Lista temelor, cu toate cerintele pentru **a doua lucrare practica**.

- Cerinte comune tuturor temelor / in principal temelor 1-17:

1. toate clase vor conține obligatoriu constructori de inițializare, parametrizati și de copiere, destructor, iar operatorii >>,<<,= să fie supraincarcati;
2. clasele derivate trebuie sa contina constructori prin care sa se evidentieze transmiterea parametrilor către constructorul din clasa de baza;
3. ilustrarea conceptelor de moștenire și functii virtuale (pure – unde se considera mai natural);
4. tratarea exceptiilor;
5. citirea informațiilor complete a n obiecte, memorarea și afisarea acestora.

- Cerinte specifice fiecarei teme:

Tema 1. liste de numere intregi, reprezentate ca tablouri uni- si bidimensionale:

Se dau urmatoarele clase:
- Vector (int dim, int *a)
- Matrice (Vector *v) – clasa abstracta
- Matrice_oarecare (int lin):Matrice
- Matrice_patratica (int dim): Matrice /*– nr de linii (dim) trebuie sa coincida cu dimensiunea oricarui vector component

Clasa derivata trebuie sa contina o metoda care sa verifice daca o matrice triunghiulara este diagonala. Pentru matricile patratice, functia de afisare sa conțină și determinantul acestora.

Tema 2. vectori de numere complexe

Se dau urmatoarele clase:
- Complex (float re,im)
- Vector (int dim, Complex *v).
- Stiva : Vector - cu operatiile de adaugare și stergere modificate corespunzator.
- Coada: Vector - cu operatiile de adaugare și stergere modificate corespunzator.

Clasele derivate trebuie sa contina o metoda prin care sa verifice dacă stiva / coada este pur "imaginara".

Tema 3. Cozi de siruri de caractere (implementate dinamic):

Retinem fiecare sir de caractere cuvant cu cuvant

Se dau urmatoarele clase:
- Nod (char* info, nod*next);
- Coada (nod * prim, nod * ultim, int dim_max); cele 2 operatii clasice de la coada
- DEQUE:Coada - cu operatiile de adaugare și stergere modificate corespunzator.
- DEQUE_MARCAJ:Coada - cu operatiile de adaugare și stergere modificate corespunzator. Nu o sa mai fie NULL o sa fie un nod marcaj pentru stergere si adaugare

putem adauga sau sterge din ambele capete, functii cu supraincarcare

Operatiile ce trebuie implementate pentru cozi sunt urmãtoarele:

Nodul marcaj se afla la inceput si contine orice informatie si daca vreau sa parcurg lista invers voi merge pana la nodul marcaj

citire litera cu litera, dc citim cuvant apelam inserare inserare pt fiecare litera

insert <cuvânt> <prioritate> adaugã cuvântul în coadã cu prioritatea respectivã
top întoarce primul cuvânt din coadã - cel cu prioritate maximã
Nu avem prioritate
pop eliminã primul element din coadã
empty goleste coada.

Aceste comenzi se vor regãsi una per linie.

Tema 4. Coada cu prioritati cu elemente de tip sir de caractere (implementata dinamic)

Se dau urmatoarele clase:
- Nod (char* info, nod*next)
- Nod_dublu:Nod {nod * ante;};
- Nod_prioritate:Nod_dublu {int prio;}

Să se implementeze clasa Coada_prioritati care sa aibă elemente de tip Nod_prioritate.
Operatiile ce trebuie implementate pentru coadã sunt urmãtoarele:

insert <cuvânt> <prioritate> adaugã cuvântul în coadã cu prioritatea respectivã
top întoarce primul cuvânt din coadã - cel cu prioritate maximã
pop eliminã primul element din coadã
empty goleste coada.

Aceste comenzi se vor regãsi una per linie.

Tema 5. multimi finite de numere întregi reprezentate ca tablouri unidimensionale

Se dau urmatoarele clase:
- Clasa Pereche(int,int)
- Multime_pereche (int n, Pereche * p)
- Stiva_pereche : Multime_pereche – cu operatiile de adaugare și stergere modificate corespunzător;
- Coada_pereche : Multime_pereche – cu operatiile de adaugare și stergere modificate corespunzător;

Să se implementeze o functie prin care se simuleaza operatiile pe o stiva folosind 2 cozi.

