HBM803E - Object Oriented Programming Techniques

MID-TERM (05/12/2020, Saturday)

Name / Surname and Student ID:

Q-1) Specifiers (10 pts)

In the file specifiers.cpp, you see 7 fields names such as 5th being __field5__, and 12 lines commented as //line-1, //line-2 and so on.

Using macro #define, we use those fields to determine access specifiers or inheritance specifiers. Those fields can be: “private”, “protected” or “public”.

a) Complete the following table by writing whether those fields are an access specifier or an inheritance specifier:

Fields	1	2	3	4	5	6	7
Answer							

Commented lines will work only for a certain combinations of those fields. For example, line-1 and line-7 will work for:
Example:
Should work: line1, line-7
Should not work: -

Fields:	1	2	3	4	5	6	7
solution-1	public	any	any	public	any	any	any

Note: You should comment out some lines in the class “Derived” for your code to compile if the lines 10, 11 and 12 will not work (comment out lines 31,32 and 33 respectively), in addition to the lines that doesn’t work in the main function.

Complete the following tables such that your answers will satisfy given conditions(you can answer as: private, protected, public, private/protected, …, any or none if the task is impossible):

b)
Should work: line-1, line-7
Should not work: line-8, line-11

Fields:	1	2	3	4	5	6	7

c)
Should work: line-1, line-7, line-11
Should not work: line-8

Fields:	1	2	3	4	5	6	7

d)
Should work: line-4, line-10
Should not work: line-5

Fields: 1 2 3 4 5 6 7

e)
Should work: line-3
Should not work: line-9

Fields: 1 2 3 4 5 6 7

f)
Should work: line-1, line-12
Should not work: line-7, line-11, line-3

Fields: 1 2 3 4 5 6 7

42 Fields, 10 points: Given that number of fields you answered correctly is N, you will get 0.25*(N-2) points

Q-2) (40 pts)
Fill question_2.cpp step by step by answering the following questions. All the code you add should be placed at the position of comment with that question's name (e.g. //a-1//), you may lose points for any other changes.

a-1) write default constructor for BaseFunc: equate N_params to 0, pointer params to a value which means it points to nothing, using initializer list.
a-2) write parameterized constructor for BaseFunc: equate N_params to input parameter a, allocate memory for N_params double variable and make params point to the starting address of the allocated memory
a-3) write destructor for BaseFunc: deallocate all allocated memories, use an if statement to avoid compiler error when the object is instantiated by default constructor

b) write method get_params: it should take one input parameter which is a pointer to double. Assuming there is N_params double variable pointed by this input parameter, equate the array params to this input array.

c) write a method for overloading operator "( )". Make sure that this method will not be used if a pointer to BaseFunc which actually points to an object of a class derived from BaseFunc calls operator "( )".
Also, make sure that the class BaseFunc cannot be instantiated, as well as its child classes that don't override this operator.

d) select a suitable inheritance specifier to dictate how LinearFunc class inherits BaseFunc class.

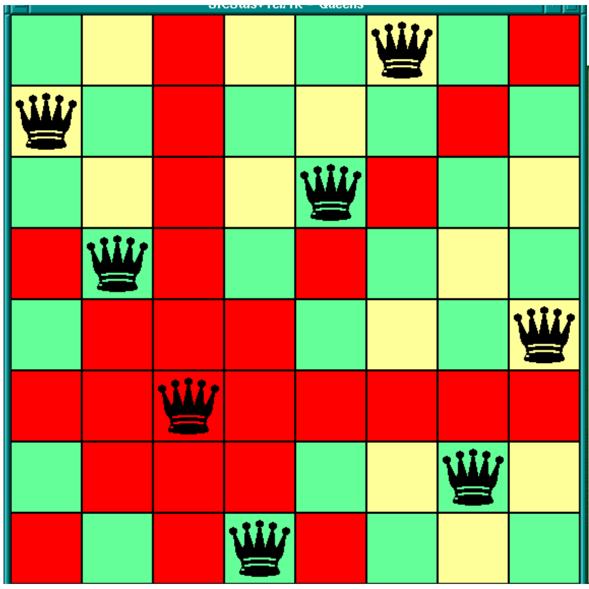
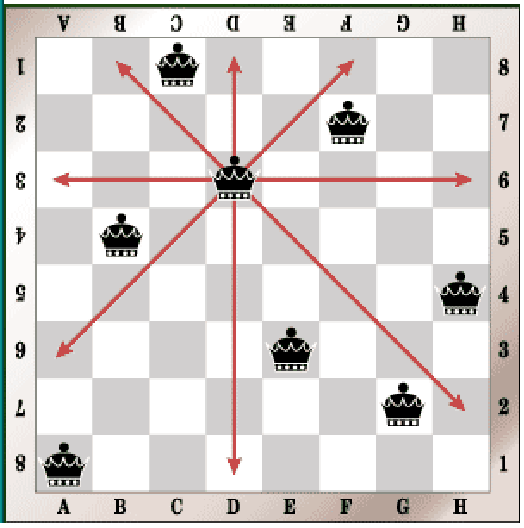
The class LinearFunc will be used to calculate $y = p_0 + p_1 x_0 + p_2 x_1 + ... + p_{N_x - 1} x_{N_x - 2} + p_{N_x} x_{N_x - 1}$with N_x = N_x

