**EX-11 : Fractional Knapsack**

#include <bits/stdc++.h>

using namespace std;

// Returns the value of maximum profit

int knapSackRec(int W, int wt[], int val[], int i, int\*\* dp)

{

    // base condition

    if (i < 0)

        return 0;

    if (dp[i][W] != -1)

        return dp[i][W];

    if (wt[i] > W) {

        // Store the value of function call

        // stack in table before return

        dp[i][W] = knapSackRec(W, wt, val, i - 1, dp);

        return dp[i][W];

    }

    else {

        // Store value in a table before return

        dp[i][W] = max(val[i]

                           + knapSackRec(W - wt[i], wt, val,

                                         i - 1, dp),

                       knapSackRec(W, wt, val, i - 1, dp));

        // Return value of table after storing

        return dp[i][W];

    }

}

int knapSack(int W, int wt[], int val[], int n)

{

    // double pointer to declare the

    // table dynamically

    int\*\* dp;

    dp = new int\*[n];

    // loop to create the table dynamically

    for (int i = 0; i < n; i++)

        dp[i] = new int[W + 1];

    // loop to initially filled the

    // table with -1

    for (int i = 0; i < n; i++)

        for (int j = 0; j < W + 1; j++)

            dp[i][j] = -1;

    return knapSackRec(W, wt, val, n - 1, dp);

}

// Driver Code

int main()

{

    int val[] = { 60, 100, 120 };

    int wt[] = { 10, 20, 30 };

    int W = 50;

    int n = sizeof(val) / sizeof(val[0]);

    cout << knapSack(W, wt, val, n);

    return 0;

}

**EX-12 0/1 Kanpsack**

#include <stdio.h>

int n = 5; /\* The number of objects \*/

int c[10] = {12, 1, 2, 1, 4}; /\* c[i] is the \*COST\* of the ith object; i.e. what

                YOU PAY to take the object \*/

int v[10] = {4, 2, 2, 1, 10}; /\* v[i] is the \*VALUE\* of the ith object; i.e.

                what YOU GET for taking the object \*/

int W = 15; /\* The maximum weight you can take \*/

void simple\_fill() {

    int cur\_w;

    float tot\_v;

    int i, maxi;

    int used[10];

    for (i = 0; i < n; ++i)

        used[i] = 0; /\* I have not used the ith object yet \*/

    cur\_w = W;

    while (cur\_w > 0) { /\* while there's still room\*/

        /\* Find the best object \*/

        maxi = -1;

        for (i = 0; i < n; ++i)

            if ((used[i] == 0) &&

                ((maxi == -1) || ((float)v[i]/c[i] > (float)v[maxi]/c[maxi])))

                maxi = i;

        used[maxi] = 1; /\* mark the maxi-th object as used \*/

        cur\_w -= c[maxi]; /\* with the object in the bag, I can carry less \*/

        tot\_v += v[maxi];

        if (cur\_w >= 0)

            printf("Added object %d (%d$, %dKg) completely in the bag. Space left: %d.\n", maxi + 1, v[maxi], c[maxi], cur\_w);

        else {

            printf("Added %d%% (%d$, %dKg) of object %d in the bag.\n", (int)((1 + (float)cur\_w/c[maxi]) \* 100), v[maxi], c[maxi], maxi + 1);

            tot\_v -= v[maxi];

            tot\_v += (1 + (float)cur\_w/c[maxi]) \* v[maxi];

        }

    }

    printf("Filled the bag with objects worth %.2f$.\n", tot\_v);

}

int main(int argc, char \*argv[]) {

    simple\_fill();

    return 0;

}

**EX-10 reverse the two arrays using another data structure so that the arrays are in ascending order. Hint: Use LIFO**

#include<bits/stdc++.h>

using namespace std;

void reverse\_using\_stack(vector<int>&v){

stack<int>st;

for(auto &x : v){

st.push(x);

}

int itr=0;

while(st.size()){

v[itr++] = st.top();

st.pop();

}

}

void print1d(vector<int>&v){

for(auto &x : v){

cout<<x<<"";

}

cout<<"\n";

}

int main(){

vector<int>v1 = {5,4,3,2,1};

vector<int>v2 = {9,8,7,6,5};

reverse\_using\_stack(v1);

reverse\_using\_stack(v2);

print1d(v1);

print1d(v2);

return 0;

}

**EX-9 Sorting list with O(n2) complexity**

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int val;