Tema 6. polinoame reprezentate ca tablouri unidimensionale (prin gradul polinomului si vectorul coeficientilor.

Se dau urmatoarele clase:
- Clasa Monom(int grad, float coef)
- Clasa Polinom(int nr_monoame, Monom *m)
- Polinom_ireductibil : Polinom.

Dacă vi se pare mai natural, puteți sa aveți clasa Polinom ca Și clasa de baza pentru Polinom_Ireductibil și pentru Polinom_reductibil.
- Clasa Polinom(int nr_monoame, Monom *m) – clasa abstracta
- Polinom_ireductibil : Polinom.
- Polinom_reductibil:Polinom.

Afisarea unui polinom reductibil să fie făcută și ca produs de 2 polinoame.

Clasa derivata trebuie sa contina o metoda care sa aplice criteriul lui Eisenstein de verificare a ireductibilitatii polinoamelor. (Webografie ajutatoare: **http://www.profesoronline.ro/teorie-3140-1.html)**

Tema 7. matrice de numere complexe reprezentate ca tablouri bidimensionale

Se dau urmatoarele clase:
- Clasa Complex(float,float)
- Matrice(Complex **v) – clasa abstracta
- Matrice_oarecare (int lin, int col) : Matrice
- Matrice_patratica (int dim): Matrice

Clasele derivate trebuie sa contina metoda care sa verifice daca o matrice triunghiulara este diagonala. Pentru matricile patratice, functia de afisare sa conțină și determinantul acestora.

Tema 8. matrice de numere complexe reprezentate ca structuri inlantuite (ca matrici rare si nu neaparat patratice)
- Aceleași cerinte ca la Tema 7.

Tema 9. arbori binari de cautare si arbori rosu-negru in reprezentare inlantuita:

Se dau urmatoarele clase:
- Nod (int info, nod*st, nod *dr)
- Nod_rosu_negru:Nod {char culoare[]};
- Arbore(int nr_noduri) – clasa abstracta
- ABC (Nod *rad):Arbore
- Arbore_bicolor (Nod_rosu_negru *rad) : Arbore

Clasa derivata trebuie sa contina o functie virtuala prin care inaltimea arborelui bicolor sa fie identificata prin Adancimea neagra a radacinii. Webografie: https://writer.zoho.com/public/miha_cio/Arbori-bicolori1/noband

Tema 10. arbori binari de cautare AVL in reprezentare inlantuita:

Se dau urmatoarele clase:
- Nod (int info, nod*st, nod *dr)
- Nod_AVL:Nod {int echilibru};
- Arbore(int nr_noduri) – clasa abstracta
- ABC (Nod *rad):Arbore
- Arbore_AVL(Nod_AVL *rad) : Arbore

Clasa derivata trebuie sa contina o functie prin care să se afiseze factorii de echilibru pentru fiecare nod.

Tema 11. arbori oarecare, in reprezentare inlantuita, prin legaturile fiu (catre fiul cel mai din stanga) si frate (catre urmatorul fiu al tatalui, in ordinea fiilor tatalui de la stanga la dreapta).

Se dau urmatoarele clase:

- Nod {int info, int nr_copii, Nod * leg[10]};
- Arbore(int nr_noduri) – clasa abstracta
- Arbore_oarecare(Nod *rad) : Arbore
- Arbore binar: Arbore oarecare

Afisarea elementelor unui arbore binar oarecare sa fie data de parcurgerea în adancime si parcurgerea in latime, cu mentiunea listei fiilor pentru fiecare nod, iar afisarea unui Arbore_binar sa conțină cele 3 parcurgeri simultan.

