```cpp
#include <iostream>
using namespace std;

const int MAX_QUEUE_LENGTH = 100;

//队列过程抽象和封装的实现:

struct Queue_process {
    int front;
    int rear;
    int buffer[MAX_QUEUE_LENGTH];
};

//数据类型的定义与操作的定义是分开的，二者之间没有显式的联系
//数据表示是公开的，无法防止使用者直接操作队列数据，会带来问题

void init(Queue_process& q) {
    q.front = 0;
    q.rear = -1;
} //容易忘记初始化

void insert_p(Queue_process& q, int i) {
    if (q.rear == MAX_QUEUE_LENGTH - 1) {
        cout << "Max length exceeded.\n";
        exit(-1);
    }
    else {
        q.rear++;
        q.buffer[q.rear] = i;
        return;
    }
}

void delete_p(Queue_process& q, int& i) {
    if (q.rear == -1) {
        cout << "There is no queue.\n";
        exit(-1);
    }
    else {
        i = q.buffer[q.front];
        q.rear--;
        for (int j = q.front; j <= q.rear; j++)
            q.buffer[j] = q.buffer[j + 1];
        return;
    }
}

//insert、delete在形式上与下面的函数f没有区别，函数f也能作用于q
//void f(Queue& q) { ...... } f(q); 操作之后q可能不再是“队列”

//队列数据抽象和封装的实现（数组）：

class Queue_data1 {
public: //对外的接口（外部可使用的内容）
    Queue_data1();
    void insert_d1(int i);
    void delete_d1(int& i);
private: //隐藏的内容，外部不可使用
    int front;
    int rear;
    int buffer[MAX_QUEUE_LENGTH];
};

//数据类型的定义与操作的定义之间有联系
//数据表示是私密的，可以防止使用者直接操作队列数据带来的问题

Queue_data1::Queue_data1() {
    front = 0;
    rear = -1;
    for (int j = 0; j < MAX_QUEUE_LENGTH; j++)
        buffer[j] = 0;
} //定义队列数据时自动初始化

void Queue_data1::insert_d1(int i) {
    if (rear == MAX_QUEUE_LENGTH - 1) {
        cout << "Max length exceeded.\n";
        exit(-1);
    }
    else {
        rear++;
        buffer[rear] = i;
        return;
    }
}

void Queue_data1::delete_d1(int& i) {
    if (rear == -1) {
        cout << "There is no queue.\n";
        exit(-1);
    }
    else {
        i = buffer[front];
        rear--;
        for (int j = front; j <= rear; j++)
            buffer[j] = buffer[j + 1];
        return;
    }
}

//队列数据抽象和封装的实现（链表）：

class Queue_data2 {
public:
    Queue_data2();
    void insert_d2(int i);
    void delete_d2(int& i);
private:
    struct Node {
        int content;
        Node* next;
    } *front, * rear;
};

Queue_data2::Queue_data2() {
    front = NULL;
    rear = NULL;
}

void Queue_data2::insert_d2(int i) {
    Node* p = new Node;
    p->content = i;
    p->next = NULL;
    if (p == NULL) {
        cout << "Max length exceeded.\n";
        exit(-1);
    }
    else {
        if (front == NULL) {
            front = p;
            rear = p;
        }
        else {
            rear->next = p;
            rear = p;
            return;
        }
    }
}

void Queue_data2::delete_d2(int& i) {
    if (front == NULL) {
        cout << "There is no queue.\n";
        exit(-1);
    }
    else {
        Node* p = front;
        front = front->next;
        i = p->content;
        delete p;
        return;
    }
} //链表比数组在两端同时插入或删除时更易操作

//main函数:

int main()
{
    Queue_process q; //定义队列数据q
    int x;
    init(q);  //对q进行初始化
    insert_p(q, 12);
    delete_p(q, x);

    Queue_data1 q1; //定义队列数据并初始化
    int y;
    q1.insert_d1(13);
    q1.delete_d1(y);

    Queue_data2 q2; //定义队列数据并初始化
    int z;
    q2.insert_d2(14);
    q2.delete_d2(z);

    return 0;
}
```