3.16

(1)

```
int Ackerman(int m, int n) {
    if (m == 0) return n + 1;
    if (n == 0) return Ackerman(m - 1, 1);
    return Ackerman(m - 1, Ackerman(m, n - 1));
}
```

(2)

```
int Ackerman(int m, int n) {
    int i, j, top = -1;
    int Sm[10], Sn[10];
    Sm[++top] = m;
    Sn[top] = n;
    while (top >= 0) {
        i = Sm[top];
        j = Sn[top--];
        if (i == 0) {
            if (top != -1) Sn[top] = j + 1;
        }
        else if (j == 0) {
            Sm[++top] = i - 1;
            Sn[top] = 1;
        }
        else {
            Sm[++top] = i - 1;
            Sm[++top] = i;
            Sn[top] = j - 1;
        }
    }
    return j + 1;
}
```

3.23

```
template<typename T>
class Queue {
public:
    Queue(int size = 100);
    ~Queue() { delete[] queue; }
    bool Enqueue(const T& x);
    bool Dequeue(T& x);
    bool getfront(T& x);
    void makeEmpty() { front = rear = tag = 0; }
    bool IsEmpty() const { return front == rear && tag == 0; }
    bool IsFull() const { return front == rear && tag == 1; }
private:
    int front, rear, tag;
```

```
    T* queue;
    int maxsize;
};

template<typename T>
Queue<T>::Queue(int size)
{
    maxsize = size;
    queue = new T[maxsize];
    assert(queue != NULL);
    front = rear = tag = 0;
}

template<typename T>
bool Queue<T>::Enqueue(const T& x)
{
    if (IsFull()) return false;
    rear = (rear + 1) % maxsize;
    queue[rear] = x;
    tag = 1;
    return true;
}

template<typename T>
bool Queue<T>::Dequeue(T& x)
{
    if (IsEmpty()) return false;
    front = (front + 1) % maxsize;
    x = queue[front];
    tag = 0;
    return true;
}

template<typename T>
bool Queue<T>::getfront(T& x)
{
    if (IsEmpty()) return false;
    x = queue[(front + 1) % maxsize];
    return true;
}
```

3.24

(1)

```
template<typename T>
bool Queue<T>::Enqueue(const T& x)
{
    if ((rear + 1) % maxsize == front) {
        int newsize = maxsize * 2;
        T* newqueue = new T[newsize];
        if (newqueue == NULL) {
            std::cerr << "Memory Allocation Error!" << std::endl;
            return false;
        }
```

```
        int newfront = 0, newrear = 0;
        while (rear != front) {
            newqueue[newrear] = queue[front];
            front = (front + 1) % maxsize;
            newrear = (newrear + 1) % newsize;
        }
        delete[] queue;
        queue = newqueue; front = newfront; rear = newrear; maxsize = newsize;
    }
    queue[rear] = x;
    rear = (rear + 1) % maxsize;
    return true;
}
```

(2)

```
template<typename T>
bool Queue<T>::Dequeue(T& x)
{
    if ((rear - front + maxsize) % maxsize <= maxsize / 4) {
        int newsize = maxsize / 2;
        T* newqueue = new T[newsize];
        if (newqueue == NULL) {
            std::cerr << "Memory Allocation Error!" << std::endl;
            return false;
        }
        int newfront = 0, newrear = 0;
        while (rear != front) {
            newqueue[newrear] = queue[front];
            front = (front + 1) % maxsize;
            newrear = (newrear + 1) % newsize;
        }
        delete[] queue;
        queue = newqueue; front = newfront; rear = newrear; maxsize = newsize;
    }
    if (rear == front) return false;
    x = queue[front];
    front = (front + 1) % maxsize;
    return true;
}
```

3.26

```
template<typename T>
class DoubleQueue {
public:
    DoubleQueue(int size = 100);
    ~DoubleQueue() { delete[] queue; }
    bool Enqueue(const T& x, int end);
    bool Dlqueue(T& x, int end);
    bool getfront(T& x, int end);
    void makeEmpty() { end1 = end2 = 0; }
    bool IsEmpty() const { return end1 == end2; }
    bool IsFull() const { return (end2 + 1) % maxsize == end1; }
```

```
private:
    int end1, end2;
    T* queue;
    int maxsize;
};

template<typename T>
DoubleQueue<T>::DoubleQueue(int size)
{
    maxsize = size;
    queue = new T[maxsize];
    assert(queue != NULL);
    end1 = end2 = 0;
}

template<typename T>
bool DoubleQueue<T>::Enqueue(const T& x, int end)
{
    if (IsFull()) return false;
    if (end == 1) {
        end1 = (end1 - 1 + maxsize) % maxsize;
        queue[end1] = x;
    }
    else {
        queue[end2] = 2;
        end2 = (end2 + 1) % maxsize;
    }
    return true;
}

template<typename T>
bool DoubleQueue<T>::Dlqueue(T& x, int end)
{
    if (IsEmpty()) return false;
    if (end == 1) {
        x = queue[end1];
        end1 = (end1 + 1) % maxsize;
    }
    else {
        end2 = (end2 - 1 + maxsize) % maxsize;
        x = queue[end2];
    }
    return true;
}

template<typename T>
bool DoubleQueue<T>::getfront(T& x, int end)
{
    if (IsEmpty()) return false;
    if (end == 1) x = queue[end1];
    else x = queue[(end2 - 1 + maxsize) % maxsize];
    return true;
}
```