**\* Programming.**

**Question 11.** How do you implement "delete" operation in a single-linked list?

**Solution:**

Steps:

1. If head is null, return null
2. Create a dummy node (in case the head is asked to be removed)
3. Pass the address of dummy node to temp
4. Delete the node use dummy = dummy->next->next to remove dummy->next
5. Return temp->next as the new head

// C++

class ListNode { // create a data structure “ListNode”

public:

int value;

ListNode\* next;

ListNode (int v): value(v), next(NULL) {}

void deleteNode (ListNode\* head, int index);

};

ListNode::deleteNode (ListNode\* head, int index) { // the index counts from 1

if (head == NULL) {

return;

}

ListNode\* dummy = new ListNode(-1); // using dummy node

ListNode\* temp = dummy;

dummy->next = head;

for (int i = 0; i < index - 1; ++i) {

dummy = dummy->next;

}

dummy->next = dummy->next->next;

temp = temp->next;

head = temp;

}

void test1(){ // remove “3” from 1->2->3->4

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

ListNode\* temp1 = head;

while(temp1 != NULL) {

cout << temp1->value << " -> ";

temp1 = temp1->next;

}

cout << "null\n";

deleteNode(head, 3);

ListNode\* temp2 = head;

while(temp2 != NULL) {

cout << temp2->value << " -> ";

temp2 = temp2->next;

}

cout << "null\n";

}

void test2(){ remove “1” from 1->2->3->4

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

ListNode\* temp1 = head;

while(temp1 != NULL) {

cout << temp1->value << " -> ";

temp1 = temp1->next;

}

cout << "null\n";

deleteNode(head, 1);

ListNode\* temp2 = head;

while(temp2 != NULL) {

cout << temp2->value << " -> ";

temp2 = temp2->next;

}

cout << "null\n";

}

void test3(){ // the list is null

ListNode\* head = NULL;

ListNode\* temp1 = head;

while(temp1 != NULL) {

cout << temp1->value << " -> ";

temp1 = temp1->next;

}

cout << "null\n";

deleteNode(head, 3);

ListNode\* temp2 = head;

while(temp2 != NULL) {

cout << temp2->value << " -> ";

temp2 = temp2->next;

}

cout << "null\n";

}

**Question 13.** Can the constructor of a class be virtual? How to realize a similar function as a virtual constructor?

**Solution:**

// C++

// Reference: Bjarne Stroustrup's C++ Style and Technique FAQ: “Why don't we have virtual constructors?”

// The constructor of a class cannot be virtual. However similar function can be realized:

struct F { // interface to object creation functions

virtual A\* make\_an\_A() const = 0;

virtual B\* make\_a\_B() const = 0;

};

void user(const F& fac) {

A\* p = fac.make\_an\_A(); // make an A of the appropriate type

B\* q = fac.make\_a\_B(); // make a B of the appropriate type

// ...

}

struct FX : F {

A\* make\_an\_A() const {

return new AX();

} // AX is derived from A

B\* make\_a\_B() const {

return new BX();

} // BX is derived from B

};

struct FY : F {

A\* make\_an\_A() const {

return new AY();

} // AY is derived from A

B\* make\_a\_B() const {

return new BY();

} // BY is derived from B

};

int main() {

FX x;

FY y;

user(x); // this user makes AXs and BXs

user(y); // this user makes AYs and BYs

user(FX()); // this user makes AXs and BXs

user(FY()); // this user makes AYs and BYs

// ...

}

/\* This is a variant of what is often called "the factory pattern". The point is that user() is completely isolated from knowledge of classes such as AX and AY.

\*/

**Question 15.** Given a string, return the longest palindrome subsequence.

**Solution:**

Steps:

1. Use dynamic programming
2. Create int M[i][j] to save the length of the longest palindrome subsequence between i and j
3. Create string S[i][j] to save the longest palindrome subsequence between i and j
4. sl is the substring (**not subsequence!)** length
5. When sl = 1, M[i][i] = 1, S[i][i] = input[i]
6. When sl = 2, if input[i] == input[j], M[i][j] = 2, S[i][j] = input[i] + input[j]
7. For sl > 2, if input[i] == input[j], M[i][j] = M[i+1][j-1] + 2, S[i][j] = input[i] + S[i+1][j-1] + input[j]
8. For sl > 2, if input[i] != input[j], M[i][j] = max(M[i][j-1], M[i+1][j]),