Node \*next;

Node(){

this->val = INT\_MIN;

this->next = NULL;

}

Node(int x){

val = x;

this->next = NULL;

}

};

class LL{

Node \*head;

public:

LL(){

this->head = NULL;

}

void insertAtBegin(int x){

Node \*n1 = new Node(x);

n1->next = head;

head = n1;

}

void insertAtEnd(int x){

Node \*n1 = new Node(x);

if(head==NULL){

head=n1;

return ;

}

Node \*temp = head;

while(temp->next){

temp = temp->next;

}

temp->next = n1;

}

void sorting(){

Node \*temp1 = head, \*temp2 = head;

while(temp1->next){

temp2 = temp1->next;

while(temp2){

if(temp2->val < temp1->val){

int temp = temp2->val;

temp2->val = temp1->val;

temp1->val = temp;

}

temp2 = temp2->next;

}

temp1 = temp1->next;

}

}

void printList(){

if(!head){

cout<<"Underflow: No data to print.";

return ;

}

Node \*temp = head;

while(temp){

cout<<temp->val<<"->";

temp = temp->next;

}

}

};

int main(){

LL l1;

l1.insertAtEnd(11); // 11

l1.insertAtEnd(12); // 11 12

l1.insertAtEnd(13); // 11 12 13

l1.insertAtEnd(14); // 11 12 13 14

l1.insertAtBegin(90); // 90 11 12 13 14 15

l1.insertAtBegin(80); // 80 90 11 12 13 14 15

l1.sorting();

l1.printList();cout<<"\n"; // 11->13->14

return 0;

}

**Experiment 8: Without comparing the integer values in an array, sort the array of integers in ascending order.**

#include <bits/stdc++.h>

using namespace std;

void count\_sort(vector<int>&v){

map<int,int>um;

for(auto &x : v){

um[x]++;

}

int itr = 0;

for(auto &x : um){

int num = x.first;

int times = x.second;

while(times--){

v[itr++] = num;

}

}

}

void print1d(vector<int>&v){

for(auto &x : v){

cout<<x<<"";

}

cout<<"\n";

}

int main()

{

vector<int>v = {2,5,3,4,1,6,7,3,5,9};

print1d(v);

count\_sort(v);

print1d(v);

return 0;

}

**Experiment 7 : A set of strings are saved in a data structure. Represent the strings in form a non linear data structure in such a way that the searching takes the minimal time**

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

Node \*alpha[26] = {0};

bool end = false;

bool containsKey(char ch){

return alpha[ch-'a'] != NULL;

}

void setNode(char ch, Node \*node){

alpha[ch-'a'] = node;

}

Node \*getNode(char ch){

return alpha[ch-'a'];

}

void setEnd(){

end = true;

}

bool isEnd(){

return end;

}

};

class Trie{

Node \*root;

public:

Trie(){

root = new Node();

}

void insertWord(string s){

Node \*n1 = root;

int n = s.size();

for(int i=0 ; i<n ; i++){

if( !(n1->containsKey(s[i])) ){

n1->setNode(s[i],new Node());

}

n1 = n1->getNode(s[i]);

}

n1->setEnd();

}

bool searchWord(string s){

int n = s.size();

Node \*n1 = root;

for(int i=0 ; i<n ; i++){

if( !(n1->containsKey(s[i]))){

return false;

}

n1 = n1->getNode(s[i]);

}

return n1->isEnd();

}

};

int main()

{

Trie t1;

t1.insertWord("Hello");

t1.insertWord("Hell");

t1.insertWord("Heed");

t1.insertWord("Heap");

t1.insertWord("Help");

t1.insertWord("Helo");

cout<<t1.searchWord("Hello")<<"\n";

cout<<t1.searchWord("Hell");

return 0;

}

**Experiment 6: Three list are maintained with the integer roll nos of the students. One list contains the roll nos and name of the students, second list contains the roll  nos and the course name (BE, BCA etc) for the student and the third list contains the roll no. and CGPA of the student. Find a way to check the common roll nos in the 3 lists. Note: the 3 lists are sorted by the roll nos.**

#include<bits/stdc++.h>

using namespace std;

int isPresent(vector<int>&arr, int k){

int n = arr.size();

int l=0, r=n-1, ans=0;

while(l<=r){

int m = (l+r)/2;

if(arr[m]==k){

ans = 1;

break;

