CS101: Spring 2018 End-Semester Exam

25th April 2018, 17.30 to 20.30 hrs

Name: _____

Roll number: _____ Division: ____ Hall: _____

Lab Group (Eg: Wed-SL2-2): _____

This examination contains 18 pages (including this cover page and a page for rough work) and 16 questions (including an optional challenge question). Maximum possible score is 100 points. Please ensure all the pages are printed clearly before proceeding.
Write all your answers in the space provided. Do not write irrelevant answers. Answers must be written only in pen (not pencil). You are allowed extra sheet only under special circumstances like total cancellation of a previously written answer. Write your roll number on all pages including any extra sheet you may use.

DO NOT WRITE IN THE TABLE BELOW

Q #	Points	Max	Grading TA	Remarks
1		6		
2		3		
3		6		
4		4		
5		5		
6		5		
7		8		
8		4		
9		4		
10		8		
11		6		
12		6		
13		7		
14		8		
15		20		
Challenge		-		

Roll number:_____

1. *[6 points]* (1.5 points each for any 4 errors mentioned correctly - 0.5 point for line number, 1 point for correct reason for that line)

```
01.  #include <simplecpp>
02.  main_program {
03.      char msg[] = "Hello";
04.      char * p1 = msg;
05.      const char * p2;
06.      char * const p3;
07.      const char * const p4 = msg;
08.      p2 = p1;
09.      p2[1] = 'a';
10.      p3 = p2;
11.      p4[4] = 'a';
12.      p4 = p1;
13.  }
```

Point out the compiler errors in the above program with line number and short error descriptions.

Line 6: uninitialized const ‘p3’
Line 9: assignment of read-only location ‘p2[1]’
Line 10: assignment of read-only variable ‘p3’
Line 10: invalid conversion from ‘const char*’ to ‘char*’
Line 11: assignment of read-only location ‘p4[4]’
Line 12: assignment of read-only variable ‘p4’

1.5 * 4 marks

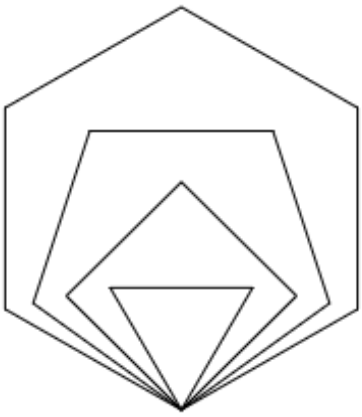
2. *[3 points]*

```
#include <simplecpp>
struct String {
  char* str;
  String(char* s) { str = s; }
  void print() { cout << str << " "; }
};
main_program {
  char defStr[] = "MA106";
  String a(defStr), b = a;
  b.str[1] = 'H';
  b.str[0] = 'P';
  b.str[4] = '8';
  a.print();
  b.print();
}
```

Predict the output of the above program: PH108 PH108 **1.5 + 1.5 marks**

Roll number:_____

3. *[6 points]*
Complete the code snippet so that the following figure can be obtained as output.



```
#include <simplecpp>
main_program {
  turtleSim();
  int side = 100, x = 6;
  left(30); // To make the outer hexagon vertical
  for(int i = 0; i < 4 ; i++) {                          2 marks
    repeat(x) { // Draw a polygon of x sides
      forward(side);
      left(360/x);                                       1 mark
    }
    side -= 10;
    left(180/(x*x-x));                                   3 marks
    // 180/(x-1) - 180/x also accepted as correct answer
    x--; // Now move to drawing inner polygons
  }
}
```

4. *[4 points]* (0 -> 1 point, 0 1 0 -> 2 points, 0 1 0 2 0 -> 3 points, 0 1 0 2 0 1 0 -> 4 points)

```
#include <simplecpp>
void f(int x) {
  if(x > 0) {
    f(--x);
    cout << x << " ";
    f(x--);
  }
}
main_program {
  int a = 3;
  f(a);
}
```

Predict the output of the above program: **0 1 0 2 0 1 0** **4 marks**

Roll number:_____

5. *[5 points]*
Two words **a** and **b** are said to be anagrams of each other if word **b** can be formed by rearranging letters of word **a**. For example, “pan” and “nap” are anagrams, and so are “listen” and “silent”. For this problem we will ignore the case of the letters, i.e. we consider capital and small letters to be same. Thus “listen” is an anagram of “LISTEN”, “sILeNt” , “SILent” etc. Fill the function below which maintains a count of the number of different letters in the first string and then checks them with the other string. If any of the strings contain any character other than a letter, the function returns -1.

