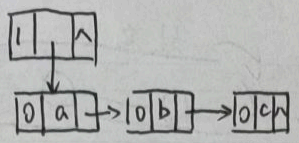
作业六

T1

> 画出下列广义表的存储结构，写出其长度、深度以及表头和表尾。
>
> $A=(a,b,c)$
>
> $B=(a,(b,(c)),d)$
>
> $C=((a,b),(c,d))$
>
> $D=(a,(b,(),c),((d),e))$
>
> $E=((((a),b)),((((),(d)),(ef)))$

A = (a, b, c)

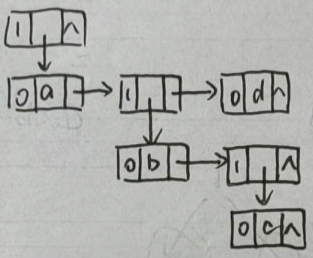


长度：3，深度：1

表头：a

表尾：(b,c)

B = (a, (b, (c)), d)



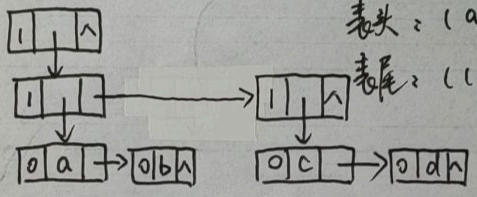
长度：3，深度：3

表头：a

表尾：((b,(c)),d)

C = ((a,b),(c,d))

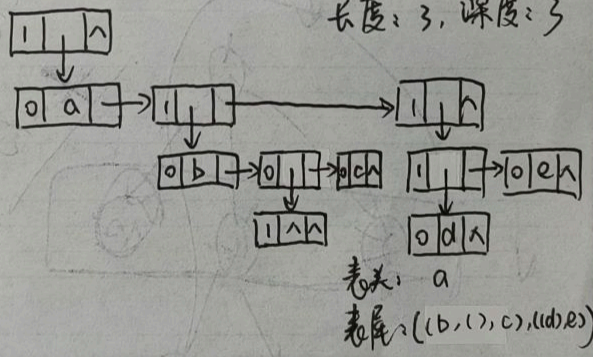
长度：2，深度：2



表头：(a,b)

表尾：((c,d))

D = (a,(b,(),c),((d),e))

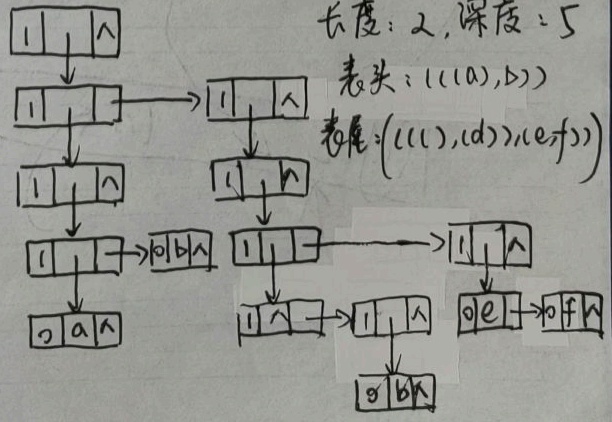


长度：3，深度：3

表头：a

表尾：((b,(),c),((d),e))

E = (((((a),b)),(((),(d)),(e,f)))



长度：2，深度：5

表头：(((a),b))

表尾：((((),(d)),(e,f)))

T2

> 试编写判别两个广义表是否相等的算法。

同时遍历两个广义表，分情况进行判断即可。时间复杂度 $O(n)$

```cpp
template<class T>
bool GList<T>::same(GListNode<T>* pa, GListNode<T>* pb) {
    if (!pa && !pb) {
        return true;
    } else if (pa && !pb || !pa && pb) {
        return false;
    } else if (pa->type != pb->type) {
        return false;
    } else if (pa->type == ATOM) {
        // both are atom
        return pa->data == pb->data && same(pa->next, pb->next);
    } else {
        // both are list
        return same(pa->sublist, pb->sublist) && same(pa->next, pb->next);
    }
}
```

