int n,i,x,y,k;
 albero *a;
a=malloc(sizeof(albero));
a=NULL;
scanf("%d", &n);
for(i=1;i<=n;i++)
{
scanf("%d", &k);
a=inserisci(a,k);
}
scanf("%d",&x);
scanf("%d",&y);
printf("%d\n", lca(a,x,y));
}



albero* inserisci(albero *a,int k)
{
if(a==NULL)

{
 a=malloc(sizeof(albero));
 a->chiave=k;
 a->left=NULL;
 a->right=NULL;
return a;
 }
else if (k>=a->chiave)
{a->right=inserisci(a->right,k); return a;}

else {a->left=inserisci(a->left,k); return a;}
```

```
}


int lca(albero *a , int x, int y) //restituisce la chiave meno
profonda comune a x e y

{
int nodo,max,min;
if(a!=NULL)
{
 if(x>y){max=x; min=y;}
 else if(x<y) {max=y; min=x;}
 else {
     return x; //sono sullo stesso ramo e sono uguali
}

if(max>=a->chiave && min<a->chiave) //sono su rami distinti
{return a->chiave;}

else if(min>=a->chiave) // sono entrambi sul ramo destro
 { return lca(a->right,x,y); }

else //sono entrambi sul ramo sinistro
{return lca(a->left,x,y);}

 }
}
```

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

//restituisce il nodo più profondo  comune a x e y in un albero binario tale che sul ramo *right ci sono oggetto >= chiave e sul ramo *left

struct tree{
 int chiave;
 struct tree *left;
 struct tree *right;
 };

typedef struct tree albero;

int lca(albero *a , int x, int y); //lowest common ancestor
albero* inserisci(albero *a, int k);

int main()
{
 int n,i,x,y,k;
 albero *a;
a=malloc(sizeof(albero));
a=NULL;
scanf("%d", &n);
for(i=1;i<=n;i++)
{
scanf("%d", &k);
a=inserisci(a,k);
}
scanf("%d",&x);
scanf("%d",&y);
printf("%d\n", lca(a,x,y));
}



albero* inserisci(albero *a,int k)
{
if(a==NULL)

{
 a=malloc(sizeof(albero));
 a->chiave=k;
 a->left=NULL;
 a->right=NULL;
return a;
 }
else if (k>=a->chiave)
{a->right=inserisci(a->right,k); return a;}
```

```
else {a->left=inserisci(a->left,k); return a;}

}



int lca(albero *a , int x, int y) //restituisce la chiave più
profonda comune a x e y

{
int nodo,max,min;
if(a!=NULL)
{
 if(x>y){max=x; min=y;}
 else if(x<y) {max=y; min=x;}
 else {return x; //sono sullo stesso ramo e sono uguali
}

if(max>=a->chiave && min<a->chiave) //sono su rami distinti
{return a->chiave;}

else if(min>=a->chiave) // sono entrambi sul ramo destro
 { return lca(a->right,x,y); }

else //sono entrambi sul ramo sinistro
{return lca(a->left,x,y);}

 }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//prende in ingresso un array di stringhe e restituisce l'ultimo
in ordine alfabetico

struct s{ char nome[dim]; };

typedef struct s T;

struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;



char* max(T *S,int n);//n=num elementi dell'array, restituisce
l'ultima parola in ordine alfabetico
void stampa(coda *q);
coda* inserisci_in_testa(coda *l, char *s);


int main()
{
coda *q;
int k,n=0,b;
int e;
T *v;
v=(T*)malloc(sizeof(T));
printf("\ninserisci la dimensione dell'array\n");
scanf("%d", &b);
for(e=0;e<=b;e++)
 {
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 printf("\ninserisci una stringa\n");
 fgets(v[n].nome,dim,stdin);
 }
printf("\n\n\n");

q=(coda*)malloc(sizeof(coda)); //creo una lista a partire
dall'array dato
q=NULL;
for(e=1;e<=b+1;e++)
```

```
{
q=inserisci_in_testa(q,v[e].nome);
 }
stampa(q);


}



char* max(T *S,int n) // restiuisce l'ultima stringa in ordine
alfabetico
{
int k=1;
char *s;
s=(char*)malloc(dim*sizeof(char));
strcpy(s,S[1].nome);
while(k<=n)
 {
 if(strcmp(s, S[k].nome)<0)
 {strcpy(s,S[k].nome);k++;}
 else k++;
 }
return s;
}


coda* inserisci_in_testa(coda *l, char *s)
 {
 coda *temp;
 temp=malloc(sizeof(coda));
 strcpy(temp->nome,s);
 temp->next=l;
return temp;
 }




void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}
```