Tema 12. grafuri

Se dau urmatoarele clase:
- Matrice (int **a) – matrice de adiacenta
- Vector (int *v, int dim)
- Lista (Vector *l) – lista de adiacenta
- Graf (int nr_noduri) – clasa abstracta
- Graf_Neorientat(Lista L) : Graf
- Graf_Orientat (Matrice A) :Graf

Clasele derivate trebuie sa contina constructor parametrizat prin care sa se evidentieze transmiterea parametrilor catre constructorul din clasa de baza, destructor si o metoda care sa afiseze vectorul de tati, dacă pentru un Graf Orientat se verifica conexitatea lui și se da un nod de plecare pe post de rădăcina.

Tema 13. grafuri orientate,

Se dau urmatoarele clase:
- Matrice (int **a) – matrice de adiacenta
- Graf (int nr_noduri) – clasa abstracta
- Graf_complet (int nr_muchii) : Graf
- Graf_antisimetric(Matrice M) : Graf
- Graf_turneu: Graf_complet, Graf_antisimetric

Clasele derivate trebuie sa contina o metoda care sa afiseze arcele unui Graf_antisiemtric, sau a unui Graf Turneu.
Obs: graf turneu – orientat complet si antisimetric. Un graf orientat se numește antisimetric dacă pentru oricare două vârfuri din graf x și y dacă există arcul (x,y), atunci nu există arcul (y,x). Un graf orientat sau neorientat se numește complet dacă oricare două vârfuri din graf sunt adiacente.

Tema 14: Din clasa Nr_Natural_Mare să se deriveze clasa

Se dau urmatoarele clase:
- Vector (int *a)
- Nr_Natural_Mare(int_nrcif, Vector V)
- Numar_Intreg_mare(char semn):Nr_Natural_Mare

Să se implementeze operatiile uzuale

Tema 15.

Se dau urmatoarele clase:
- Vector (int *a)
- Nr_Natural_Mare(int_nrcif, Vector V)
- Numar_Fibonacci_mare:Nr_Natural_Mare

Sa se implementeze ierarhia de clase mai sus mentionata, iar clasa derivata trebuie sa o functie virtuala care sa afiseze numarul Fibonacci, precum si descompunerea lui in suma de numere Fibonacci.

Obs: Autorul Temelor 16 – 17 este Lect. Dr. Mihai Gabroveanu (http://inf.ucv.ro/~mihaiug/courses/poo/labs/Probleme%20C++.pdf)

Tema 16.
Sa se implementeze un vector(tablou) in care se pot pastra obiecte de tipuri diferite definite de utilizator. Pentru aceasta se defineşte clasa Object ca o clasă de bază pentru toate tipurile derivate:

```
class Object{
public:
Object() {};
virtual ~Object() {};
virtual void display() = 0;
};
```

Clasa ObArray, defineşte un vector de pointeri de tip Object* . Nu este necesar să fie limitată dimensiunea vectorului deoarece se asigură creşterea dimensiunii acestuia atunci când este necesar.

```
class ObArray : public Object {
Object **p; // vector de pointeri
int size; // numar de elemente la un moment dat
int grows; // increment de creştere a dimensiunii
int dimens; // dimensiune vector
public:ObArray(int size=0,int grows, int dimens);//Constructor
~ObArray();//Destructor
void RemoveAll();//Elimina toate obiectele din vector
int getSize(); // Intoarce numarul de elemente din vector.
int add(Object* x); //Adauga un element la vector.
int insertAt(int i, Object *x);//Insereaza un element pe // o pozitie data
int removeAt(int i); //Elimina elementul de pe pozitia i
Object* getAt(int i); //Intoarce elementul de la pozitia i
void display();// Afisare elementelor din tablou.
};
```

Sa se utilizeze aceste clase pentru a memora un tablou de Puncte si un tablou de numere Complexe.

Clasele derivate trebuie sa contina constructori parametrizati prin care sa se evidentieze transmiterea parametrilor catre constructorul din clasa de baza si destructori

Tema 17.

Se dau urmatoarele clase:

- Clasa Persoana(id, nume)
- Clasa Abonat:Persoana(nr_telefon)
- Clasa Abonat_Skype: Abonat (id_skype)
- Clasa Abonat_Skype_Romania (adresa_mail) : Abonat_Skype
- Clasa Abonat_Skype_Extern(tara) : Abonat_Skype

Sa se construiasca clasa Agenda ce contina o lista de abonati si sa se supraincarce operatorul [ ](indexare)care returneaza abonatul cu numele precizat .

