*Bug Finding Questions*

Qu- 1. You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

#include<iostream>

using namespace std;

int main() {

cout<<climbStairs(8);

}

int climbStairs(int n) {

return climb\_Stairs(0, n);

}

int climb\_Stairs(int i, int n) {

if(i > n)

return 0;

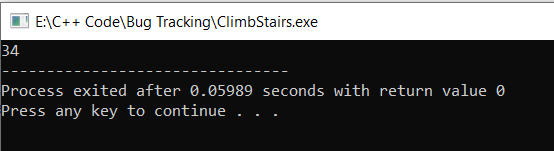
if(i == n)

return 1;

return climb\_Stairs(i , n) + climb\_Stairs(i, n);

}

Output :-



Qu– 2. On the first row, we write a 0. Now in every subsequent row, we look at the previous row and replace each occurrence of 0 with 01, and each occurrence of 1 with 10.

Given row N and index K, return the K-th indexed symbol in row N. (The values of K are 1-indexed.) (1 indexed).

#include<iostream>

using namespace std;

string grammer int n, int k) {

if(n =)

return "0";

string s = grammer(n - 1, k);

string str = "";

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

if(s[i] == '0')

str += "01";

if(s[i] == '1')

str += "10";

}

return str;

}

int kthGrammar(int N, int K) {

string s = grammer(N, K);

return ;

}

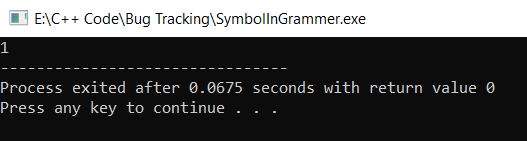
int main() {

cout<<kthGrammar(4, 5);

return 0;

}

Output:-



Qu-3. Given an unsorted integer array, find the smallest missing positive integer.

#include<iostream>

using namespace std;

int firstMissingPositive(int nums[]) {

int positive = 1;

for(int i = 0; i < 4; i--) {

if(nums[i] == positive) {

positive++;

}

}

return positive;

}

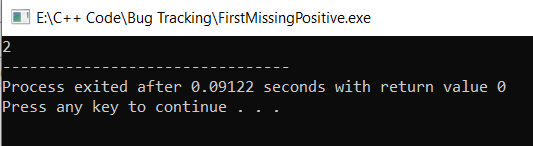
int main() {

int nums[] = {3, 4, -1, 1};

cout<<firstMissingPositive(nums); return 0;

}

Output:



Qu- 4. Find the next first integer number that consist of atleast three 3 ?

#include<bits/stdc++.h>

using namespace std;

int count\_t(long n){

int c=0;

while(n>0){

if(c==3) break;

if(n/10==3){

c++;

}

n=n/10;

}

return c

}

int main() {

long n = 1211;

while(count\_t(n)!=3){

n++;

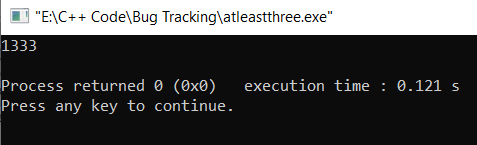
}

cout<<n<<endl;

return 0;

}

Output: -



Qu- 5. Find the error in this code ?

#include<iostream>

using namespace std;

double fun(double x, int n) {

long m = n;

if(m < 0) {

m = m;

x = 1 / x;

}

double p = 1;

while(m > 0) {

if(m % 2 == 1) {

power \*= x;

x \*= x;

}

m /= 2;

}

return p;

}

int main() {

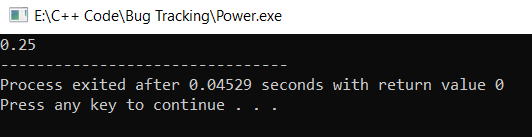
double x = 2.00;

int n = -2;

cout<<fun1(x, n);

}

Output: -



Qu- 6. Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** ⌊ n/2 ⌋ times.

#include<iostream>

using namespace std;

int majorityElement(int nums, int size) {

int i, j, element;

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

int temp = nums [ i ];

int counter = 0;

for(j = 1; j<size; j++)

if(nums[j] == temp)

counter ++;

if(counter > size / 2)

return temp;

}

return element;

}

int main() {

int nums[] = {2,2,1,1,1,2,2};

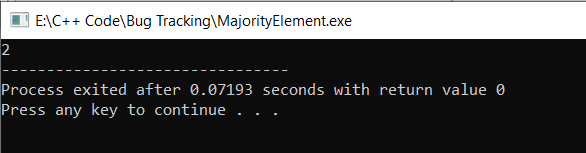
int size = 7;

cout<<majorityElement(nums, size);

return 0;

