```
#include <stdlib.h>


inline void ERR(const char* s) { cerr << s << endl;  exit(1); }


template <typename T> class Item { T node; Item* next;

public:

  Item(const T &elem, Item* n=0) { node=elem; next=n; }

  T& get_node() { return node; }

  Item* &get_next() { return next; }

};


template <typename T> class Stack { Item<T> *top; T rab;

public:

  Stack() { top = 0; }

  void push(const T& elem) { top = new Item<T>(elem,top); }

  T& pop() { if(!top) ERR("Stack::pop: empty stack"); rab = top->get_node();

          Item<T> *p=top; top = p->get_next(); delete p; return rab; }

  bool empty() const {return top == 0; }

};


template <typename T> class Queue { Item<T> *head,*tail; T rab;

public:

  Queue() { tail = head =0; }

  void put(const T& elem) {

          if(tail==0) tail = head = new Item<T>(elem); else tail = (tail->get_next() = new Item<T>(elem)); }

  T& get() { if(head==0) ERR("Queue::get: queue is empty"); rab = head->get_node();

          Item<T> *p=head;  head=head->get_next(); delete p; if(head==0) tail=0; return rab; }
  bool empty() { return head==0;}
};


template <typename T> class List { Item<T> *front,*back; T rab;
Item<T>* find(Item<T>* &F, const T& k) { if(front==NULL) return (F=NULL);
        Item<T> *ptr=F=front; if(front->get_node()==k) return 0;
        while((F=ptr->get_next())!=NULL) { if(F->get_node()==k) break; ptr=F; }
        return ptr; }
public:
List() { front = back =0; }
bool empty() { return front==0; }
void push_back(const T& elem) {
        if(back==0) front = back = new Item<T>(elem);
        else back = (back->get_next() = new Item<T>(elem));  }
T& pop_back() {
        if(back==0) ERR("List::pop_back: list is empty");
        rab=back->get_node(); Item<T> *p=front;
        if(front==back) front = back = 0; else { while(p->get_next()!=back) p=p->get_next();
                back=p; p=p->get_next(); back->get_next()=0;}
        delete p; return rab; }
bool insert_after(const T& k, const T& after) {
        Item<T> *c; find(c,after); if(c==0) return 0;
        c->get_next()=new Item<T>(k,c->get_next()); return 1; }
bool remove(const T& k) { Item<T> *b,*c; b=find(c,k);
        if(!c) return 0;
        if(b==NULL) { front=front->get_next(); delete (c); }
        else { b->get_next()=c->get_next(); delete(c); }
```

```
            return 1;}
void push_front(const T& elem) {
            front = new Item<T>(elem,front);
            if(back==0) back = front; }
T& pop_front() {
            if(front==0) ERR("List::pop_front: list is empty");
            Item<T> *p=front; rab = p->get_node();
            front=p->get_next(); delete p;
            if(front==0) back=0; return rab; }
void sort() { Item<T> *D=0;
            while(front!=0) { Item<T> *p=front; rab = front->get_node();
              while(p=p->get_next())if(p->get_node()>rab) rab=p->get_node();
              D = new Item<T>(rab,D); remove(rab); }
            front = D; back = front;
            while(back->get_next()!=0) back=back->get_next(); }
void revers() { Item<T> *D=0;
            while(front!=0) D = new Item<T>(pop_front(),D);
            front = D; back=front;
            while(back->get_next()!=0) back=back->get_next(); }
T& operator[](int i) { Item<T>* p=front;
            while(i-- && p) p=p->get_next(); if(p) return p->get_node();
            ERR("LIST::operator[]: end of List appear");}
friend ostream& operator<<(ostream& out, const List<T>& a) {
            Item<T>* p=a.front;
            while(p) { out << p->get_node() << endl; p=p->get_next(); }
            return out; }
};
```