Obs: Autorul Temelor 16 – 17 este drd Mircea Digulescu

Tema 18.

Magazin: Exista un anumit stoc din mai multe tipuri de bunuri (unele masurate la bucata altele la greutate, altele la volum). Fiecare produs are un pret pe unitatea de masura si un cost. Clientul soseste cu o lista de cumparaturi cu bunuri. Magazionerul ii comunica clientului pretul total iar acesta decide daca cumpara sau nu. Daca cumpara se realizeaza, se produce schimbul bunuri contra bani. La finalul zilei se realizeaza inchiderea zilei, afisand totaluri de bunuri vandute si banii obtinuti. Scrieti un program care sa permita aceste operatiuni.

Precizari:

- Tipurile de Produsele pot fi cel putin: Varza (la buc), Faina (la kg) – cal. I, II sau III, Bere la doza (buc) – cu Brand si tip (blonda/bruna), **Vin** varsat (rosu sec si alb sec) – la volum, **Vin** de soi - la sticla (buc) – cu soi (Cabernet Savignon (rosu sec), Merlot (rosu dulce), Savignon Blanc (alb dulce), Chardonnay (alb dulce)), an culegere si tara de origine (Franta, Argentina, Chile, Australia); Cartofi (la Kg) – rosii sau albi si Jucarii (la buc) – unicat (cu nume text).
- Magazinul detine Articole de stoc.
- Clientul poate specifica vag produsele, omitand anumite trasaturi (eg. Vin, Vin rosu; Faina) sau detaliat (vin rosu sec din 1996 din Franta). Magazionerul propune cele mai profitabile variante avand in vedere precizarile clientului si disponibilul de stoc.
    - In lista de cumparaturi a clientului se pot repeta tipuri de produs.

Tema 19.

Restaurant: La un restaurant exista mai multe mese. La fiecare masa se pot aseza 1 sau mai multi clienti (maxim in functie de capacitatea mesei). Clientii sosesc in grupuri si se aseaza la o masa suficient de mare. Apoi comanda din meniul care contine mai multe feluri de mancare si bauturi fiecare avand un pret. Fiecare client comanda pentru el. Dupa ce mananca parasesc masa (eventual pe rand). Cand un client paraseste masa el doreste nota de plata pentru ce a consumat el. La final se elibereaza nota de plata pentru clientii ramasi. Scrieti un program care sa permita aceste operatiuni.
Precizari:

- Feluri de mancare pot fi cel putin: Ciorba – de burta, de legume (ambele cu sau fara smantana si eventual ardei) si de ciuperci; Fel principal – de pui (Frigarui sau Tocana) sau de vita (Chateaubriand – precizat cat de bine facut sa fie, stroganoff sau file – la 100g), Garnitura (doar la fel principal) – cartofi (prajiti sau piure) sau orez sarbesc si Desert – Inghetata la cupa (intre 3 si 5 cupe) sau tort krantz.
- Suplimentar exista bauturi: Vin varsat (rosu sau alb) – la 100ml, Sticle de vin (Bordeux din 1996 sau Castel Bolovanu 2004), Sticle de bere (cu/fara alcool) sau apa plata (la sticla).
- Interactiunea cu chelnerul se desfasoara sub forma unui dialog in care aceasta afla progresiv dorintele clientului (e.g. De baut va aduc ceva? Da. Vin, bere sau apa? Vin. La pahar sau la sticla? La pahar. Cati ml? 200ml. Multumesc; doriti ciorba? Da. De care? De legume. Cu sau fara smantana? Cu. Cu sau fara Ardei? Fara. Fel principal doriti? Nu. Desert doriti? Da. Inghetata sau tort kranz? Inghetata. Cate cupe? 4).
- Chelnerul intreaba pe rand fiecare persoana de la masa ce doreste.
- Chelnerul poate fi chemat de un client la o masa si poate sa i se ceara Nota individuala sau nota colectiva (a tuturor clientilor ramasi la masa).

Tema 20.