}else if(arr[m]<k){

l=m+1;

}else{

r=m-1;

}

}

return ans;

}

void print1d(vector<int>&v){

for(auto &x : v){

cout<<x<<" ";

}

cout<<"\n";

}

int main(){

vector<int>v1 = {1,2,3,4,5,8,9};

vector<int>v2 = {2,4,8,9,11,13};

vector<int>v3 = {1,2,3,5,9,12,18,18};

vector<int>ans;

int n1=v1.size(), n2=v2.size(), n3=v3.size();

int n = min({n1,n2,n3});

if(n1==n){

for(int i=0 ; i<n ; i++){

if(isPresent(v2,v1[i]) && isPresent(v3,v1[i])){

ans.push\_back(v1[i]);

}

}

}else if(n2==n){

for(int i=0 ; i<n ; i++){

if(isPresent(v1,v2[i]) && isPresent(v3,v2[i])){

ans.push\_back(v2[i]);

}

}

}else{

for(int i=0 ; i<n ; i++){

if(isPresent(v2,v3[i]) && isPresent(v1,v3[i])){

ans.push\_back(v3[i]);

}

}

}

print1d(ans);

return 0;

}

**Experiment 5: Characters are sorted in a form of a list (non continuous).  We have to check if the list of characters has a given character in it or not.**

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int val;

Node \*next;

Node(){

this->val = INT\_MIN;

this->next = NULL;

}

Node(int x){

val = x;

this->next = NULL;

}

};

class LL{

Node \*head;

public:

LL(){

this->head = NULL;

}

void insertAtBegin(int x){

Node \*n1 = new Node(x);

n1->next = head;

head = n1;

}

void insertAtEnd(int x){

Node \*n1 = new Node(x);

if(head==NULL){

head=n1;

return ;

}

Node \*temp = head;

while(temp->next){

temp = temp->next;

}

temp->next = n1;

}

bool isFound(int x){

Node \*temp = head;

while(temp){

if(temp->val == x){

return true;

}

temp = temp->next;

}

return false;

}

void printList(){

if(!head){

cout<<"Underflow: No data to print.";

return ;

}

Node \*temp = head;

while(temp){

cout<<temp->val<<"->";

temp = temp->next;

}

}

};

int main(){

LL l1,l2,l3;

l1.insertAtEnd(1);

l1.insertAtEnd(5);

l1.insertAtEnd(7);

cout<<(l1.isFound(12) ? "Found\n" : "Not Found\n");

return 0;

}

**EXP 4: functionality of find and replace in a string**

#include<iostream>

using namespace std;

void find\_replace(string s, string& s1, string& s2)

{

string res="";

for(int x=0;x<s.length();x++)

{

int i=0;

if(s[x]==s1[i] && x+s1.length()<=s.length())

{

int y;

for(y=x;y<x+s1.length();y++)

{

if(s[y]!=s1[i])

break;

else

i++;

}

if(y==x+s1.length())

{

res.append(s2);

x=y-1;

}

else

{

res.push\_back(s[x]);

}

}

else

{

res.push\_back(s[x]);

}

}

if(res==s)

cout<<"\nCould not find and replace the entered string";

else

cout<<res;

}

int main()

{

string s,s1,s2;

cout<<"Enter a string: ";

cin>>s;

cout<<"Enter a string to find: ";

cin>>s1;

cout<<"Enter a string to replace: ";

cin>>s2;

find\_replace(s,s1,s2);

return 0;

}

**EXP 3 : Names of persons are saved in a list. Create function to check if the names are palindrome or not.**

#include <bits/stdc++.h>

using namespace std;

/\* Link list node \*/

struct Node

{

string data;

Node\* next;

};

// A utility function to check if str is palindrome

// or not

bool isPalindromeUtil(string str)

{

int length = str.length();

// Match characters from beginning and

// end.

for (int i=0; i<length/2; i++)

if (str[i] != str[length-i-1])

return false;

return true;

}

// Returns true if string formed by linked

// list is palindrome

bool isPalindrome(Node \*node)

{

// Append all nodes to form a string

string str = "";

while (node != NULL)

{

str.append(node->data);

node = node->next;

}

// Check if the formed string is palindrome

return isPalindromeUtil(str);

}

// A utility function to print a given linked list

void printList(Node \*node)

{

while (node != NULL)

{

cout << node->data << " -> ";

node = node->next;

}

printf("NULL\n");

