3.7

```
bool match(std::string s)
{
    std::stack<char> p;
    int l = s.length();
    for (int i = 0; i < l; i++)
    {
        if (s[i] == '(' || s[i] == '{' || s[i] == '[')
            p.push(s[i]);
        else if (s[i] == ')')
        {
            if (p.top() == '(')
                p.pop();
            else
                return false;
        }
        else if (s[i] == '}')
        {
            if (p.top() == '{')
                p.pop();
            else
                return false;
        }
        else if (s[i] == ']')
        {
            if (p.top() == '[')
                p.pop();
            else
                return false;
        }
    }
    if (p.empty())
        return true;
    else
        return false;
}
```

3.8

```
std::string polish(std::string str)
{
    std::map<char, int> priority;
    priority['+'] = 1;
    priority['-'] = 1;
    priority['*'] = 2;
    priority['/'] = 2;
    std::stack<char> s1, s2;
```

```
    int i = 0;
    while (i < str.length())
    {
        char iter = str[i];
        if ((iter <= 'z' && iter >= 'a') || (iter <= 'Z' && iter >= 'A') || iter
== '(')
        {
            if (iter != '(')
                s1.push(iter);
            else
                s2.push(iter);
            i++;
        }
        else if (iter == ')')
        {
            while (!s2.empty() && s2.top() != '(')
            {
                s1.push(s2.top());
                s2.pop();
            }
            if (!s2.empty())
                s2.pop();
            i++;
        }
        else if (s2.empty() || s2.top() == '(')
        {
            s2.push(iter);
            i++;
        }
        else if (priority[iter] > priority[s2.top()])
        {
            s2.push(iter);
            i++;
        }
        else
        {
            s1.push(s2.top());
            s2.pop();
        }
    }
    while (!s2.empty())
    {
        s1.push(s2.top());
        s2.pop();
    }
    while (!s1.empty())
    {
        s2.push(s1.top());
        s1.pop();
    }
    std::string ans;
    while (!s2.empty())
    {
        ans.push_back(s2.top());
```

```
            s2.pop();
        }
        return ans;
    }
```

3.9

```
double polish_value(std::vector<std::string> data)
{
    std::stack<double> s;
    for (auto iter : data)
    {
        if (iter == "+" || iter == "-" || iter == "*" || iter == "/")
        {
            double num1 = s.top();
            s.pop();
            double num2 = s.top();
            s.pop();
            if (iter == "+")
                num1 = num2 + num1;
            else if (iter == "-")
                num1 = num2 - num1;
            else if (iter == "*")
                num1 = num1 * num2;
            else
                num1 = num2 / num1;
            s.push(num1);
        }
        else
            s.push(strtof(iter.c_str(), nullptr));
    }
    return s.top();
}
```

3.10

```
#define ElemType int
typedef struct QNode
{
    ElemType data;
    struct QNode *next;
} QNode, *QueuePtr;
typedef struct
{
    QueuePtr rear;
    QNode head;
```

```
} LinkQueue;
void InitQueue(LinkQueue &Q)
{
    Q.rear = new QNode;
    Q.rear->next = Q.rear;
}
void EnQueue(LinkQueue &Q, ElemType e)
{
    QueuePtr S = new QNode;
    S->data = e;
    S->next = Q.rear->next;
    Q.rear->next = S;
    Q.rear = S;
}
void DeQueue(LinkQueue &Q, ElemType &e)
{

    QNode *q = Q.rear->next->next;
    e = q->data;
    Q.rear->next->next = q->next;
    if (Q.rear->next->next == Q.rear->next)
        Q.rear = Q.rear->next;
    delete q;
}
```

3.11

```
#define MaxQSize 7
#define ElemType int
typedef struct
{
    ElemType *base;
    int rear;
    int length;
} Queue;

Queue InitQueue()
{
    Queue Q;
    Q.base = (ElemType *)malloc(MaxQSize * sizeof(ElemType));
    Q.rear = 0;
    Q.length = 0;
    return Q;
}

void EnQueue(Queue &Q, ElemType x)
{
    if (Q.length == MaxQSize)
    {
        printf("队满");
```

```
        exit(0);
    }
    Q.base[Q.rear] = x;
    Q.rear = (Q.rear + 1) % MaxQSize;
    Q.length++;
}

ElemType DeQueue(Queue &Q)
{
    ElemType x;
    if (Q.length == 0)
    {
        printf("队空");
        exit(0);
    }
    x = Q.base[(Q.rear + MaxQSize - Q.length + 1) % MaxQSize];
    Q.length--;
    return x;
}
```