Atelier auto: Exista mai multe categorii de vehicule (motocicleta, bicicleta, automobil) care pot sosi la un garaj pentru a descoperi ce probleme tehnice au si a obtine un deviz de reparatii. Fiecare vehicul poate avea probleme din diferite categorii: frane, directie, noxe, motor, caroserie. Nu toate vehiculele pot avea toate tipurile de probleme. Fiecare tip de problema are o solutie tehnica ce presupune un anumit consum de materiale (de diferite tipuri) si un anumit efort uman (in ore-om). Fiecare material are un cost la fel ca si ora-om de efort uman. Pentru fiecare vehicul ce soseste in garaj trebuie calculat devizul estimativ.

Precizari:

- Tipurile de probleme vor include cel putin: Frane – placute uzate (fata/spate la un auto/moto), discuri uzate (oricare din 4 la un auto, oricare din 2 la moto), lant tocit (bicicleta), lant lipsa (bicicleta), defectiune capitala; Motor - Nivel ulei scazut (la moto si auto), carburator murdar (la auto), motor topit (la moto si auto); Directie – ghidon stramb (bicicleta, moto), roata stramba (toate), defectiune

capitala (toate); Noxe – Vehiculul arde ulei (moto si auto), Vehiculul este anterior anului 2000 (moto si auto); Caroserie (auto) – Caroserie corodata de rugina, Caroserie stramba (aripi stanga/dreapta, fata/spate, bara, capota) - foarte stramba sau asa-si-asa (pe fiecare componenta in parte).

- Solutiile presupun un consum de materiale. Materialele sunt toate la bucata. Placutele de frana de pe spate difera de cele de pe fata. Placutele de frana sunt aferente fiecarei roti. Nivel ulei scazut, motor topit si vehiculul arde ulei toate presupun schimb de ulei.
- Fiecare problema are o solutie (mai putin defectiunea capitala). Pot exista tipuri de materiale comune mai multor solutii. Toate solutii consuma suruburi (minim 5).
- Vehiculul suporta operatiuni de traumatizare (e.g. Uzeaza discul de frana de pe spate dreapta sau Incepe sa arzi ulei).
- Vehiculul odata conectat la tester (in atelier) se va putea autodiagnostica verificand fiecare componenta a sa si returnand lista de probleme.

1. Agentia de turism: Exista turisti. Ei au anumite caracteristici (barbati/femei, grupa de varsta si activitatile preferate). Exista destinatii turistice (Amsterdam, Thassos, etc.). Ele permit o serie de activitati (plaja, mers pe munte, vizitat muzee, etc.) dintre care unele pot fi potrivite doar pentru anumite sexe sau grupe de varsta. Unele activitati necesita un autovehicul personal. O activitate dureaza un numar de zile. O activitate poate fi si de grup (pot lua mai multi parte la ea). Pentru a ajunge la o destinatie se foloseste fie transport cu avionul fie transport individual cu autovehiculul. Pentru un grup de turisti (cu caracterstici si preferinte variate) sa se aleaga o destinatie potrivita si sa se realizeze un plan de activitati (o lista) pe zile (maxim 14) astfel incat intreg grupul sa fie satisfacut.
   Precizari:
   - O activitate se desfasoara intr-un anumit loc la destinatie. Daca distanta de la cazare pana la acel loc este mai mare de 2 Km, este necesar un autovehicul.
   - Unele destinatii permit inchirierea de autovehicule, altele nu.
   - Orice activitate dureaza minim jumatate de zi.
   - O destinatie poate avea mai multe plaje. O plaje poate fi acoperita cu nisip sau pietricele mici (mai ales in Grecia).
   - Unele plaje sunt de nudisti. Unele plaje de nudisti nu accepta minori.
   - Mersul pe munte se poate efectua in grup. La fel si plaja.
   - Mersul pe munte necesita un munte. Muntele prezinta diferite trasee de dificultate usoara/medie/grea. Trasele de dificultate grea sunt interzise copiilor sub 16 ani sau adultilor peste 90. Unele trasee pe munte necesita mai multe zile. Dintre acestea unele presupun opriri la o manastire. Unele manastiri nu accepta vizitatori femei.
   - Exista mai multe tipuri de muzee (de arta, de istorie, stiintifice). In anumite destinatii este interzisa vizitarea muzeelor stiintifice de catre femei.
   - Acelasi tip de activitati poate fi comun mai multor destinatii.