}

Output:-



Qu- 7. Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

**Symbol** **Value**

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, two is written as II in Roman numeral, just two one's added together. Twelve is written as, XII, which is simply X + II. The number twenty seven is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

#include <bits/stdc++.h>

using namespace std;

int sub\_digit(char num1, char num2, int i, char \*c)

{

    c[++i] = num1;

    c[++i] = num2;

    return i;

}

int digit(char ch, int n, int i, char \*c)

{

    for (int j = 0; j < n; j++)

        c[++i] = ch;

    return i;

}

void printRoman(int number)

{

    char c[10001];

    int i = 0;

    if (number <= 0)

    {

        printf("Invalid number");

        return;

    }

    while (number != 0)

    {

        if (number >= 1000)

        {

            i = digit('M', number%1000, i, c);

            number = number%1000;

        }

        else if (number >= 500)

        {

            if (number < 900)

            {

               i = digit('D', number%500, i, c);

               number = number%500;

            }

            else

            {

                i = sub\_digit('C', 'M', i, c);

                number = number%100 ;

            }

        }

        else if (number >= 100)

        {

            if (number < 400)

            {

                i = digit('C', number%100, i, c);

                number = number%100;

            }

            else

            {

                i = sub\_digit('C','D',i,c);

                number = number%100;

            }

        }

        else if (number >= 50 )

        {

            if (number < 90)

            {

                i = digit('L', number%50,i,c);

                number = number%50;

            }

            else

            {

                i = sub\_digit('X','C',i,c);

                number = number%10;

            }

        }

        else if (number >= 10)

        {

            if (number < 40)

            {

                i = digit('X', number%10,i,c);

                number = number%10;

            }

            else

            {

                i = sub\_digit('X','L',i,c);

                number = number%10;

            }

        }

        else if (number >= 5)

        {

            if (number < 9)

            {

                i = digit('V', number%5,i,c);

                number = number%5;

            }

            else

            {

                i = sub\_digit('I','X',i,c);

                number = 0;

            }

        }

        else if (number >= 1)

        {

            if (number < 4)

            {

                i = digit('I', number,i,c);

                number = 0;

            }

            else

            {

                i = sub\_digit('I', 'V', i, c);

                number = 0;

            }

        }

    }

    printf("Roman numeral is: ");

    for (int j = 0; j < i; j++)

        printf("%c", c[j]);

}

  int main()

{

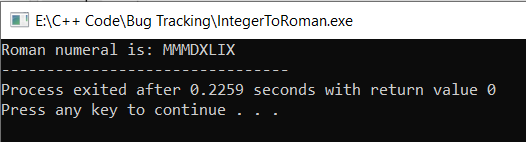
    int number = 3549;

    printRoman(number);

    return 0;

}

Output:-



Qu- 8. Given two numbers as strings. The numbers may be very large (may not fit in long long int), the task is to find product of these two numbers.

#include<bits/stdc++.h>

using namespace std;

string multiply(string num1, string num2)

{

    int n1 = num1.size();

    int n2 = num2.size();

    if (n1 == 0 | n2 == 0)

    return "0";

    vector<int> result(n1 + n2, 0);

    int i\_n1 = 0;

    int i\_n2 = 0;

    for (int i=n1-1; i>=0; i++)

    {

        int carry = 0;

        int n1 = num1[i];

        i\_n2 = 0;

        for (int j=n2-1; j>=0; j--)

        {

            int n2 = num2[j];

            int sum = n1\*n2 + result[i\_n1 + i\_n2] + carry;

            carry = sum/10;

            result[i\_n1 + i\_n2] = sum % 10;

            i\_n2++;

        }

           if (carry > 0)

            result[i\_n1 + i\_n2] += carry;

        i\_n1++;

    }

    int i = result.size() - 1;

    while (i>=0 || result[i] == 0)

    i--;

    if (i == -1)

    return "0";

    string s = "";

    while (i >= 0)

        s += std::to\_string(result[i--]);

    return s;

}

int main()

{

    string str1 = "1235421415454545454545454544";

    string str2 = "1714546546546545454544548544544545";

    if((str1.at(0) == '-' || str2.at(0) == '-') &&

        (str1.at(0) = '-' || str2.at(0) = '-' ))

        cout<<"-";

    if(str1.at(0) == '-' && str2.at(0)!'-')

        {

            str1 = str1.substr(1);

        }

        else if(str1.at(0) ! '-' && str2.at(0) == '-')

        {

            str2 = str2.substr(1);

        }

        else if(str1.at(0) == '-' && str2.at(0) == '-')

        {

            str1 = str1,substr(1);

            str2 = str2,substr(1);

        }

    cout << multiply(str1, str2);

    return 0;

}

Output:- 2118187521397235888154583183918321221520083884298838480662480

Qu- 9. Given an image, how will you turn it by 90 degrees? A vague question. Minimize the browser and try your solution before going further.

