MODEL EXAMINATION

G.MUKESH

192211309

1. #include <iostream>

#include <vector>

#include <string>

using namespace std;

void generateParenthesesHelper(vector<string>& result, string current, int open, int close, int max) {

if (current.length() == max \* 2) {

result.push\_back(current);

return;

}

if (open < max) {

generateParenthesesHelper(result, current + "(", open + 1, close, max);

}

if (close < open) {

generateParenthesesHelper(result, current + ")", open, close + 1, max);

}

}

vector<string> generateParentheses(int n) {

vector<string> result;

generateParenthesesHelper(result, "", 0, 0, n);

return result;

}

int main() {

int n = 3;

vector<string> combinations = generateParentheses(n);

cout << "Combinations of well-formed parentheses for n=" << n << ":" << endl;

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

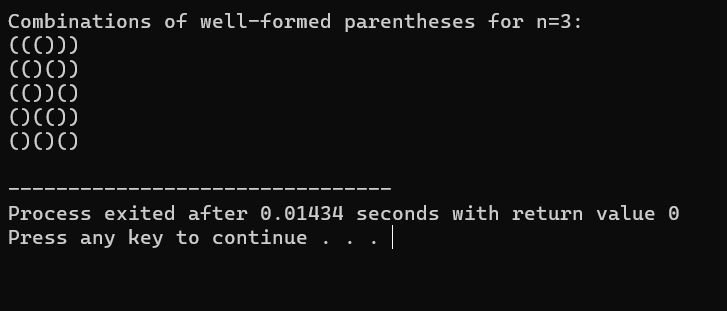
cout << combinations[i] << endl;

}

return 0;

}

OUTPUT:



2. #include<iostream>

using namespace std;

int main(){

int x,p=0;

cout<<"Enter the number";

cin>>x;

int n=x;

while (x!=0){

int a=x%10;

p=p\*10 + a;

x=x/10;

}

if(n==p){

return true;

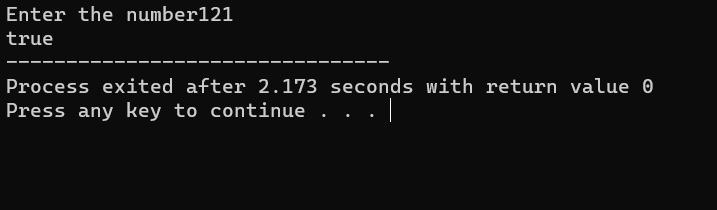
}

else{

return false;

}

}



3. #include<iostream>

using namespace std;

int main(){

int x,p=0;

cout<<"Enter the number";

cin>>x;

int n=x;

while (x!=0){

int a=x%10;

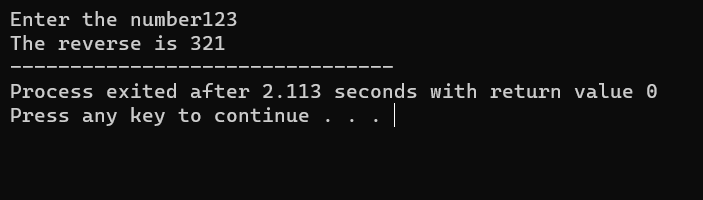
p=p\*10 + a;

x=x/10;

}

cout<<"The reverse is "<<p;

}



4. #include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

int m = nums1.size();

int n = nums2.size();

vector<int> merged(m + n);

int i = 0, j = 0, k = 0;

// Merge the two sorted arrays

while (i < m && j < n) {

if (nums1[i] <= nums2[j]) {

merged[k++] = nums1[i++];

} else {

merged[k++] = nums2[j++];

}

}

while (i < m) {

merged[k++] = nums1[i++];

}

while (j < n) {

merged[k++] = nums2[j++];

}

// Calculate median

int mid = (m + n) / 2;

if ((m + n) % 2 == 0) {

return (double)(merged[mid - 1] + merged[mid]) / 2.0;

} else {

return merged[mid];

}

}

int main() {

int arr1[] = {1, 3};

int arr2[] = {2};

vector<int> nums1(arr1, arr1 + sizeof(arr1) / sizeof(arr1[0]));

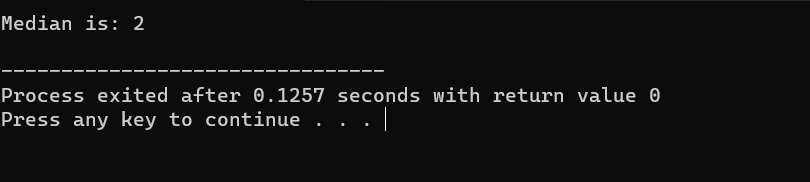
vector<int> nums2(arr2, arr2 + sizeof(arr2) / sizeof(arr2[0]));

double median = findMedianSortedArrays(nums1, nums2);

cout << "Median is: " << median << endl;

return 0;

}



5. #include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<vector<int> > threeSum(vector<int>& nums) {

vector<vector<int> > result;

sort(nums.begin(), nums.end());

int n = nums.size();

for (int i = 0; i < n - 2; ++i) {

if (i == 0 || (i > 0 && nums[i] != nums[i - 1])) {

int target = -nums[i];

int left = i + 1;

int right = n - 1;

while (left < right) {

int sum = nums[left] + nums[right];

if (sum == target) {

vector<int> triplet;

triplet.push\_back(nums[i]);

triplet.push\_back(nums[left]);

triplet.push\_back(nums[right]);

result.push\_back(triplet);