Reference for **vector** class:
Vectors are sequence containers representing arrays that can change in size.
Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. Further, elements of a vector can be accessed using the [] operator similar to accessing elements in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Member function **size()**: Returns the number of elements in the vector.
Member function **resize(unsigned int n)**: Resizes the vector so that it contains n elements. All elements are initialized by default to 0.

```
int isAnagram(vector<char> str1, vector<char> str2) {
  vector<int> cnt; // Array of counts for alphabets
  cnt.resize(26); // Resize cnt to size 26                    1 mark
  for (int i = 0; i < str1.size(); i++) {
    if (str1[i] >= 'a' && str1[i] <= 'z')
      Cnt[str1[i]-'a']++;                                    0.5 mark
    else if (str1[i] >= 'A' && str1[i] <= 'Z')
      cnt[str1[i]-'A']++;                                    0.5 mark
    else
       return -1;
  }
  for (int i = 0; i < str2.size(); i++) {
    if (str2[i] >= 'a' && str2[i] <= 'z')
      Cnt[str1[i]-'a']--;                                    1 mark
    else if (str2[i] >= 'A' && str2[i] <= 'Z')
      cnt[str1[i]-'A']--;                                    1 mark
    else
      return -1;
  }
  for(int i = 0; i < 26; i++) {
// Check if both strings contain same number of the ith alphabet
    if (cnt[i] != 0)                                         1 mark
      return 0; // str1 and str2 are not anagrams
  }
  return 1; // str1 and str2 are anagrams of each other
}
```

Roll number:_____

6. *[5 points]* (1.5 points each the 2 errors mentioned correctly - 0.5 point for line number, 1 point for correct reason for that line) (1 point each for the changes suggested)

```
01.  #include <simplecpp>
02.  struct V3 {
03.    int x, y, z;
04.    V3(int _x, int _y, int _z) {
05.      x = _x; y = _y; z = _z;
06.    }
07.    V3(const V3& v) {
08.      x = v.x; y = v.y; z = v.z;
09.    }
10.    V3 operator+(V3 &v) {
11.      return V3(x+v.x, y+v.y, z+v.z);
12.    }
13.    V3 operator*(V3 &v) {
14.      return V3(x*v.x, y*v.y, z*v.z);
15.    }
16.  };
17.  main_program {
18.    V3 a(1, 2, 3), b(4, 5, 6), c(2, 3, 4), d(3, 4, 5);
19.    V3 ans;
20.    ans = (a + b) * (c + d);
21.  }
```

Point out the compiler errors in the above program with line number and short error descriptions. What should be added/modified to correct these errors?

1. Line 19: no matching function for call to V3::V3() or default constructor not defined
2. Line 20: invalid initialization of non-const reference of type ‘V3&’ from an rvalue of type ‘V3’

Suggested changes:
Error 1:
A default constructor must be added to the struct V3.
Error 2:
Change `V3 operator*(V3 &v) to V3 operator*(V3 v)`
Or
Change `V3 operator*(V3 &v) to V3 operator*(const V3 &v)`
Or
Change `V3 operator*(V3 &v) to V3 operator*(const V3 v)`

1.5 + 1.5 + 1 + 1 marks

Roll number:_____

7. *[8 points]*
Provided below are 2 structs in C++ corresponding to a student and the courses he takes. The structure corresponding to a student contains the number of courses (*num_courses*) that the student took and all the courses(*courses*) that he took. The structure corresponding to a course contains the course code(*course_code*), the number of credits(*credits*) for the course and the grade(*grade*) that the student was assigned in that course. Now we will use these structs to define 2 functions:

- **float calculate_cpi(student A)**: This function computes the CPI of the student based on the credits weighted average of grades he/she secured in the courses he/she took
- **bool cleared_prereq(student A, int course_code, int pass_grade)**: Checks if the student managed to clear a particular course whose course_code and passing_grade are provided. Return true or false accordingly

Complete the structure and function definitions in the space below.

```
struct course {                                                    1 mark
  int course_code;
  int credits;
  int grade;
};

struct student {                                                   2 marks
  int num_courses;
  course courses[num_courses]; //Even course courses[] accepted
};

float calculate_cpi(student A) {                                   3 marks
  float sum=0;
  int total_credits=0;
  for(int i=0;i<A.num_courses;i++)
  {
    sum += A.courses[i].credits*A.courses[i].grade;
    total_credits += A.courses[i].credits;
  }
  return sum/float(total_credits);
}//Other correct solutions will also be accepted

bool cleared_prereq(student A, int course_code, int pass_grade) {
  for(int i=0;i<A.num_courses;i++)                                  2 marks
  {
    if(A.courses[i].course_code==course_code && A.courses[i].grade
>= pass_grade)
      return true;
  }
  return false;
}//Other correct solutions will also be accepted
```

8. *[4 points]*

```
#include<simplecpp>
struct myStruct {
  int a;
  myStruct() {cout << "D ";}
  myStruct(int x) {cout << "N ";}
  myStruct(myStruct &x) {cout << "C ";}
};
myStruct myFunc(myStruct p) {
  myStruct result = p;
  return result;
}
main_program {
  myStruct v1(35);
  myStruct v2;
  v2 = v1;
  myStruct v3 = v2;
  v1 = myFunc(v3);
  return 0;
}
```

Predict the output of the above program. Assume there is NO return value optimisation (RVO) performed by the compiler.

N D C C C C

1 + 1 + 0.5*4 marks

9. *[4 points]* *(0.5 marks for each blank are for writing **abs()**)*

You are given an array **arr** of size **size** that only contains all numbers from **1** to **(size-1)** in random order. Further, the array has exactly one element which occurs twice. Complete the code to find the repeating element of the array **arr**. ***HINT:*** *Iterate over the indices of the array and mark elements (by turning them negative) which are at the position corresponding to the value stored at the iterating index. The duplicate number corresponds to a negative element of the array being encountered.*

```
int findRepeating(int arr[], int size) {
  int missingElement = 0;
  for (int i = 0; i < size; i++) {
    if ( arr[abs(arr[i])] < 0 ) {        2.5 marks
        missingElement = abs(arr[i]);    0.5(abs) + 1(arr[i]) marks
        break;
    }
    arr[abs(arr[i])] = -arr[abs(arr[i])];
  }
  return missingElement;
}
```

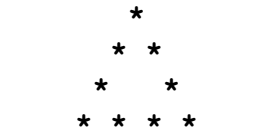
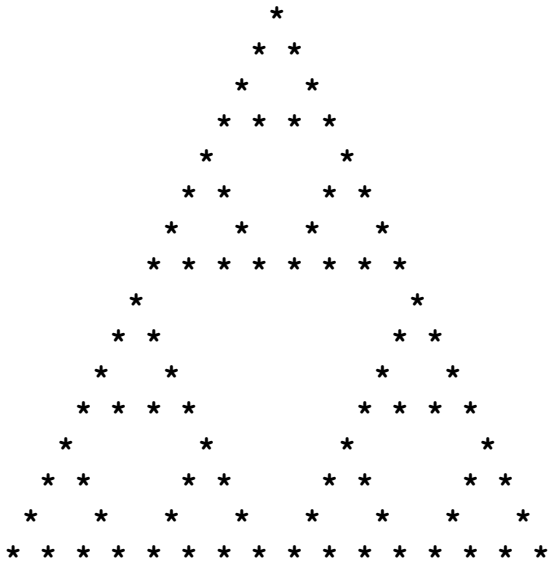
10. *[8 points]*
Complete the following program which given a big integer N, prints the binary representation of the integer N. N can be very large (around 10^{100}), therefore it is represented using a character/integer array. Assume you also have access to an additional function **`bool checkZero(int a[])`** which returns **`true`** if the number represented by **`a[]`** is 0 and **`false`** otherwise.

```
#include <simplecpp>
// Divide the present number in a[] by 2
// The division is done digit-wise accumulating carry all along
void divideBy2(int a[], int n) {
  int i, carry = 0, temp;
  for (i = 0; i < n; ++i) {
    // Alternate solutions to temp, carry possible - like
    // temp = (a[i]+10*carry)/2, carry = (a[i]+carry)%2
    // Store in temp: quotient when ith digit is divided by 2
    temp = (a[i]+carry)/2;                                  1.5 marks
    // Update the carry after division of ith digit
    carry = 10*((a[i]+carry)%2);                            1.5 marks
    a[i] = temp;
  }
}
main_program {
  char N[110];
  int a[110], ans[400];
  int i, n, bits = 0;
  cin >> N;
  for (i = 0; N[i] != '\0'; ++i) {
    a[i] = N[i]-'0';
  }
  n = i;
  // Loop as long as number (represented by a[]) is not 0
  // Appropriately call checkZero() here
  while (!checkZero(a)){ 0.5 for !, 1 for fn call     1.5 marks
    // Remainder on division of number by 2
    ans[bits] = a[n-1]%2;                                   1 mark
    // Divide the number by 2
    divideBy2(a, n);                                        1 mark
    ++bits;
  }
  // Bits are stored in reverse order in ans[]
  // Complete the for loop to print ans[] in reverse order
  for (i = bits-1; i >= 0; --i) {              0.5 + 0.5 + 0.5 marks
    cout << ans[i];
  }
}
```

Roll number:_____

11. *[6 points]*

A **Sierpinski Triangle** is a recursive pattern made of repeating equilateral triangles. Here we will explore a non-recursive method to create this pattern as is shown below for some examples (n = number of rows in the pattern) [Assume that the input n is a power of 2]

n = 2	n = 4	n = 8	n = 16
			

```
#include <simplecpp>
void Sierpinski(int n) {
  for (int y = n - 1 ; y >= 0 ; y--){                    1 + 0.5 + 0.5 marks
    for (int i = 0; i < y; i++)
      cout << " "; // Single space
    for (int x = 0; x + y < n; x++) {                    2 marks
      if(x & y)                                          2 marks
        cout << "  "; // 2 spaces
      else
        cout << "* "; // * followed by a space
    }
    cout << endl;
  }
}
main_program {
    int n;
    cin >> n; // Assume n is a power of 2
    Sierpinski(n);
}
```

12. *[6 points]*

Complete the below code which computes the product C of the matrix multiplication of 2 matrices A and B using recursion. [Static variables in a function: If a variable is declared as static in a function's scope it is only initialized once and retains its value in between recursive function calls] Your code should work for all values of M1, M2, N1, N2.

```
#include <simplecpp>
const int M1 = 5, M2 = 4, N1 = 4, N2 = 6;
/* Function which recursively computes C = AB */
void matmul(int a[M1][N1], int b[M2][N2], int c[M1][N2]){
//For all blanks of c[M1][N2] here and in main_program   1 mark
  static int i = 0, j = 0, k = 0;
  if (i < M1) {
    if (j < N2) {                                          0.5 marks
      if (k < N1) {                                        1 mark
        c[i][j] += a[i][k]*b[k][j];
        k = k+1 ;                                          1 mark
        matmul(a, b, c);
      }
      k = 0;
      j = j+1 ;                                            1 mark
      matmul(a, b, c);
    }
    j = 0 ;                                                1 mark
    i = i+1 ;                                              0.5 marks
    matmul(a, b, c);
  }
}
main_program {
  int A[M1][N1], B[M2][N2], C[M1][N2] = {0};
  for (int i = 0; i < M1; i++) {     /* Initialize matrix A */
    for (int j = 0; j < N1; j++)
      cin >> A[i][j];
  }
  for (int i = 0; i < M2; i++) {     /* Initialize matrix B */
    for (int j = 0; j < N2; j++)
      cin >> B[i][j];
  }
  matmul(A, B, C);
  for (int i = 0; i < M1; i++) {     /* Print matrix C */
    for (int j = 0; j < N2; j++)
      cout << C[i][j] << " ";
    cout << endl;
  }
}
```

13. *[7 points]*

Let **A** be an array of integers sorted in increasing order. A function for binary search can be used to quickly find if a specific target value is present in the array **A**. It works as follows: Binary search begins by comparing the middle element of the array with the target value. If the target value matches the middle element, its position in the array is returned. If the target value is less than or greater than the middle element, the search continues in the lower or upper half of the array, respectively, eliminating the other half from consideration[1].

You must complete the code snippet which implements a modified version of binary search for the following problem:

Given an array **A** of integers sorted in increasing order and two integers **lo** and **hi**, the function should return **true** if **A** contains any element **n** such that **lo <= n <= hi**. If no such element exists in the array, the function should return **false**.

```
#include <simplecpp>
// A is an array of integers sorted in increasing order
bool adaptedBinarySearch(int A[],int start,int end,int lo,int hi){
  int first = A[start], last = A[end-1], mid = A[(start+end)/2];
  if ((last < lo) || (first > hi)) return false;          1.5 marks
  if ((mid >= lo) && (mid <= hi)) return true;            1.5 marks
  if (mid < lo)                                                1 mark
    return adaptedBinarySearch(A,(start+end)/2,end,lo,hi);     1 mark
  if (mid > hi)                                                1 mark
    return adaptedBinarySearch(A,start,(start+end)/2,lo,hi);   1 mark
}
main_program {
  int n, lo, hi;
  cin >> n >> lo >> hi;
  int A[n];
  for (int i = 0; i < n; ++i)
    cin >> A[i];
  adaptedBinarySearch(A, 0, n, lo, hi);
}
```

[1] Source: https://en.wikipedia.org/wiki/Binary_search_algorithm

14. *[8 points]*

Complete the following code which given an integer array **arr**, prints out all possible permutations of the elements of **arr** where elements are allowed to repeat in a permutation.

Example 1:

arr = {1, 3}

All possible permutations are **{arr[0], arr[0]}, {arr[0], arr[1]}, {arr[1], arr[0]}**, and **{arr[1], arr[1]}** i.e. the output of the program must be

1 1
1 1
3 1
3 3

Example 2:

arr = {1, 3, 5}

There are 27 possible permutations: **{1, 1, 1}, {1, 1, 3}, {1, 1, 5}, {1, 3, 1}, {1, 3, 3}, {1, 3, 5}, {1, 5, 1}, {1, 5, 3}, {1, 5, 5}, {3, 1, 1}, {3, 1, 3}, {3, 1, 5}, {3, 3, 1}, {3, 3, 3}, {3, 3, 5}, {3, 5, 1}, {3, 5, 3}, {3, 5, 5}, {5, 1, 1}, {5, 1, 3}, {5, 1, 5}, {5, 3, 1}, {5, 3, 3}, {5, 3, 5}, {5, 5, 1}, {5, 5, 3}, {5, 5, 5}** which must be printed in 27 separate lines by the code below.

```
#include <simplecpp>
int n; // Size of array
void combinations(int arr[], int pos[], int h) {
  if (h == n) { // Handle base case                    1.5 marks
    for (int i = 0 ; i < n ; i++)
      cout << arr[pos[i]] << " ";                      2.5 marks
    cout << endl;
    return;
  }
  for (int i = 0 ; i < n; i++) {
    pos[h] = i;                                        2 marks
    combinations(arr, pos, h+1);                       2 marks
  }
}
main_program {
  cin >> n;
  int arr[n];
  for(int i = 0; i < n; ++i)
    cin >> arr[i];
  int pos[n];
  combinations(arr, pos, 0);
}
```

Roll number:_____

15. *[20 points]*
The aim of this question is to write code for computing inverse of a matrix. For this purpose, we first define a Struct **Matrix**. This struct stores every 2D matrix as a single array in the row major form (Elements of the 2D matrix of row i are followed by elements of row i+1 in the array) i.e;

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 2 & -2 & 0 \\ 1 & 2 & -1 \end{bmatrix}$$

Matrix A is stored as the array [3 1 -1 2 -2 0 1 2 -1] in the Struct Matrix

Since the size of the matrix is not known beforehand, we assume that the 2D matrix does not exceed dimensions of 20x20 [Hence the maximum array dimension of the Struct **Matrix** is 400]. The Struct **Matrix** has 3 members, an array which will stores the 2D matrix in row-major form and 2 integers which store the number of rows and columns of the matrix. Provided below is information on the Struct:

- Matrix(int r, int c) : Constructor which creates a 2D matrix of all zeros of size r x c
- Matrix(int** given_matrix, int r, int c) : Constructor which takes an existing 2D matrix `given_matrix` of size r x c
- Matrix& operator=(const Matrix& rhs) : Assignment Operator
- ~Matrix() : Destructor
- int& operator()(int r, int c): Returns the element matrix[r][c] of the matrix
- Matrix operator/(int scalar) : Member Function which divides every element of the matrix with the scalar `scalar`
- void printmat() : Member Function which prints the matrix as space-separated ints

Template code for Struct **Matrix** is provided. Fill the blanks to complete the code.