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

//operazioni su liste

struct Lista{ int info;
 struct Lista *next;};

typedef struct Lista lista;

lista* inserisci_in_testa(lista *l, int m);
lista* cancella_ultimo(lista *l);
void stampa(lista *l);
int lunghezza(lista *l);//restituisce la lunghezza della lista
lista* cancella_primo(lista *l);
int trovato(lista *l, int x);//1 se x appartiene alla lista e 0
altrimenti
lista* cancellax(lista *, int x);//cancella la prima occorrenza di
x



int main()
{
lista *l;
int m,x,i=0;
l=malloc(sizeof(lista));
l=NULL;
while (i<5)
{
printf("\n\ninserisci un intero\n\n");
scanf("%d",&m);
l=inserisci_in_testa(l,m);
i++;
}
printf("\n\n\n %d\n\n", lunghezza(l));
l=cancella_ultimo(l);
stampa(l);
printf("\ninserisci un intero da ricercare nella lista\n\n");
scanf("%d", &x);
printf("\n\n%d\n\n", trovato(l,x));
l=cancellax(l,x);
stampa(l);


return 0;
}

void stampa(lista *l)
{ if (l!=NULL)
```

```
{printf("\n%d\n", l->info);
stampa(l->next);    }
 }

lista* inserisci_in_testa(lista *l, int m)
{
lista *temp;
temp=malloc(sizeof(lista));
temp->info=m;
temp->next=l;
return temp;
}

int lunghezza(lista *l)
{
int k=0;
while(l!=NULL)
{k++; l=l->next; }

return k;
}


lista* cancella_ultimo(lista *l)
{
if(l!=NULL)
{
if(l->next==NULL)
{l=cancella_primo(l); return l;}
else {l->next=cancella_ultimo(l->next); return l;}
}
}

lista* cancella_primo(lista *l)
{
lista *temp;
temp=malloc(sizeof(lista));
temp=l;
l=l->next;
free(temp);
return l;
}

lista* cancellax(lista *l, int x)
{
if(l!=NULL)
{if(l->info==x) {l=cancella_primo(l); return l; }
else l->next=cancellax(l->next,x); return l; }
}
```

