Barem Teorie

T1) Am definita clasa Irat de numere irationale (cu parte reala si parte imaginara). Instantiez obiectele x, y, z de acest fel. Ce trebuie facut ca bucata de cod urmatoare sa compileze:
y=2*x;
x=x+1;
z=y/2;
descrierea conceptului/conceptelor, sintaxa, proprietati, restrictii.

5 variante dar toate asemanatoare
0.1 p pentru mentionat redefinirea operatorilor
0.2 p pentru redefinirea corecta a operatorului * (se poate face doar ca functie independenta)
0.1 p pentru redefinirea celorlalti operatori (+, /)
0.1 p pentru alte proprietati ale redefinirii operatorilor
(trebuie neaparat sintaxa corecta macar pentru unul din operatorii *, +, /) altfel max 0.3
sintaxa nu e data max 0.3, sintaxa gresita (Irat operator/(Irat & x)) max 0.2

T2) Dati cat mai multe variante de a se modifica starea (variabilele de instanta) unui obiect constant care apeleaza o functie constanta.
0.2 p pentru fiecare din urmatoarele (const_cast, cast away constness, mutable, volatile) cu un maxim de 0.5
prostii -0.1p pentru fiecare prostie
daca se precizeaza ca din functii const NU se poate modifica starea obiectului max 0.1p

T3) Cum functioneaza operatorul de atribuire implicit (dat de compilator), al unei clase compuse (cu date dintr-o clasa de baza),si cum trebuie scris de programator, operatorul de atribuire pentru aceeasi clasa compusa? Sintaxa.

0.1 op= implicit -copiere bit cu bit
0.1 op=implicit -apel op= pt date membre (sau baza!! -pt cei care au inteles derivare)
0.1. suprascriere ca metoda
0.1 suprascriere -cand exista date alocate dinamic
0.1 op = suprascris nu apeleaza implicit op= pt datele membre -> trebuie apelat explicit
0.1 sintaxa op = antet(C& operator=(C&)){return *this }
/alep op= membre(data=param.data)
/apel op=baza(baza::operator=(param)) (pt cei care au inteles gresit derivare)
ultimele doua sunt sau exclusiv

T4) Descrieti specializarea explicita pentru sabloane (template-uri) de clase.
Sintaxa, proprietati, observatii.
0.1p specifice pentru anumite tipuri de date
0.1p sintaxa: template<>class Nume<Tip_particular>
0.1p exemplu
0.1p nu se pastreaza nimic din clasa template pe care o specializeaza, deci trebuie rescrise metodele
0.1p e prioritara fata de template pt instantiere cu acelasi tip de data 0.1p
0.1p fara prostii

T5) Descrieti cum se poate re-arunca o exceptie: sintaxa, proprietati, restrictii, utilizare.
0.1p **throw;**
0.2p ca e try in try si primul throw x iar in primul catch se face throw; si se prinde in al doilea catch
0.1p ca se face cand se vrea re-procesarea unei exceptii din alt punct de vedere
0.1p altele: ca se poate face din functii sau nu, etc.
0.1p fara prostii

T6) Sa se scrie cod pentru urmatoarea situatie:

sa se construiasca o clasa CLS care sa aibe ca date de instanta cel putin doi intregi unul constant c si unul normal i.

se citeste n de la tastatura (va fi intotdeuna mai mare decat 10)

sa se instantieze n obiecte din aceasta clasa CLS cu restrictiile urmatoare:

1. primele 3 obiecte vor avea initializate c si i cu c=2 si i=3

2. obiectele de la 5 la 6 vor avea c si i la valoarea 1

3 obiectele de la 7 la n vor avea c=numarul obiectului respectiv, i va fi n-c, adica al optulea obiect va avea c=8, al noualea obiect va avea c=9, etc.

Sa se descrie notiunea de POO care va ajuta sa rezolvati restrictiile 1,2,3

5 variante similar cu unele mici diferente

- incalcarea conceptelor de POO (e.g. atribuirea de valori catre variabile const dupa initializare sau incalcarea encapusularii informatiiei - acesarea in main a unei proprietati private) - 0p
- Mentionarea si utilizarea corecta in cod a listei de initializare a constructorului pentru initializarea variabile const 0.2p:
  - 0.1p mentionare
  - 0.1p utilizare
- Implementarea instantierii celor n obiecte 0.3p:
  - 0.2p instantierea obiectelor conform restrictiilor (se acorda punctaj partial)
  - 0.1p salvarea obiectelor intr-o structura de tip colectie:
    - daca se folosesc pointeri cu new/delete (i.e. alocarea dinamica): 0.05p pentru alocarea propriu-zisa si 0.05 pentru dezalocare
    - daca se foloseste vector => 0.1p