**\* Programming.**

**Question 10.** Explain the following code: const int\* const fun(const int\* const& p) const;

**Solution:**

1. This is a constant member function.

It cannot modify any non-static members or call any member functions that are not constant.

2. This function returns a constant pointer pointing to a constant int.

3. This function takes a reference to a constant pointer pointing to a constant int.

**Question 12.** Implement the interface for matrix class in C++.

**Solution:**

// C++

// Reference: https://www.quantstart.com/articles/Matrix-Classes-in-C-The-Header-File

template <typename T> class QSMatrix {

private:

std::vector<std::vector<T> > mat;

unsigned rows;

unsigned cols;

public:

QSMatrix(unsigned \_rows, unsigned \_cols, const T& \_initial);

QSMatrix(const QSMatrix<T>& rhs);

virtual ~QSMatrix();

// Operator overloading, for "standard" mathematical matrix operations

QSMatrix<T>& operator=(const QSMatrix<T>& rhs);

// Matrix mathematical operations

QSMatrix<T> operator+(const QSMatrix<T>& rhs);

QSMatrix<T>& operator+=(const QSMatrix<T>& rhs);

QSMatrix<T> operator-(const QSMatrix<T>& rhs);

QSMatrix<T>& operator-=(const QSMatrix<T>& rhs);

QSMatrix<T> operator\*(const QSMatrix<T>& rhs);

QSMatrix<T>& operator\*=(const QSMatrix<T>& rhs);

QSMatrix<T> transpose();

// Matrix/scalar operations

QSMatrix<T> operator+(const T& rhs);

QSMatrix<T> operator-(const T& rhs);

QSMatrix<T> operator\*(const T& rhs);

QSMatrix<T> operator/(const T& rhs);

// Matrix/vector operations

std::vector<T> operator\*(const std::vector<T>& rhs);

std::vector<T> diag\_vec();

// Access the individual elements

T& operator()(const unsigned& row, const unsigned& col);

const T& operator()(const unsigned& row, const unsigned& col) const;

// Access the row and column sizes

unsigned get\_rows() const;

unsigned get\_cols() const;

};

**Question 14.** Is it okay for a non-virtual function of the base class to call a virtual function?

**Solution:**

//Reference: https://isocpp.org/wiki/faq/strange-inheritance#calling-virtuals-from-base

Yes. It is allowed. If all derived classes have the same algorithm for method A. Method A needs to call method B. However, each derived class has their own algorithm to implement method B. In this case, method A could be non-virtual while method B should be virtual function.

**Question 16.** How to inverse a string of sentence (without reverse the word)

**Solution:**

Steps:

1. Use a pointer to iterator the whole sentence from the last characters
2. If the pointer encounters white space, continues to find a normal character and denote it as the end of a word
3. If the pointer encounters a normal character, continues to find a white space and denote it as the beginning of a word.
4. Append this word to new sentence.
5. Time complexity is O(n). Space complexity is O(1).

// Java

public class Solution {

public static String reverse(String test)

{

StringBuilder result = new StringBuilder();

for(int i = test.length() - 1; i>= 0;)

{

while(i >= 0 && test.charAt(i) == ' ') i--;

int end\_word = i;

while(i >= 0 && test.charAt(i) != ' ') i--;

int begin\_word = i;

result.append(test.substring(begin\_word + 1, end\_word + 1) + " ");

}

return result.toString().trim();

}

public static void main(String[] args)

{

System.out.println(reverse(" qwe sdoad rjowq djfaois "));

}

}

**Question 18.** The book problem: There is a group of N (2 ≤ N ≤ 1000) people which are numbered 1 through N, and everyone of them has not less than the book, which others want to read. Write the program which finds a way of transferring the book so that it will visit every man only once, passing from the friend to the friend, and, at last, has come back to the owner. Note: if A is a friend of B then B is a friend of A. INPUT: First line of input contains number N. Next N lines contain information about friendships. (i+ 1)-th line of input contains a list of friends of i-th man. OUTPUT: If there is no solution then your program must output ’No solution’. Else your program must output exactly N + 1 number: this sequence should begin and should come to end by number 1, any two neighbors in it should be friends, and any two elements in it, except for the first and last, should not repeat.

**Solution:**

Steps:

1. This is a Hamilton Cycle problem. Use backtracking(DFS) algorithm.

2. Use an array *path* to store the current way to transfer books.

3. When the length of the path is N, it means the book has been passed to the last people, then we need to check if the last people is the friend of the first people. If yes, then we find a way. If not, then the current path is not a good way, we should go back and try another way.

4. For each people, check all its friends that is not already in path to verify if it should be included.

5.Time complexity is O(2^N).

//Java

import java.io.\*;

import java.util.\*;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

import java.util.stream.Stream;

public class Passingbook {

public static void findfriendpass(int N, HashMap<Integer, ArrayList<Integer>> friends)

{

int[] path = new int[N];

path[0] = 1;

if(findfriendpass\_aux(N, friends, path, 1))

{

for(int people:path)

System.out.println(people);

System.out.println(1);

}

else

System.out.println("No Solution");

}

public static boolean findfriendpass\_aux(int N, HashMap<Integer, ArrayList<Integer>> friends, int[] path, int pos)

{

if(pos == N)

{

if(friends.get(path[pos - 1]).contains(1))

return true;

else

return false;

}

for(int v = 1; v <= N; v++)

{

if(isOKtoAdd(v, friends,path,pos))

{

path[pos] = v;

if(findfriendpass\_aux(N, friends, path,pos + 1))

return true;

path[pos] = 0;

}

}

return false;

}

public static boolean isOKtoAdd(int v, HashMap<Integer,ArrayList<Integer>> friends, int[] path, int pos)

{

if(!friends.get(path[pos - 1]).contains(v))

return false;

for(int item:path)

if(item == v)

return false;

return true;

}

public static void main(String[] args)

{

HashMap<Integer, ArrayList<Integer>> friends = new HashMap<Integer, ArrayList<Integer>>();

try {

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("input.txt")));

int N = Integer.parseInt(br.readLine());

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

{

IntStream friend\_intstream = Stream.of(br.readLine().split(" ")).mapToInt(Integer::parseInt);

ArrayList<Integer> friend = (ArrayList<Integer>) friend\_intstream.boxed().collect(Collectors.toList());

friends.put(i, friend);

}

findfriendpass(N, friends);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}