```
int trovato(lista *l, int x)
{
int found=0;
if (l!=NULL)
{ if(l->info==x) {found=1; return found;}
else return trovato(l->next,x);
}
}
```

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

//ordina una lista di interi

struct lista{
 int info;
 struct lista *next;

};

typedef struct lista Lista;

Lista* ordina(Lista *l, int m);
int min(Lista *l, int m);/*restituisce 1 se m è <= di tutti gli
elementi di l e 0 altrimenti*/
Lista* inserisci_in_testa(Lista *l, int m);
void stampa(Lista *l);

int main()
 {
Lista *l;
int m,n,i;
l=malloc(sizeof(Lista));
printf("\ninserisci il numero di elementi della lista\n");
scanf("%d",&n);
printf("\ninserisci il primo elemento della lista\n");
scanf("%d", &(l->info));
l->next=NULL;
for(i=1;i<n;i++)
 {
 printf("\ninserisci un elemento\n");
 scanf("%d",&m);
 l=ordina(l,m);
 }
printf("\n\n\n");
stampa(l);
return 0;

 }


void stampa(Lista *l)
 {
 if(l!=NULL)
  {
 printf("\n%d\n",l->info);
 stampa(l->next);
  }
 }
```

```
Lista* inserisci_in_testa(Lista *l, int m)
 {
 Lista *temp;
 temp=malloc(sizeof(Lista));
 temp->info=m;
 temp->next=l;
return temp;
 }



int min(Lista *l, int m)
 {
int trovato=1;
if(l!=NULL)
{ if(m>(l->info)) trovato=0;
else trovato=min(l->next,m);}
return trovato;
 }

Lista* ordina(Lista *l,int m)/*inserzione ordinata in lista d
interi non vuota*/
{
if(l!=NULL)
 {
 if(min(l,m)==1) l=inserisci_in_testa(l,m);
 else l->next=ordina(l->next,m);
  }
else l=inserisci_in_testa(l,m);
return l;
 }
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//Ordinamento alfabetico di stringhe in numero arbitrario con sentinella

struct s{ char nome[dim]; };

typedef struct s T;

struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;



void stampa(coda *q);
coda* inserisci_in_testa(coda *l, char *s);
int min(coda *q, char *s);//restituisce 1 se s è il minimo di q e 0 altrimenti
coda* ordina(coda *l,char *m); //inserzione ordinata


int main()
{
coda *q;
int k,n=0,b=1;
int e=1; //sentinella
T *v;
v=(T*)malloc(sizeof(T));
while (e!=0)
 {
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 fgets(v[n].nome,dim,stdin);
 b++;
 scanf("%d", &e);
 }
printf("\n\n\n");

q=(coda*)malloc(sizeof(coda)); //creo una lista a partire dall'array dato
q=NULL;
for(k=1;k<=b+1;k++)
```

```
{
q=ordina(q,v[k].nome);
 }
stampa(q);


}



coda* inserisci_in_testa(coda *l, char *s)
 {
 coda *temp;
 temp=malloc(sizeof(coda));
 strcpy(temp->nome,s);
 temp->next=l;
return temp;
 }



int min(coda *q, char *s)
{
int m=1;
if(q!=NULL)
{
 if (strcmp(s, q->nome)>0)
 {
  m=0;
 }
else m=min(q->next,s);
 }
return m;
}


coda* ordina(coda *l,char *m) //inserzione ordinata
{
if(l!=NULL)
 {
 if(min(l,m)==1) l=inserisci_in_testa(l,m);
 else l->next=ordina(l->next,m);
  }
else l=inserisci_in_testa(l,m);
return l;
 }
```

```
void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//Ordinamento alfabetico di b stringhe

struct s{ char nome[dim]; };

typedef struct s T;

struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;




void stampa(coda *q);
coda* inserisci_in_testa(coda *l, char *s);
int min(coda *q, char *s);//restituisce 1 se s è il minimo di q e
0 altrimenti
coda* ordina(coda *l,char *m); //inserzione ordinata


int main()
{
coda *q;
int k,n=0,b;
int e;
T *v;
v=(T*)malloc(sizeof(T));
printf("\ninserisci la dimensione dell'array\n");
scanf("%d", &b);
for(e=0;e<=b;e++)
 {
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 fgets(v[n].nome,dim,stdin);
 }
printf("\n\n\n");

q=(coda*)malloc(sizeof(coda)); //creo una lista a partire
dall'array dato
q=NULL;
for(e=1;e<=b+1;e++)
{
```

```
q=ordina(q,v[e].nome);
 }
stampa(q);


}



coda* inserisci_in_testa(coda *l, char *s)
 {
 coda *temp;
 temp=malloc(sizeof(coda));
 strcpy(temp->nome,s);
 temp->next=l;
return temp;
 }



int min(coda *q, char *s)
{
int m=1;
if(q!=NULL)
{
 if (strcmp(s, q->nome)>0)
 {
  m=0;
 }
else m=min(q->next,s);
 }
return m;
}


coda* ordina(coda *l,char *m) //inserzione ordinata
{
if(l!=NULL)
 {
 if(min(l,m)==1) l=inserisci_in_testa(l,m);
 else l->next=ordina(l->next,m);
  }
else l=inserisci_in_testa(l,m);