- Un turist care exprima preferinte pentru un anumit tip de activitate poate omite anumite detalii, fiind indiferent la ele sau le poate preciza (e.g. plaja; plaja cu nisip; mers pe munte minim 2 zile traseu mediu).

Tema 21.

Firma de Training: O firma de Traning ofera mai multe Programe de Training (dezvoltand o anumita Competenta) pentru cursantii sai. Exista o multime fixa maximala de cursanti. Programele sunt alcatuite dintr-o serie de cursuri si eventuale alte programe de Training (incluse). Pentru a termina un program de training, cursantul trebuie sa promoveze toate cursurile ce fac parte din acel program (direct sau indirect). Un curs (e.g. „Programare OOP in C++") este urmat in cadrul unui Program de un anumit cursant o singura data, chiar daca face parte din mai multe sub-Programe. Daca un cursant a picat un Program, el il poate reface daca a luat minim 3 la toate prezentarile de pana atunci. Cursurile sunt de mai multe feluri (Programare elementara (OOP/non-OOP, ce limbaj), Limba straina (ce limba), Matematica (algebra/geometrie/analiza), Resurse Umane, Financiar, Legislatie auto, Sofat auto practic (tipul vehicului), Comunicarea NLP). Tutorul va completa pentru fiecare cursant al fiecarei prezentari a unui Program o Fisa de Evaluare. Ea va contine o portiune generala (ce cursant, ce program, ce prezentare) apoi notele la fiecare Lucrare Practica a fiecarei prezentari de curs continuta. Cursurile au un numar diferit de lucrari practice. Unele cursuri (Programare, Matematica, Comunicare NLP) au evaluare scrisa/orala de final. Pentru fiecare curs rezulta o nota finala. Pentru unele cursuri la calculul mediei se elimina cea mai slaba nota, pentru altele atat cea mai slaba cat si cea mai buna (alegeti voi la care!). Unele cursuri pot acorda ponderi diferite diferitelor LP (ramase in calcul). Nota la cursul de NLP este nota de la examenul oral daca media celorlalte LP este peste 5 si 1 altfel. Nota unui program se calculeaza ca media notelor programelor continute *direct*. Scopul este ca pe baza fiselor individuale sa se obtina o situatie statistica generala: per cursant (ce competente are si la ce nivel) si per competenta (cat cursanti au o anumita competenta peste un anumit nivel).

Precizari:

- Finalizarea cu promovare a unui Program de Training genereaza o Competenta.
- In functie de nota integului Program, Competenta poate fi la nivel mic (5-6), mediu (7-8) si mare (9-10).
- Exista programe de Finantist: Matematica (algebra) si Financiar; Manager: Resurse Umane, Financiar, Limba(engleza); Programator: Matematica (toate trei), Programare, Limba(engleza); Manager Echipa Programatori: Manager, Programare si Comunicare NLP; Sofer: Legislatie auto, Sofat auto practic; Manager echipa de tiristi: Limba(turca), Manager si Legislatie auto.
- Functionalitatea programului este de urmatorul fel: Sosesc un grup de cursanti care doresc sa urmeze un anumit Program. De la tasatura se citesc corespunzator datele din fisa de evaluare, apoi se afiseaza nota finala si nivelul de competenta obinut pentru fiecare. Fiecare cursant opteaza daca doreste sa refaca cursurile pe care le-a promovat anterior sau nu (caz in care se preia nota).

- La cererea utilizatorului se afiseaza statistica cu numarul de cursanti (fosti) avand o anumita competenta la un anumit nivel.

Tema 22.

