```c
#include "esercizio.h"
//Evviva Stalin :)
int ricercaLivello(TipoAlbero a, TipoInfoAlbero v, int livello_corrente){
if (a==NULL) return 0;
if(a->info==v) return 0;
return 1+ricercaLivello(a->sinistro,v,livello_corrente);
return 1+ricercaLivello(a->destro,v,livello_corrente);
}
int verificaNodi(TipoAlbero a, int livello){
if(a==NULL) return 0;
if(a->info%2==livello%2) return 1+verificaNodi(a->sinistro,livello+1)+verificaNodi(a->destro,livello+1);
return  verificaNodi(a->sinistro,livello+1)+verificaNodi(a->destro,livello+1);
}

int singleChildSum(TipoAlbero a){
if(a->sinistro==NULL && a->destro!= NULL) return a->info+singleChildSum(a->destro);
if(a->sinistro!=NULL && a->destro==NULL )return a->info+singleChildSum(a->sinistro);
if(a->sinistro!=NULL && a->destro!=NULL) return singleChildSum(a->sinistro)+singleChildSum(a->destro);
}



void diobestia(TipoListaSCL* l,TipoAlbero a){
        if (a==NULL) return;
        if(a->sinistro == NULL && a->destro ==NULL) listPushBack(l,a->info);
        diobestia(l,a->sinistro);
        diobestia(l,a->destro);
}
TipoListaSCL* listaNodiFoglia(TipoAlbero a){
        TipoListaSCL* hue = (TipoListaSCL*) malloc(sizeof(TipoListaSCL));
        diobestia(hue,a);
        TipoListaSCL* temp = hue;
        hue=hue->next;
        free(temp);
        return hue;
}

void pep(TipoAlbero a, Coda * r){
        if(a==NULL) return;
        if(!estVuoto(a->sinistro)&&!estVuoto(a->destro))        inCoda(r,a->info);
        pep(a->sinistro,r);
        pep(a->destro,r);
}
Coda* codaNodiDueFigli(TipoAlbero a){
        Coda* r =(Coda*) malloc(sizeof(Coda));
        pep(a, r);
        return r;
}

int prof(TipoAlbero a){
        if(a==NULL) return 0;
        if(a->sinistro == NULL) return 1+prof(a->destro);
        if    (a->destro==NULL) return 1+prof(a->sinistro);
        if(prof(a->sinistro)>prof(a->destro)) {
                return 1+prof(a->sinistro);
        }
        return 1+prof(a->destro);
}
void lollerino(TipoListaSCL* l,TipoAlbero a){
        if(a==NULL) return;
        if(prof(a->sinistro)>=prof(a->destro)&&!estVuoto(a->sinistro)){
                listPushBack(l,a->sinistro->info);
                lollerino(l,a->sinistro);
        }
        else{
        if(!estVuoto(a->destro)){
                listPushBack(l,a->destro->info);
```

```
                lollerino(l,a->destro);
        }
        }
        return;
}


TipoListaSCL* listaPercorso(TipoAlbero a){
        TipoListaSCL* hue = (TipoListaSCL*) malloc(sizeof(TipoListaSCL));
        lollerino(hue,a);
        hue->value=a->info;
        return hue;
}




TipoAlbero albBinVuoto () {
        return 0;
}

TipoAlbero creaAlbBin( TipoInfoAlbero infoRadice, TipoAlbero sx, TipoAlbero dx ) {
        TipoAlbero a = ( TipoAlbero ) malloc ( sizeof (TipoNodoAlbero) ) ;
        a->info = infoRadice ;
        a->sinistro = sx ;
        a->destro = dx ;
        return a ;
}



bool estVuoto ( TipoAlbero a ) {
        return ( a == NULL );
}

TipoInfoAlbero radice ( TipoAlbero a ) {
        if ( a == NULL ) {
                printf ( " ERRORE accesso albero vuoto \n " );
                return ERRORE_InfoAlbero;
        }
        else
                return a->info;
}

TipoAlbero sinistro ( TipoAlbero a ) {
        if ( a == NULL ) {
                printf ( " ERRORE accesso albero vuoto \n " );
        return NULL;
        }
        else
                return a->sinistro ;