return l;
 }
```

```
void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//Ordinamento alfabetico di b stringhe

struct s{ char nome[dim]; };

typedef struct s T;


struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;




void stampa(coda *q);
coda* inserisci_in_testa(coda *l, char *s);
int min(coda *q, char *s);//restituisce 1 se s è il minimo di q e
0 altrimenti
coda* ordina(coda *l,char *m); //inserzione ordinata


int main()
{
coda *q;
int k,n=0,b;
int e;
T *v;
v=(T*)malloc(sizeof(T));
printf("\ninserisci la dimensione dell'array\n");
scanf("%d", &b);
for(e=0;e<=b;e++)
 {
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 printf("\ninserisci una stringa\n");
 fgets(v[n].nome,dim,stdin);
 }
printf("\n\n\n");

q=(coda*)malloc(sizeof(coda)); //creo una lista a partire
dall'array dato
q=NULL;
```

```
for(e=1;e<=b+1;e++)
{
q=ordina(q,v[e].nome);
 }
stampa(q);


}



coda* inserisci_in_testa(coda *l, char *s)
 {
 coda *temp;
 temp=malloc(sizeof(coda));
 strcpy(temp->nome,s);
 temp->next=l;
return temp;
 }



int min(coda *q, char *s)
{
int m=1;
if(q!=NULL)
{
 if (strcmp(s, q->nome)>0) //s è più grande di q->nome in ordine
alfabetico
 {
  m=0;
 }
else m=min(q->next,s);
 }
return m;
}


coda* ordina(coda *l,char *m) //inserzione ordinata
{
if(l!=NULL)
 {
 if(min(l,m)==1) l=inserisci_in_testa(l,m);
 else l->next=ordina(l->next,m);
  }
else l=inserisci_in_testa(l,m);
return l;
 }
```

```
void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//minima stringa tra due (alfabeticamente)

struct Coda{
 char nome[dim];
 struct Coda *next;
};

typedef struct Coda coda;

void stampa(coda *q);
coda* inserisci_in_fondo(coda *q, char *s);
int min(coda *q, char *s);
char* minima(coda *q);



int main ()
{

   char s1[dim];
   char s2[dim];
   int ret;
   char *s;
   coda *q;

   q=malloc(sizeof(coda));
   q=NULL;
   printf("\n inserisci la prima stringa\n\n");
   fgets(s1,dim,stdin);
   printf("\n inserisci la seconda stringa\n\n");
   fgets(s2,dim,stdin);


   ret = strcmp(s1, s2);

   if(ret <= 0)
   {
     q=inserisci_in_fondo(q,s1);
     q=inserisci_in_fondo(q,s2);
   }
else {q=inserisci_in_fondo(q,s2);
     q=inserisci_in_fondo(q,s1);   }

printf("\n\n\n");
stampa(q);
```

```
printf("\n%s\n", minima(q));

   return 0;
}

coda* inserisci_in_fondo(coda *q, char *s)
{
 if(q==NULL)
 {q=malloc(sizeof(coda));
strcpy(q->nome,s);
 q->next=NULL;
 return q;
  }
else {q->next=inserisci_in_fondo(q->next,s);
      return q;}

}


char* minima(coda *q)
{
char *s;
if(q!=NULL)
{
  s=q->nome;
  if (min(q,s)==1)
  return s;
  else
   { q=q->next;
     s=minima(q);
     return s;
    }
 }
}




void stampa(coda *q)
{
  while(q!=NULL)
{printf("%s\n", q->nome);
 q=q->next; }

}


int min(coda *q, char *s)
{
```

```
int m=1;
if(q!=NULL)
{
 if (strcmp(s, q->nome)>0)
 {
  m=0;
 }
else m=min(q->next,s);
 }
return m;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define dim 15

//prende in ingresso un array di stringhe e restituisce l'ultimo
in ordine alfabetico (+ funzione di minimo)

struct s{ char nome[dim]; };

typedef struct s T;


int min(T *S, char *t, int n);
char* max(T *S,int n);//n=num elementi dell'array, restituisce
l'ultima parola in ordine alfabetico



int main()
{
int k,n=0,b;
int e;
T *v;
v=(T*)malloc(sizeof(T));
printf("\ninserisci la dimensione dell'array\n");
scanf("%d", &b);
for(e=0;e<=b;e++)
{
 n++;
 v=realloc(v,(n+1)*sizeof(T));
 printf("\ninserisci una stringa\n");
 fgets(v[n].nome,dim,stdin);
}
for(k=1;k<=b+1;k++)
{printf("\n%s\n",v[k].nome);}

printf("\n%s\n", max(v,b+1));
return 0;
}

int min(T *S, char *t, int n) //restituisce 1 se t è il minimo
alfabetico di S e 0 altrimenti
{
int r=1;
int k=0;
while(k<=n && r==1)
{
 if(strcmp(t, S[k].nome)>0) r=0;//t non è il minimo
 else k++;
```

```
}
return r;
}


char* max(T *S,int n) // restiuisce l'ultima stringa in ordine
alfabetico
{
int k=1;
char *s;
s=(char*)malloc(dim*sizeof(char));
strcpy(s,S[1].nome);
while(k<=n)
 {
 if(strcmp(s, S[k].nome)<0)
 {strcpy(s,S[k].nome);k++;}
 else k++;
 }
return s;
}
```