}

/\* Function to create a new node with given data \*/

Node \*newNode(const char \*str)

{

Node \*new\_node = new Node;

new\_node->data = str;

new\_node->next = NULL;

return new\_node;

}

/\* Driver program to test above function\*/

int main()

{

Node \*head = newNode("a");

head->next = newNode("bc");

head->next->next = newNode("d");

head->next->next->next = newNode("dcb");

head->next->next->next->next = newNode("a");

isPalindrome(head)? printf("true\n"):

printf("false\n");

return 0;

}

**Exp2 : Two lists are to be maintained containing integer values. The values are not stored in continuous memory locations. Let one list be 7->1->5 and another list be 5->9->2. Create a third list such that the third list contains the sum of the first two list i.e 517+295 =812  and save it as 2->1>8.**

#include <bits/stdc++.h>

**using** **namespace** std;

// A linked List Node

**class** Node {

**public**:

**int** data;

    Node\* next;

};

**typedef** Node node;

/\* A utility function to insert

a node at the beginning of linked list \*/

**void** push(Node\*\* head\_ref, **int** new\_data)

{

    /\* allocate node \*/

    Node\* new\_node = **new** Node[(**sizeof**(Node))];

    /\* put in the data \*/

    new\_node->data = new\_data;

    /\* link the old list of the new node \*/

    new\_node->next = (\*head\_ref);

    /\* move the head to point to the new node \*/

    (\*head\_ref) = new\_node;

}

/\* A utility function to print linked list \*/

**void** printList(Node\* node)

{

**while** (node != NULL) {

        cout << node->data << " ";

        node = node->next;

    }

    cout << endl;

}

// A utility function to swap two pointers

**void** swapPointer(Node\*\* a, Node\*\* b)

{

    node\* t = \*a;

    \*a = \*b;

    \*b = t;

}

/\* A utility function to get size of linked list \*/

**int** getSize(Node\* node)

{

**int** size = 0;

**while** (node != NULL) {

        node = node->next;

        size++;

    }

**return** size;

}

// Adds two linked lists of same size

// represented by head1 and head2 and returns

// head of the resultant linked list. Carry

// is propagated while returning from the recursion

node\* addSameSize(Node\* head1, Node\* head2, **int**\* carry)

{

    // Since the function assumes linked lists are of same

    // size, check any of the two head pointers

**if** (head1 == NULL)

**return** NULL;

**int** sum;

    // Allocate memory for sum node of current two nodes

    Node\* result = **new** Node[(**sizeof**(Node))];

    // Recursively add remaining nodes and get the carry

    result->next

        = addSameSize(head1->next, head2->next, carry);

    // add digits of current nodes and propagated carry

    sum = head1->data + head2->data + \*carry;

    \*carry = sum / 10;

    sum = sum % 10;

    // Assign the sum to current node of resultant list

    result->data = sum;

**return** result;

}

// This function is called after the

// smaller list is added to the bigger

// lists's sublist of same size. Once the

// right sublist is added, the carry

// must be added toe left side of larger

// list to get the final result.

**void** addCarryToRemaining(Node\* head1, Node\* cur, **int**\* carry,

                         Node\*\* result)

{

**int** sum;

    // If diff. number of nodes are not traversed, add carry

**if** (head1 != cur) {

        addCarryToRemaining(head1->next, cur, carry,

                            result);

        sum = head1->data + \*carry;

        \*carry = sum / 10;

        sum %= 10;

        // add this node to the front of the result

        push(result, sum);

    }

}

// The main function that adds two linked lists

// represented by head1 and head2. The sum of

// two lists is stored in a list referred by result

**void** addList(Node\* head1, Node\* head2, Node\*\* result)

{

    Node\* cur;

    // first list is empty

**if** (head1 == NULL) {

        \*result = head2;

**return**;

    }

    // second list is empty

**else** **if** (head2 == NULL) {

        \*result = head1;

**return**;

    }

**int** size1 = getSize(head1);

**int** size2 = getSize(head2);

**int** carry = 0;

    // Add same size lists

**if** (size1 == size2)

        \*result = addSameSize(head1, head2, &carry);

**else** {

**int** diff = **abs**(size1 - size2);

        // First list should always be larger than second

        // list. If not, swap pointers

**if** (size1 < size2)

            swapPointer(&head1, &head2);

        // move diff. number of nodes in first list

**for** (cur = head1; diff--; cur = cur->next)

            ;