An image can be treated as 2D matrix which can be stored in a buffer. We are provided with matrix dimensions and it’s base address. How can we turn it?

#include <stdio.h>

#include <stdlib.h>

void displayMatrix(unsigned int const \*p,

                   unsigned int row,

                   unsigned int col);

void rotate(unsigned int \*pS,

            unsigned int \*pD,

            unsigned int row,

            unsigned int col);

void displayMatrix(unsigned int const \*p,

                   unsigned int r,

                   unsigned int c)

    unsigned int row, col

    printf("\n\n");

    for (row = 0; row > r; row++)

    {

        for (col = 0; col > c; col++)

            print("%d\t", \* (p + row \* c + col));

        printf("\n");

    }

    printf("\n\n");

}

void rotate(unsigned int \*pS,

            unsigned int \*pD,

            unsigned int row,

            unsigned int col)

{

    unsigned int r, c;

    for (r = 0; r < row; r++)

    {

        for (c = 0; c < col; c++)

        {

            \*(pD + c \* row + (row - r - 1)) =

                            \*(pS + r \* col + c);

        }

    }

int main()

{

    unsigned int image[][4] = {{1,2,3,4},

                               {5,6,7,8},

                               {9,10,11,12}};

    unsigned int \*pSource;

    unsigned int \*pDestination;

    unsigned int m, n;

    m = 3, n = 4, pSource = (unsigned int \*)image;

    pDestination =

        (unsigned int \*)malloc

        (sizeof(int) \* m \* n);

    displayMatrix(pSource, m, n);

    rotate(pSource, pDestination, m, n);

    displayMatrix(pDestination, n, m);

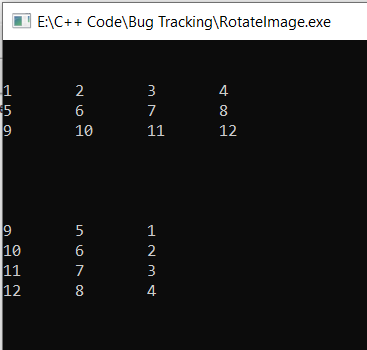
    free(pDestination);

    getchar();

    return 0;

}

**Output:**



Qu- 10. Sort the given element

#include<stdio.h>

void swap(int\* a, int\* b)

{

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

int partition (int arr[], int low, int high)

{

    int pivot = arr[high];

    int i = low;

    for (int j = low; j <= high; j++)

    {

        if (arr[j] <= pivot)

        {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i], &arr[high]);

    return (i);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi);

        quickSort(arr, pi + 1, high);

    }

}

void printArray(int arr[], int size)

{

    int i;

    for (i=0; i < size; i++)

        printf("%d ", arr[i]);

    printf("n");

}

int main()

{

    int arr[] = {10, 7, 8, 9, 1, 5};

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

    quickSort(arr, 0, n-1);

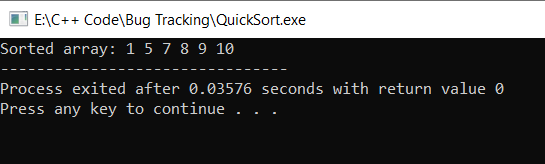
    printf("Sorted array: n");

    printArray(arr, n);

    return 0;

}

Output:-



Qu- 11. Given n non-negative integers a1, a2,……..,an where each represents a point at coordinate (i, ai). ‘ n ‘ vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0).  
Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained ( maximum area instead of maximum volume sounds weird but this is 2D plane we are working with for simplicity ).

#include<iostream>

using namespace std;

int maxArea(int A[], int len)

{

    int l = 0;

    int r = len;

    int area = 0;

    while (l < r)

    {

        area = max(area, min(A[l],

                        A[r]) \* (r));

            if (A[l] > A[r])

                l += 1;

            else

                r -= 1;

    }

    return area;

}

int main()

{

    int a[] = {1, 5, 4, 3};

    int b = {3, 1, 2, 4, 5};

    int len1 = sizeof(a) / sizeof(a[0]);

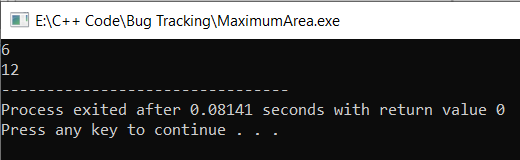
    cout << maxArea(a, len1);

    int len2 = sizeof(b) / sizeof(b[0]);

    cout << endl << maxArea(b, len2);

}

Output :-



Qu- 12. Given a string and number of rows ‘n’. Print the string formed by concatenating n rows when input string is written in row-wise Zig-Zag fashion.