e) Write parameterized constructor for LinearFunc: it should call a suitable base class constructor with an adequate input using initializer list, and equate N_x to the input parameter n_x in the constructor body (between {})

f) Override the operator ( ) so that it takes a pointer to double as input parameter, calculate result of the function using this input parameter (assuming that it points to an array of adequate size) and the array of parameters "params".

g) call the function "dummy" from the main function, by passing LinearFunc object f into it. You don't have to create a pointer for this purpose.

Q-3) N-Queens problem (50 pts)

N Queens problem is a famous problem among computer programmers.
In this problem, N queens (tr: vezir) are placed in a N*N chess board (standard chess board is 8*8, so the most common form is 8 Queens)
For a solution to be feasible, no queen should be able to attack any other queen. In other words, two or more queens shouldn't be placed in the same row, column or diagonal.

In this question, we will use an integer array of size N to represent a solution. There will be exactly one queen in every column in those solutions, regardless if the solution is feasible or not. The elements of the solution array represents the row of a queen, and index of that element is the column of that queen. (First picture will be (1,3,5,7,2,0,6,4) and second will be (7,3,0,2,5,1,6,4) if we start enumerating columns from left and rows from the top)

Our aim is to find all the feasible solutions to a N queens problem. We will encapsulate our functions and data in a class.

A brute force approach to this problem would be trying all the possible solutions and keeping feasible ones. But this approach does not scale well: each queen can be at any position in that column, so there is N possible place for N different queens, resulting in N to the power of N different possible solutions. This will take too much computational time for larger N. The computational cost of an algorithm is often dependent on one or more parameters, and we express this kink of computational complexity with big-O notation. O(N) means that complexity of that algorithm will increase linearly with N, O(logN) means it will increase proportional to the logarithm of N, and so on. Brute force approach to this problem would have a computational comlexity of $O(N^N)$, which is huge.

We can save some of this time by not calculating solutions that are obviously infeasible. For example, there is no point to calculate N^{N-2} solutions of the form (0,0,…..) because we already know that first two queens are in the same row so no solution of that form will be feasible. When we place queens only to the free rows, we will need to try N! solutions (for 8 queen, first queen can be placed in any row, second one can be placed in any of the other 7 rows, next one in 6 rows etc, resulting in 8! combinations).
To implement such an algorithm, we need to categorize partial solutions as potentially feasible or infeasible.

In this algorithm, we will place queens only to available rows. A row is available for that column if that row isn’t used before, or isn’t on the same diagonal with any previous queens. We will start with first column, and put first queen on the first row in the first move. Then, we will check if the partial solution so far is feasible (it is always feasible for the first column) and proceed to the next column. We place second queen to the first row of second column, and get partial solution (0,0,…) which is infeasible. When a solution is infeasible, we try next row in that column. Partial solution (0,1,…) is also infeasible, but (0,2,….) is feasible. When we find an available row, we proceed to the next column and start trying by (0,2,0,…) if we are not on the last column. Our solutions become full solutions in the last column, so we save feasible solutions in the last column instead of proceeding to the next column. When we try each row in a column, we should go back to the previous column, and continue to try next rows at this column. You can read the description or pseudocode of “is_feasible” and “solve” methods for further details.