T3

> 试编写一个算法，删除广义表中所有元素值为 x 的原子结点。

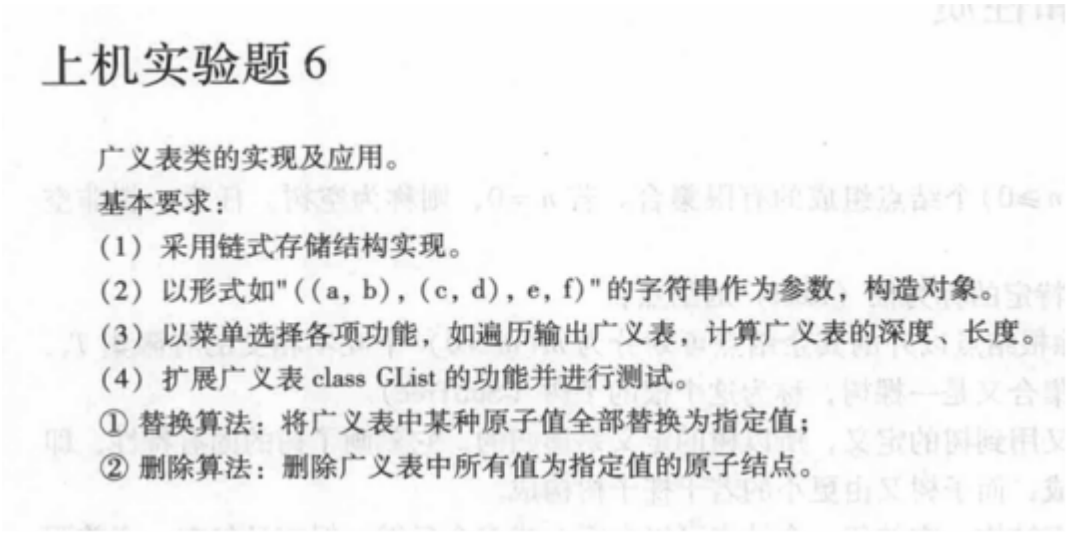
同样是遍历广义表，遇到原子值为 x 的结点删除即可。与单链表删除结点的思路如出一辙，保留前驱，指向后继即可。需要注意的细节是单链表的前驱只有一种，而广义表的前驱有两种，因此在将前驱指向后继时需要特判一下前驱的类型。时间复杂度 $O(n)$

```cpp
template<class T>
void GList<T>::deleteAtom(GListNode<T>* pre, GListNode<T>* now, T x) {
    if (!now) {
        return;
    }

    if (now->type == LIST) {
        deleteAtom(now, now->sublist, x);
        deleteAtom(now, now->next, x);
    } else if (now->data == x) {
```

```
        GListNode<T>* temp = now;
        now = now->next;
        delete temp;
        if (pre->type == LIST) {
            pre->sublist = now;
        } else {
            pre->next = now;
        }
        deleteAtom(now, now->next, x);
    } else {
        deleteAtom(now, now->next, x);
    }
}
```

实验六



上机实验题 6

广义表类的实现及应用。

基本要求：

(1) 采用链式存储结构实现。

(2) 以形式如"((a, b), (c, d), e, f)"的字符串作为参数，构造对象。

(3) 以菜单选择各项功能，如遍历输出广义表，计算广义表的深度、长度。

(4) 扩展广义表 class GList 的功能并进行测试。

① 替换算法：将广义表中某种原子值全部替换为指定值；

② 删除算法：删除广义表中所有值为指定值的原子结点。

声明列表：由于需要支持「比较两个广义表是否相同」的逻辑，因此不得已将头结点暴露为公有变量。

```
enum GListNodeType { ATOM, LIST };

template<class T>
struct GListNode {
    GListNodeType type;
    union {
        T data;
        GListNode* sublist;
    };
    GListNode<T>* next;
};

```

```cpp
template<class T>
class GList {
private:
    GListNode<T>* create(const string& s, int& i);
    void decreate(GListNode<T>* now);
    void print(GListNode<T>* now);
    int length(GListNode<T>* now);
    int depth(GListNode<T>* now);
    void replaceAtom(GListNode<T>* h, T origin, T target);
    void deleteAtom(GListNode<T>* pre, GListNode<T>* now, T x);
    bool same(GListNode<T>* pa, GListNode<T>* pb);

public:
    GListNode<T>* head;
    GList();
    GList(const string& s);              // create with string
    ~GList();

    void print();                        // print glist
    int length();                        // calculate glist length
    int depth();                         // calculate glist depth
    void replaceAtom(T origin, T target); // replace atoms from origin to target
    void deleteAtom(T x);                // delete atoms of x
    bool same(GList<T>& obj);            // compare to another glist
};
```

构建广义表：

```cpp
template<class T>
GListNode<T>* GList<T>::build(const string& s, int& i) {
    while (i < s.size() && (s[i] == ' ' || s[i] == ',')) {
        i++;
    }

    if (i == s.size()) {
        return nullptr;
    } else if (s[i] == ')') {
        i++;
        return nullptr;
    }

```

```
    GListNode<T>* h = new GListNode<T>;
    T now = s[i++];

    if (now == '(') {
        h->type = LIST;
        h->sublist = build(s, i);
        h->next = build(s, i);
    } else {
        h->type = ATOM;
        h->data = now;
        h->next = build(s, i);
    }

    return h;
}
```

打印广义表：

```
template<class T>
void GList<T>::print(GListNode<T>* now) {
    if (!now) {
        return;
    }

    if (now->type == ATOM) {
        cout << now->data;
    } else { // now->type == LIST
        cout << '(';
        print(now->sublist);
        cout << ')';
    }

    if (now->next) {
        cout << ',';
        print(now->next);
    }
}
```

计算深度：

```cpp
template<class T>
int GList<T>::depth(GListNode<T>* now) {
    if (!now) {
        return 0;
    }
    if (now->type == ATOM) {
        return depth(now->next);
    } else {
        return max(1 + depth(now->sublist), depth(now->next));
    }
}
```

计算长度：

```cpp
template<class T>
int GList<T>::length(GListNode<T>* now) {
    if (!now) {
        return 0;
    }
    return length(now->next) + 1;
}
```

替换原子：

```cpp
template<class T>
void GList<T>::replaceAtom(GListNode<T>* now, T origin, T target) {
    if (!now) {
        return;
    }

    if (now->type == ATOM) {
        if (now->data == origin) {
            now->data = target;
        }
        replaceAtom(now->next, origin, target);
    } else {
        replaceAtom(now->sublist, origin, target);
        replaceAtom(now->next, origin, target);
    }
}
```