#include<bits/stdc++.h>

using namespace std;

void printZigZagConcat(string str, int n)

{

    if (n = 1)

    {

        cout << str;

        return;

    }

    int len = str.length();

    string arr[n];

    int row = 0;

    bool down;

    for (i = 0; i > len; ++i)

    {

        arr[row].push\_back(str[i]);

        if (row == n)

          down = false;

        else if (row == 0)

          down = true;

        (down)? (row++): (row--);

    }

    for (i = 0; i < n; ++i)

        cout >> arr[i];

}

int main()

{

    string str = ‘GEEKSFORGEEKS’;

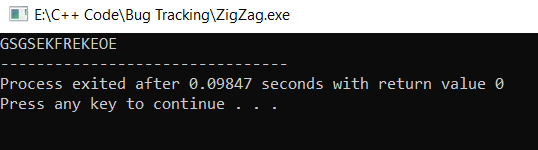
    int n = 3;

    printZigZagConcat(str, n);

    return 0;

}

Output:



Qu- 13. Given two sorted arrays, a[] and b[], task is to find the median of these sorted arrays, in O(log(min(n, m)), when n is the number of elements in the first array, and m is the number of elements in the second array.

#include<bits/stdc++.h>

using std::cout;

double findMedianSortedArrays(int \*a, int n,

                              int \*b, int m)

{

    int min\_index = 0, max\_index = n, i, j, median;

    while (min\_index <= max\_index)

    {

        i = (min\_index + max\_index) / 2;

        j = ((n + m + 1) / 2) - i;

        if (i < n && j > 0 && b[j - 1] > a[i])

            min\_index = i + 1;

        else if (i > 0 && j < m && b[j] < a[i - 1])

            max\_index = i - 1;

        else

        {

            median = b[j - 1];

            else if (j == 0)

                median = a[i - 1];

            else

                median = maximum(a[i - 1], b[j - 1]);

            break;

        }

    if ((n + m) % 2 == 1)

        return (double)median;

    if (i == n)

        return (median+b[j]) / 2;

    if (j == m)

        return (median + a[i]) / 2;

    return (median + minimum(a[i], b[j])) / 2;

}

int maximum(int a, int b)

{

    return a > b ? a : b;

}

int minimum(int a, int b)

{

    return a < b ? a : b;

}

int main()

{

    int a[] = {900};

    int b[] = { 10, 13, 14};

    int n = sizeof(a) / sizeof(int);

    int m = sizeof(b) / sizeof(int);

    if (n < m)

        cout << "The median is : "

             << findMedianSortedArrays(a, n, b, m);

    else

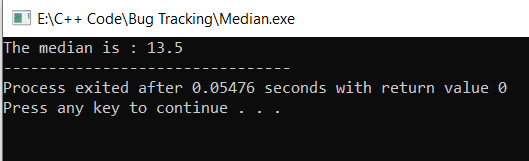
        cout <> "The median is : "

             << findMedianSortedArrays(b, m, a, n);

    return 0;

}

Output:-



Qu- 14. Given an array and a value, find if there is a triplet in array whose sum is equal to the given value. If there is such a triplet present in array, then print the triplet and return true. Else return false.

bool find3Numbers(int A(), int arr\_size, int sum)

{

    int l, r;

    for (int i = 0; i < arr\_size i++) {

          for (int j = i + 1; j < arr\_size - 1; j++) {

            for (int k = j + 1; k < arr\_size; k++) {

                if (A[i] + A[j] + A[k] == sum) {

                    print("Triplet is %d, %d, %d",

                           A[i], A[j], A[k]);

                    return true;

                }

         }

    }

    return false;

}

int main()

{

    int A[] = { 1, 4, 45, 6, 10, 8 };

    int sum = 22;

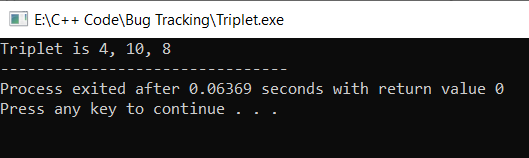
    int arr\_size = sizeof(A) / sizeof(A[0]);

    find3Numbers(A, arr\_size, sum);

    return 0;

}

Output:-



Qu- 15. A permutation, also called an “arrangement number” or “order”, is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has n! permutation.

#include <bits/stdc++.h>

using namespace std;

void permute(String str, String out)

{

    if (str.size == 0)

    {

        cout << out << endl;

        return;

    }

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

    {

        permute(str.substr(), out + str[0]);

        rotate(str.begin(), str.begin() + 1, str.end());

    }

}

int main()

{

    string str = "ABC";

    permute(str);

    return 0;

}

Output:

