**题目：**

实现线性表的顺序存储结构，并分析每个基本操作（算法）的时间复杂度.

CODE:

#include <iostream>

#include <stdlib.h>

#define maxlength 11

using namespace std;

//顺序表的顺序储存结构

struct LIST

{

char a[maxlength];

int last;

};

typedef int position;

position End(LIST L)

{

return L.last + 1;

}

void Insert(char x, position p, LIST &L)

{

position q;

if(L.last >= maxlength - 1)

{

cout << L.last << " " << maxlength << endl;

cout << "Error! List is full" << endl;

exit(1);

}

else if((p > L.last + 1) || (p < 1))

{

cout << "Error! Position does not Exist!" << endl;

exit(1);

}

else

{

for(q = L.last; q >= p; q--)

{

L.a[q+1] = L.a[q];

L.last = L.last + 1;

L.a[p] = x;

}

}

}

void Delete(position p, LIST &L)

{

position q;

if((p > L.last) || (p < 1))

{

cout << "Error! The position does not exist!" << endl;

exit(1);

}

else

{

L.last = L.last - 1;

for(q = p; q <= L.last; q++)

{

L.a[q] = L.a[q+1];

}

}

}

position Location(char x, LIST L)

{

position q;

for(q = 1; q <= L.last; q++)

{

if(L.a[q] == x)

{

return q;

}

}

return L.last+1;

}

void Print(LIST L)

{

for(int i = 1; i < maxlength; i++)

{

cout << L.a[i] << " " << endl;

}

}

int main()

{

LIST L;

char ch;

int p;

cout << "输入10个字符：" << endl;

for(int i = 1; i < maxlength; i++)

{

cin >> ch;

L.a[i] = ch;

}

L.last = maxlength-2;

cout << "线性表的长度为：" << endl;

cout << End(L) << endl;

cout << "请输入你要插入的位置：" << endl;

cin >> p;

cout << "请输入你要插入的字母：" << endl;

cin >> ch;

Insert(ch, p, L);

cout << "现在线性表为：" << endl;

Print(L);

cout << "请输入你要删除的序号" << endl;

cin >> p;

Delete(p, L);

cout << "现在线性表为：" << endl;

Print(L);

cout << "请输入你要查找的字符" << endl;

cin >> ch;

cout << ch << "在" << Location(ch, L) << "处" << endl;

return 0;

}

**复杂度分析：**

1. End(LIST L)：值执行L.last+1一次，时间复杂度为O(1).
2. Insert(char x, position p, LIST &L):最坏的情况下执行q-p次基本操作，所以最坏的时间复杂度为O(N).
3. void Delete(position p, LIST &L):循环执行L.last-p，所以最坏的时间复杂度为O(N).
4. Location(char x, LIST L):循环在最坏的情况下执行n次，在循环被都是基本操作，所以最坏时间复杂度为O(N).