// Skip duplicates

while (left < right && nums[left] == nums[left + 1]) left++;

while (left < right && nums[right] == nums[right - 1]) right--;

left++;

right--;

} else if (sum < target) {

left++;

} else {

right--;

}

}

}

}

return result;

}

int main() {

int arr[] = {-1, 0, 1, 2, -1, -4};

vector<int> nums(arr, arr + sizeof(arr) / sizeof(arr[0]));

vector<vector<int> > triplets = threeSum(nums);

cout << "Triplets with sum 0:" << endl;

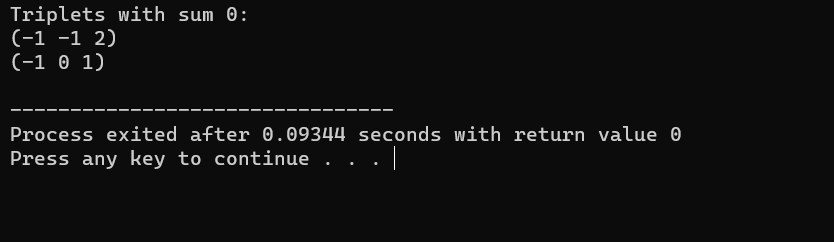
for (size\_t i = 0; i < triplets.size(); ++i) {

cout << "(" << triplets[i][0] << " " << triplets[i][1] << " " << triplets[i][2] << ")" << endl;

}

return 0;

}



6. #include <iostream>

#include <vector>

#include <algorithm>

#include <climits>

#include <cmath>

using namespace std;

int main() {

int arr[] = {-1, 2, 1, -4};

vector<int> nums(arr, arr + sizeof(arr) / sizeof(arr[0]));

int target = 1;

sort(nums.begin(), nums.end());

int closestSum = INT\_MAX;

int minDiff = INT\_MAX;

int n = nums.size();

for (int i = 0; i < n - 2; ++i) {

int left = i + 1;

int right = n - 1;

while (left < right) {

int sum = nums[i] + nums[left] + nums[right];

int diff = abs(sum - target);

if (diff < minDiff) {

minDiff = diff;

closestSum = sum;

}

if (sum < target) {

left++;

} else {

right--;

}

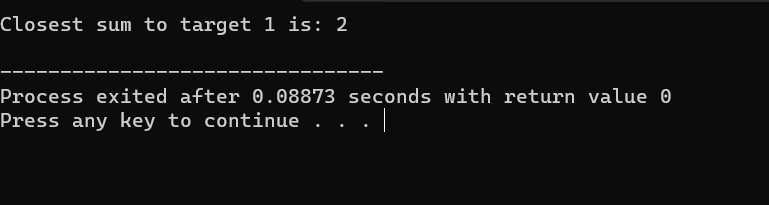
}

}

cout << "Closest sum to target " << target << " is: " << closestSum << endl;

return 0;

}



7. #include <iostream>

#include <string>

#include <algorithm>

using namespace std;

void printLongestSubstring(const string& s) {

int maxLength = 0;

int start = 0;

int end = 0;

int n = s.length();

int lastIndex[256] = { -1 };

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

if (lastIndex[s[i]] >= start) {

start = lastIndex[s[i]] + 1;

}

lastIndex[s[i]] = i;

if (i - start + 1 > maxLength) {

maxLength = i - start + 1;

end = i;

}

}

cout << "Longest substring without repeating characters: ";

for (int i = start; i <= end; ++i) {

cout << s[i];

}

cout << endl;

}

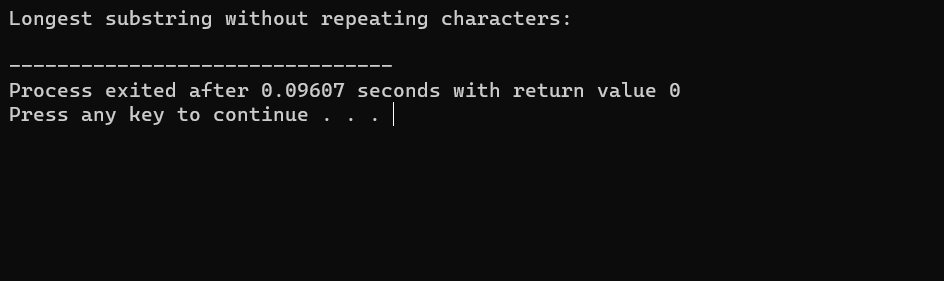
int main() {

string s = "abcabcbb";

printLongestSubstring(s);

return 0;

}



8. #include <iostream>

#include <string>

using namespace std;

bool isMatch(const char \*s, const char \*p) {

if (\*p == '\0') return \*s == '\0';

bool firstMatch = (\*s != '\0' && (\*s == \*p || \*p == '.'));

if (\*(p + 1) == '\*') {

return isMatch(s, p + 2) || (firstMatch && isMatch(s + 1, p));

} else {

return firstMatch && isMatch(s + 1, p + 1);

}

}

int main() {

const char \*s = "aa";

const char \*p = "a\*";

bool result = isMatch(s, p);

cout << "Matching: " << (result ? "true" : "false") << endl;

return 0;

}

