```
#include <stdlib.h>
#include <malloc.h>
#define Link_Item struct Item*
#define Link_List struct List*
#define Link_Stack struct Stack*
#define Link_Queue struct Queue*

void ERR(const char* s) { fputs(s,stderr); exit (1); }

struct Item { T node; Link_Item next; };
Link_Item Item_Create(T elem, Link_Item n) {
   Link_Item rab = (Link_Item)calloc(1,sizeof(struct Item));
   rab->node=elem; rab->next=n; return rab; }
T Item_get_node(Link_Item a) { return a->node; }
Link_Item Item_get_next(Link_Item a) { return a->next; }
void Item_Delete(Link_Item a) { free(a); }

struct Stack { Link_Item Top; };
void Stack_ini(Link_Stack a) { a->Top = NULL; }
int Stack_Is_Empty(const Link_Stack a) { return a->Top==NULL; }
void Stack_push(Link_Stack a, T elem) { a->Top=Item_Create(elem,a->Top); }
T Stack_pop(Link_Stack a) {
   if(Stack_Is_Empty(a)) ERR("Stack::pop: empty stack");
   Link_Item p = a->Top; T rab = Item_get_node(p); a->Top = Item_get_next(p);
   Item_Delete(p); return rab; }

struct Queue { Link_Item Head; Link_Item Tail; };
void Queue_ini(Link_Queue a) { a->Head = a->Tail = NULL; }
int Queue_Is_Empty(const Link_Queue a) { return a->Head==NULL; }
void Queue_put(Link_Queue a, T elem) {
   if(Queue_Is_Empty(a)) a->Tail = a->Head = Item_Create(elem,NULL);
   else a->Tail= (a->Tail->next = Item_Create(elem,NULL)); }
```

```
T Queue_get(Link_Queue a) {
   if(Queue_Is_Empty(a)) ERR("Queue::get: gueue is empty");
   Link_Item p = a->Head; T rab = Item_get_node(p); a->Head=Item_get_next(p);
   Item_Delete(p); if(Queue_Is_Empty(a)) a->Tail=NULL; return rab; }

struct List { Link_Item Front; Link_Item Back; };
void List_ini(Link_List a) { a->Front = a->Back = NULL; }
int List_Is_Empty(Link_List a) { return a->Front==NULL; }
Link_Item List_Find(Link_List a, Link_Item *F, T key) { if(List_Is_Empty(a)) return (*F=NULL);
   Link_Item ptr = *F = a->Front; if(Item_get_node(*F)==key) return NULL;
   while((*F=Item_get_next(ptr))!=NULL) { if(Item_get_node(*F)==key) break; ptr=*F; }
   return ptr; }
void List_Insert_front(Link_List a, T elem) { a->Front = Item_Create(elem,a->Front);
   if(a->Back==NULL) a->Back = a->Front; }
int List_Insert_after(Link_List a, T elem, T after) { Link_Item c; List_Find(a,&c,after);
   if(c==NULL) return 0; c->next = Item_Create(elem,Item_get_next(c)); return 1; }
int List_Insert_back(Link_List a, T elem) {
   if(List_Is_Empty(a)) a->Front = a->Back = Item_Create(elem,NULL);
   else a->Back = (a->Back->next = Item_Create(elem,NULL)); }
T List_Remove_front(Link_List a) { if(List_Is_Empty(a)) ERR("List::Remove_front: list is empty");
   Link_Item p=a->Front; T rab=Item_get_node(p); a->Front=Item_get_next(p); Item_Delete(p);
   if(a->Front==NULL) a->Back=NULL; return rab; }
int List_Remove(Link_List a, T key) { Link_Item b; Link_Item c; b=List_Find(a,&c,key);
   if(c==NULL) return 0;
   if(b==NULL) a->Front=Item_get_next(a->Front); else b->next=Item_get_next(c);
   Item_Delete(c); return 1; }
T List_Remove_back(Link_List a) { if(List_Is_Empty(a)) ERR("List::Remove_back: list is empty");
   Link_Item p=a->Front; T rab = Item_get_node(a->Back);
   if(a->Front==a->Back) a->Front = a->Back = NULL;
   else { while(p->next!=a->Back) p=p->next; a->Back=p; p=p->next; a->Back->next=NULL; }
   Item_Delete(p); return rab; }
void List_Revers(Link_List a) { Link_Item p=NULL;
```

```
    while(a->Front!=NULL) p=Item_Create(List_Remove_front(a),p);
    a->Front=p; while(Item_get_next(p)!=NULL) p=Item_get_next(p); a->Back=p; }
void List_Sort(Link_List a, int (*f)(const T a, const T b)) { Link_Item F;
    while(a->Front!=NULL) { Link_Item p = a->Front; T rab = Item_get_node(p);
        while(p=Item_get_next(p)) if(f(p->node,rab)>0) rab=Item_get_node(p);
        F=Item_Create(rab,F); List_Remove(a,rab); }
    a->Front = F; while(F->next) F=F->next; a->Back=F; }
```