S[i][j] = M[i][j-1] >= M[i+1][j] ? S[i][j-1] : S[i+1][j]

1. Return S[0][len - 1], where len is the length of original string

// C++

class Solution {

public:

string longestPalindrome(string input) {

if (input.size() == 0) {

return input;

}

int len = input.size();

int i, j, sl; // sl is substring length

int M[len][len]; // M[][] saves the length of longest subsequence between i and j

string S[len][len] = {""}; // S[][] saves the longest subsequence between i and j

for(i = 0; i < len; ++i) {

M[i][i] = 1;

S[i][i] = input[i];

}

for(sl = 2; sl <= len; ++sl) {

for (i=0; i<len - sl + 1; i++) {

j = i + sl - 1;

if (input[i] == input[j] && sl == 2) {

M[i][j] = 2;

S[i][j] = input[i] + S[i][j] + input[j];

} else if (input[i] == input[j]){

M[i][j] = M[i+1][j-1] + 2;

S[i][j] = input[i] + S[i+1][j-1] + input[j];

} else {

M[i][j] = max(M[i][j-1], M[i+1][j]);

S[i][j] = M[i][j-1] >= M[i+1][j] ? S[i][j-1] : S[i+1][j];

}

}

}

return S[0][len - 1];

}

void test1() {

string test = longestPalindrome("");

for (int i = 0; i < test.size(); ++i) {

cout << test[i];

}

cout << endl;

}

void test2() {

string test = longestPalindrome("ab");

for (int i = 0; i < test.size(); ++i) {

cout << test[i];

}

cout << endl;

}

void test3() {

string test = longestPalindrome("aa");

for (int i = 0; i < test.size(); ++i) {

cout << test[i];

}

cout << endl;

}

void test4() {

string test = longestPalindrome("quantitative");

for (int i = 0; i < test.size(); ++i) {

cout << test[i];

}

cout << endl;

}

};

**Question 17.** Say you have an array for which the i-th element is the price of a given stock on day i. Design an algorithm to find the maximum profit. You may complete at most two transactions. Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

**Solution:**

Steps:

1. Use dynamic programming
2. Follow “buy the dips, sell the rips” strategy, divide and conquer
3. For each day i, scan the daily prices, find the maximum profit in day 0 to day i
4. For each day i, scan the daily prices, find the maximum profit in day i to day (n-1)
5. Compare step 3 and step 4, find the maximum profit max(Profit[0,i] + Profit[i, n-1])

// C++

class Solution {

public:

int maxProfit(vector<int> prices) {

if (prices.size() <= 1) {

return 0;

}

vector<int> profit(prices.size());

int buy = 0;

buy = prices[0];

profit[0] = 0;

for (int i = 1; i < prices.size(); i++) {

profit[i] = max(profit[i - 1], prices[i] - buy);

buy = min(buy, prices[i]);

}

int sell = prices[prices.size() - 1];

int best = 0;

for (int i = prices.size() - 2; i >= 0; i--) {

best = max(best, sell - prices[i] + profit[i]);

sell = max(sell, prices[i]);

}

return best;

}

void test1() {

int a[] = {2,3,2,1,4,5,2,11};

vector<int> v\_a(a, a + sizeof(a)/sizeof(int));

int test = maxProfit(v\_a);

cout << test << endl;

}

void test2() {

int a[] = {2};

vector<int> v\_a(a, a + sizeof(a)/sizeof(int));

int test = maxProfit(v\_a);

cout << test << endl;

}

void test3() {

int a[] = {};

vector<int> v\_a(a, a + sizeof(a)/sizeof(int));

int test = maxProfit(v\_a);

cout << test << endl;

}

void test4() {

int a[] = {2,3};

vector<int> v\_a(a, a + sizeof(a)/sizeof(int));

int test = maxProfit(v\_a);

cout << test << endl;

}

void test5() {

int a[] = {3,2};

vector<int> v\_a(a, a + sizeof(a)/sizeof(int));

int test = maxProfit(v\_a);

cout << test << endl;

}

void test6() {

int a[] = {3,3};

vector<int> v\_a(a, a + sizeof(a)/sizeof(int));

int test = maxProfit(v\_a);

cout << test << endl;

}

};