Now that you have filled and completed the Struct Matrix, we can move onto functions which compute the inverse of a square matrix. This in turn involves 4 functions which compute Inverse, Adjoint, Determinant and Co-Factor. From basic Matrix Algebra:
Given a matrix M, its inverse is given by $M^{-1} = \frac{1}{det(M)} adj(M)$ where `det(M)` is the determinant of M, and `adj(M)` is the adjoint of M. Look at the following example to recall how to compute the adjoint and determinant of a matrix. For above mentioned A matrix,

$$\text{Cofactor of } 3 = A_{11} = \begin{vmatrix} -2 & 0 \\ 2 & -1 \end{vmatrix} = 2$$

$$\text{Cofactor of } 1 = A_{12} = -\begin{vmatrix} 2 & 0 \\ 1 & -1 \end{vmatrix} = 2$$

$$\text{Cofactor of } -1 = A_{13} = \begin{vmatrix} 2 & -2 \\ 1 & 2 \end{vmatrix} = 6$$

$$\text{Cofactor of } 2 = A_{21} = -\begin{vmatrix} 1 & -1 \\ 2 & -1 \end{vmatrix} = -1$$

$$\text{Cofactor of } -2 = A_{22} = \begin{vmatrix} 3 & -1 \\ 1 & -1 \end{vmatrix} = -2$$

$$\text{Cofactor of } 0 = A_{23} = -\begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix} = -5$$

$$\text{Cofactor of } 1 = A_{31} = \begin{vmatrix} 1 & -1 \\ -2 & 0 \end{vmatrix} = -2$$

$$\text{Cofactor of } 2 = A_{32} = -\begin{vmatrix} 3 & -1 \\ 2 & 0 \end{vmatrix} = -2$$

$$\text{Cofactor of } -1 = A_{33} = \begin{vmatrix} 3 & 1 \\ 2 & -2 \end{vmatrix} = -8$$

$$\text{The cofactor matrix of } A \text{ is } [A_{ij}] = \begin{bmatrix} 2 & 2 & 6 \\ -1 & -2 & -5 \\ -2 & -2 & -8 \end{bmatrix}$$

Roll number:_____

$$adjA = (A_{ij})^T = \begin{bmatrix} 2 & -1 & -2 \\ 2 & -2 & -2 \\ 6 & -5 & -8 \end{bmatrix}$$

$$\det(A) \text{ or } |A| = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}.$$

The below provided code has recursive implementations for finding the Determinant and Adjoint of the matrix using calls to computing the Co-Factor for specific elements. You need to fill the blanks in the code for the functions Co-Factor, Determinant and Adjoint.

```
#include <simplecpp>
struct Matrix {
  int matrix[400];
  int rows;
  int columns;
  Matrix(int r, int c) {
    rows = r;
    columns = c;
    for(int i = 0; i < 400; i++)
      matrix[i] = 0;
  }
  Matrix(int given_matrix[20][20], int r, int c) {
    rows = r;
    columns = c;
    for(int i = 0; i < rows; i++) {
      for(int j = 0; j < columns; j++) {
        matrix[i*columns+j] = given_matrix[i][j];          2 marks
      }
    }
  }
  Matrix& operator=(const Matrix& rhs) {
    if (&rhs == this)
      return *this;                                       0.5 marks
    for (int i = 0; i < rhs.rows*rhs.columns; i++)
      matrix[i] = rhs.matrix[i];
    if (rhs.rows*rhs.columns < rows*columns) {
      for (int i = rhs.rows*rhs.columns; i < rows*columns; i++)
        matrix[i] = 0;                                    1 mark
    }
    rows = rhs.rows;
    columns = rhs.columns;
    return *this;                                         0.5 marks
  }
  ~Matrix(){}
  int& operator()(int r, int c) {
    if (r >= rows || r<0 || c >= columns || c<0)          1 mark
      cout << "Index out of bounds" << endl;
    return matrix[r * columns + c];                       1 mark
  }
```

```
  Matrix operator/(int scalar) {
    Matrix result(rows, columns);
    if (scalar != 0) {
      for (int i = 0; i < rows*columns; i++) {
        result.matrix[i] = matrix[i] / scalar ;            1 mark
      }
    }
    return result;
  }
  void printmat() {
    for(int i = 0; i < rows; i++) {
      for(int j = 0; j < columns; j++)
        cout << matrix[i*columns+j] << " ";                1 mark
      cout << endl;
    }
  }
};

/* Co-Factor of Matrix M of size n x n of the (p,q)th element */
Matrix getCofactor(Matrix M, int p, int q, int n) {
  int i = 0, j = 0;
  Matrix temp(n-1, n-1);
  for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
      if (row != p && col != q) {                          1 mark
        temp(i,j) = M(row, col);                           1 mark
        j++;
        if (j == n - 1) {
          j = 0;                                           1.5 marks
          i++;                                             1.5 marks
        }
      }
    }
  }
  return temp;
}

/* Determinant of Matrix M whose size is n x n */
int determinant(Matrix mat, int n) {
  int D = 0, sign = 1;
  if (n == 1)
    return mat(0,0); //Also acceptable mat.matrix[0]       1 mark
  Matrix temp(n, n);
  for (int f = 0; f < n; f++) {
    temp = getCofactor(mat, 0, f, n);                      1 mark
    D += sign * mat(0,f) * determinant(temp, n - 1);       1 mark
    sign = -sign;
  }
  return D;
}
```