}

TipoAlbero destro ( TipoAlbero a ) {
        if ( a == NULL ) {
                printf ( " ERRORE accesso albero vuoto \n " );
                return NULL;
        }
        else
                return a->destro;
}

TipoListaSCL* creaLista(){
        return NULL;
}
```

```c
int lunghezzaLista(TipoListaSCL* l){
        int lun = 0;
        if(l == NULL)
                return 0;
        lun = 1;
        TipoListaSCL* aux = l;
        while(aux->next != NULL){
                aux = aux->next;
                lun++;
        }
        return lun;
}

TipoListaSCL* listPushFront(TipoListaSCL* l, TipoInfoAlbero info){
         TipoListaSCL* new_element = ( TipoListaSCL* ) malloc ( sizeof (TipoListaSCL) ) ;
         new_element->next = l;
         new_element->value = info;
         return new_element;
}

TipoListaSCL* listPushBack(TipoListaSCL* l, TipoInfoAlbero info){
         TipoListaSCL* new_element = ( TipoListaSCL* ) malloc ( sizeof (TipoListaSCL) ) ;
         new_element->value = info;
         new_element->next = NULL;

         if( l == NULL ){
                l = listPushFront(l, info);
                return l;
         }

         TipoListaSCL* aux = l;
         while(aux->next != NULL)
                aux = aux->next;

         aux->next = new_element;
         return l;
}

void printList(TipoListaSCL* l){
  if(l == NULL)
    return;
  TipoListaSCL* aux = l;
  while(aux != NULL){
    printf(" %d ", aux->value);
    aux = aux->next;
  }
  return;
}
```

```c
Coda* codaVuota() {
  Coda* r = (Coda*) malloc (sizeof(Coda));
  r->data = NULL; // Necessario per usare realloc
  r->size = 0;
  r->nelem = 0;
  return r;
}

bool estVuota(Coda* c) {
        if(c == NULL)
```

```
                return 0;
        return (c->nelem==0);
}

void inCoda(Coda* c, TipoInfoAlbero e){
  c->nelem++;
  if (c->nelem > c->size){
    // Raddoppiamento dimensione array
    c -> size = 2 * c-> nelem;
    c -> data = (TipoInfoAlbero*) realloc(c->data,c->size*sizeof(TipoInfoAlbero));
  }
  c->data[c->nelem -1] = e;
}

void outCoda(Coda* c){
  if (c == NULL || c->nelem == 0){
    printf("ERRORE: input NULL o coda vuota");
    exit(1);
  }
  c->nelem--;
  if (c->nelem < c->size/2){
    // Dimezzamento array
    c->size /= 2;
    c->data = (TipoInfoAlbero*) realloc(c->data,c->size*sizeof(TipoInfoAlbero));
  }
  // Copia tutti gli elementi della coda nella componente precedente
  int i;
  for (i = 1; i <= c->nelem; i++){
    c->data[i-1]=c->data[i];
  }
}

void printInfo(TipoInfoAlbero e){
        printf("%d", e);
}

TipoInfoAlbero primo(Coda* c){
  if (c->nelem == 0){
    printf("ERRORE: coda vuota");
    exit(1);
  }
        return c->data[0];
}

IteratoreCoda* creaIteratoreCoda(Coda *c) {
    IteratoreCoda *it = (IteratoreCoda *)malloc(sizeof(IteratoreCoda));
    it->c = c;
    it->ptr = 0;
    return it;
}

bool hasNext(IteratoreCoda *it) {
    return it->ptr < it->c->nelem;
}

TipoInfoAlbero next(IteratoreCoda *it) {
    TipoInfoAlbero r = ERRORE_InfoAlbero;
    if (hasNext(it)) {
        r = it->c->data[it->ptr];
        it->ptr ++;
    }
    else
        printf("ERRORE Iteratore non valido.\n");
    return r;
}
```

```c
void printCoda(Coda* c){
  if(c == NULL)
  {
        printf(" coda non allocata\n");
        return;
  }
  IteratoreCoda* it = creaIteratoreCoda(c);
  while(hasNext(it)){
    printInfo(next(it));
    printf(" ");
  }
  printf("\n");
}
```