Hint: Number of solutions to N Queen problem are given in the following table for N=0..12

N:	0	1	2	3	4	5	6	7	8	9	10	11	12
#solution	1	1	0	0	2	10	4	40	92	352	724	2680	14200

<u>Write a class named “N_Queen”.</u>

This class should have data attributes: integer “N” with no default value, integer “sol_counter” with an initial value 0, a pointer to integer “sol” and a pointer to pointer to integer “solutions” with intial values NULL. All should be inaccessible from main function.

<u>A parameterized constructor (2 pts)</u>: This should take an input argument integer, equate N to this input argument, allocate memory of N integers to “sol” and equate those integers to -1. It should not allocate memory for “solutions”

<u>A destructor (2 pts)</u>: This should deallocate all allocated memory (on pointers sol and solutions). It cannot deallocate memory if no memory is allocated, so be sure that some memory is allocated before deallocation.

<u>A method “is_feasible” (10 pts)</u> that takes an integer “col” as input argument: The purpose of this method is to check if any conflict is present between the queen at the column “col” and queens at columns 0...col-1.
These is 3 conditions that create a conflict: 2 queen are in the same row, or 2 queens are in the same diagonal.
You should check if the row of the queen at the column “col” is the same with any previous queen.
For diagonals, you should check if the sum of column and row number are the same, and if the difference of column and row are the same.
If one of those are the same, then the current partial solution sol[0...col] is not feasible. If so, this function should return “false”. If it is feasible, it should return “true”.

<u>A method “allocate_solutions” (1 pts)</u> that takes no argument: It should allocate memory for two dimensional array. Each row in this array will hold a feasible solution, every row should have N integers. The number of rows should be equal to the number of solutions for N.

<u>A method “add_solution” (1 pts)</u> that that writes the intent of the array “sol” to the sol_counter-th row of “solutions” and increase sol_counter by 1.
A method “print_solutions” to print all found solutions (there should be sol_counter real solutions at the array “solutions” at any given time).

A method “solve” that takes no argument: (30 pts)

This method should return the number of times the method “is_feasible” is used. You should create a variable in this method “solve”, and increment in in this method (not in “is_feasible”), and return in at the end. This task is not handled in the pseudocode, you should add it.

Implement the following pseudocode in this method:

```
allocate memory for “solutions”
current column is 0th column
while current column is equal or greater than 0:
    increase the current solution by 1 for the current column
    if all rows in the current column are checked:
        make the current solution -1 for the current column
        return to the previous column (decrease current column)
    else if the current solution is feasible until the current column:
        if the current column is the last column:
            add the current solution to “solutions”
        else:
            proceed to the next column (increase current column)
```

In the main function:
create a N_Queen object of size N, call the solve() method of that object and print the return value (which should be number of calls to “is_feasible” method made in the “solve” method) and print all the solutions found.

Your program should yield the following results

N	#calls to “is_feasible”	#feasible solutions
5	220	10
8	15720	92
10	348150	724

Lastly, answer the following questions by choosing one of the options (4 pts):
This algorithm can be seen as a (Breadth-First Search / Depth-First Search).
This algorithm uses (backtracing / gradient of the objective function / heuristics).
This algorithm is (stochastic / deterministic).
Computational complexity of this algorithm is upper bounded by (O(N!) / O(N^N) / O(NlogN)). (select minimum complexity that bounds)

Rules

- The slides and notes used in the practice sections can be used in the exam
- Internet sources can be used, but copy-paste codes and code sharing is strictly prohibited.
- **Copy-Paste from internet** sources can be detected and **is not allowed**.
- **Code sharing is not allowed**.
- Please upload your answers packed as a single file (int .tar, .tar.gz, .zip or .rar format)
- You are expected to answer question-1 with a text file q1.txt, in which your answers to each subsection are written on the same line seperated by commas. You are expected to updoad codes q2.cpp and q3.cpp and include your written answers to multiple choice question of q-3 included at the end as comments.