Roll number:_____

```
/*Adjoint of Matrix M whose size is n x n*/
Matrix adjoint(Matrix A, int n) {
  Matrix adj(n,n);
  if (n == 1) {
    adj(0,0) = 1;  //Also acceptable adj.matrix[0] = 1;    1 mark
    return adj;
  }
  int sign = 1;
  Matrix temp(n, n);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      temp = getCofactor(A, i, j, n);                      1 mark
      sign = ((i+j)%2==0) ? 1: -1;                         1 mark
      adj(j,i) = (sign)*(determinant(temp, n-1));          1 mark
    }
  }
  return adj;
}

/*Inverse of Matrix M whose size is n x n*/
Matrix inverse(Matrix A, int n) {
  Matrix inverse(n, n);
  int det = determinant(A, n);
  Matrix adj = adjoint(A, n);
  inverse = adj/det;
  return inverse;
}

main_program {
  int m,n;
  cin >> m >> n;
  int array[20][20];
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      cin >> array[i][j];
  }
  Matrix A(array, m, n);
  if (m == n) {
    if (determinant(A, m) != 0)
    inverse(A, n).printmat();
  }
}
```

Roll number:_____

16. Challenge Question (No direct credit right now. To be used for AP grade) *[15 points]*
The following code rotates the matrix element of a 5 x 5 matrix such that the outermost elements are rotated anticlockwise while the elements one level inside are rotated clockwise and so on. Fill in the blanks so that the code works as expected. An example is as follows:

Input Matrix	Output Matrix
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34	14 19 24 29 34 13 26 21 16 33 12 27 22 17 32 11 28 23 18 31 10 15 20 25 30

```
#include <simplecpp>
void printMatrix(int a[5][5]) {
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
      cout << a[i][j] << " ";
    }
    cout << endl;
  }
  cout << endl;
}
void rotate(int mat[5][5], int x) {
  int anti_clockwise;
  if (x%2!=0) anti_clockwise = 3;                    2 marks
  else        anti_clockwise = 1;                    2 marks
  for (int i = 0; i < anti_clockwise; i++) {
    for (int y = x; y < 4-x; y++){                   1 mark
      //Following lines are worth 10 marks combined
      mat[x][y]     = mat[y][4-x];
      mat[y][4-x]   = mat[4-x][4-y];
      mat[4-x][4-y]= mat[4-y][x];
      mat[4-y][x]   = mat[x][y];
    }
  }
}
main_program {
  int a[5][5];
  for (int i = 0;i < 5; i++) {
    for (int j = 0; j < 5; j++) {
      a[i][j] = i*5 + j;
    }
  }
  printMatrix(a);
  rotate(a, 0);
  rotate(a, 1);
  printMatrix(a);
}
```

Roll number:_____

ROUGH WORK