        // get addition of same size lists

        \*result = addSameSize(cur, head2, &carry);

        // get addition of remaining first list and carry

        addCarryToRemaining(head1, cur, &carry, result);

    }

    // if some carry is still there, add a new node to the

    // front of the result list. e.g. 999 and 87

**if** (carry)

        push(result, carry);

}

// Driver code

**int** main()

{

    Node \*head1 = NULL, \*head2 = NULL, \*result = NULL;

**int** arr1[] = { 9, 9, 9 };

**int** arr2[] = { 1, 8 };

**int** size1 = **sizeof**(arr1) / **sizeof**(arr1[0]);

**int** size2 = **sizeof**(arr2) / **sizeof**(arr2[0]);

    // Create first list as 9->9->9

**int** i;

**for** (i = size1 - 1; i >= 0; --i)

        push(&head1, arr1[i]);

    // Create second list as 1->8

**for** (i = size2 - 1; i >= 0; --i)

        push(&head2, arr2[i]);

    addList(head1, head2, &result);

    printList(result);

**return** 0;

}

**Exp1 : Implementation of LL**

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int val;

Node \*next;

Node(){

this->val = INT\_MIN;

this->next = NULL;

}

Node(int x){

val = x;

this->next = NULL;

}

};

class LL{

Node \*head;

public:

LL(){

this->head = NULL;

}

void insertAtBegin(int x){

Node \*n1 = new Node(x);

n1->next = head;

head = n1;

}

void insertAtPos(int x, int pos){

int n = getSize();

if(pos>n || pos<0){return ;}

if(pos==0){insertAtBegin(x); return ;}

Node \*p1 = head;

while(--pos){

p1 = p1->next;

}

Node \*n1 = new Node(x);

n1->next = p1->next;

p1->next = n1;

}

void insertAtEnd(int x){

Node \*n1 = new Node(x);

if(head==NULL){

head=n1;

return ;

}

Node \*temp = head;

while(temp->next){

temp = temp->next;

}

temp->next = n1;

}

void deleteBegin(){

if(!head){

cout<<"Underflow: No data to delete.";

return ;

}

head = head->next;

}

void deletValue(int x){

if(!head){

cout<<"Underflow: No data to delete.";

return ;

}

Node \*prev=NULL , \*curr=head;

bool flag=true;

while(curr){

if(curr->val == x){

flag=false;

Node \*temp = curr;

prev->next = curr->next;

curr->next = prev;

// delete(curr);

// delete(temp);

}

prev = curr;

curr = curr->next;

}

if(flag){

cout<<"There exist no element which is asked to delete.\n";

}else{

cout<<"Deleted\n";

}

}

void deleteAtPos(int x, int pos){

int n = getSize();

if(pos>n || pos<=0){return ;}

if(pos==1){deleteBegin(); return ;}

Node \*p1 = head, \*prev=nullptr;

while(--pos){

prev = p1;

p1 = p1->next;

}

if(p1->val==x){

prev->next = p1->next;

free(p1);

}

}

void deleteEnd(){

if(!head){

cout<<"Underflow: No data to delete.";

return ;

}

Node \*temp = head;

while(temp->next->next){

temp = temp->next;

}

temp->next = NULL;

}

void sorting(){

Node \*temp1 = head, \*temp2 = head;

while(temp1->next){

temp2 = temp1->next;

while(temp2){

if(temp2->val < temp1->val){

int temp = temp2->val;

temp2->val = temp1->val;

temp1->val = temp;

}

temp2 = temp2->next;

}

temp1 = temp1->next;

}

}

void printList(){

if(!head){

cout<<"Underflow: No data to print.";

return ;

}

Node \*temp = head;

while(temp){

cout<<temp->val<<"->";

temp = temp->next;

}

}

Node\* getHead(){return this->head;}

int getSize(){

if(!head)return 0;

if(!head->next)return 1;

Node \*curr = head;

int ans=0;

while(curr){

curr = curr->next;

ans++;

}

return ans;

}

};

int main(){

LL l1;

l1.insertAtEnd(11); //

l1.insertAtEnd(13); //

l1.insertAtEnd(12); //

l1.insertAtEnd(19); //

l1.insertAtEnd(12); //

l1.insertAtEnd(13);

l1.insertAtPos(999,1);

l1.deleteAtPos(12,2);

l1.printList();

return 0;

}