Curierat rapid: O firma de curierat are mai multe vehicule in teritoriu (scutere, masini, dube). Fiecare are o capacitate (o masa maxima admisa si un volum total posibil). Unele dube sunt frigorifice si pot transporta bunuri reci. Fiecare vehicul se afla intr-un anumit punct (x,y). Firma (dispeceratul) primeste comenzi de livrare a unor colete (mai multe articole cu masa / volum diferite, unele dintre ele putand fi reci) de la un client (aflat la alte coordonate x,y) catre o destinatie (din nou x,y). Intreg coletul este preluat de un singur livrator. Soferii se deplaseaza cu viteze diferite (scutere – 20 Km/s, masini – 10 Km/s, dube – 5 Km/s) catre destinatie, in linie dreapta (scutere) sau manhattan (masini, dube). Odata livrat un colet, un vehiculul ramane la destinatie daca nu mai are alte colete de livrat. Coletele trebuie sa ajunga „la timp". Pentru diferite colete la timp poate insemna: pana la o anumita secunda din zi, intr-un interval de timp maxim (de la preluarea comenzii/preluarea coletului), cat mai repede (fara limita de timp). Firma poate aplica 3 strategii de alocare a coletelor (e.g. vehiculul cel mai apropiat de client, vehiculul care – conform strategiei sale curente – poate livra cel mai repede coletul, vehiculul cel mai putin incarcat). Vehiculele se aleg dintre cele capabile sa transporte coletul (au spatiu) si ar reusi conform strategiei curente sa il livreze la timp. Daca nu exista nici un astfel de vehicul, clientul este refuzat. De asemenea, soferul vehiculului poate opta pentru 3 strategii de livrare/ridicare (coletul cel mai urgent primul, coletul cel mai apropiat de pozitia curenta, first-come-first-served). Sa se simuleze operatiunile firmei de curierat intr-o zi.

Precizari:

- Un vehicul nu isi schimba destinatia pe parcursul unui drum (doar in momentele in care ridica/livreaza un colet).
- Strategia soferului poate trata unitar (la fel) ambele tipuri de operatiuni (ridicare si livrare) sau poate sa faca distinctie intre ele (e.g. mai intai livrari apoi ridicari).
- Strategiile de alocare pot fi schimbate de manager pe parcursul unei zile. La fel si strategiile fiecarui sofer (tot de manager) – insa fara ca un colet sa ajunga sa nu mai poata fi livrat la timp.
- Performanta algoritmului de alocare nu este cruciala pentru aceasta cerinta la acest curs.

Tema 23.

Harti: O firma de software a primit cerinta de a scrie o aplicatie care determina drumul minim intre doua orase alese dintr-o regiune, folosind sistemul de strazi existent. Diferite regiuni au diferete sisteme de strazi, unele putand forma un graf linie, altele un arbore, altele un dag, altele un graf general si altele un graf complet. Fiecare oras are coordonatele (x,y) si drumul dintre doua orase (daca exista strada intre ele) este distanta plana (euclidiana) intre ele. Firma de software ruleaza algoritmul pe propriile servere, tinand

datele pentru o serie de regiuni, pentru care primeste cereri de la clienti (drum minim de la A la B regiunea R; introduce drum intre A si B in regiunea R; creaza o regiune). Algoritmul trebuie sa ruleze cat de cat rapid avand in vedere tipul de graf asociat.

Precizari:

- Drumul minim returnat consta doar in distanta totala de parcurs.
- Observatii: Intr-un graf complet drumul minim de la A la B este fix strada AB. Intr-un arbore exista un singur drum intre doua noduri (care se poate calcula tinand cont de stramosul comun). Intr-un graf linie costul drumului minim se poate calcula in O(1) usor.
- Strazile sunt bidirectionale, mai putin in dag.

Tema 24.

Bibilioteca: O biblioteca ofera suplimentar clientilor sai un serviciu de stocare si regasire a informatiilor. Astfel, fiecare client poate solicita ca bibilioteca sa memoreze informatii pe care el le-a furnizat si pe care le-a numit intr-un anumit fel. Informatiile de memorat pot fi de tip text (siruri de caractere), numeric (numere naturale), matematic (numere complexe) si de tip adresa (tara, judet, oras, strada, numar). De asemenea, informatiile pot avea un nume (e.g. “Adresa Mariei”, “Numarul meu preferat”). Odata introduse in bibilioteca, informatiile capata un ID. Biblioteca ofera urmatoarele servicii: Adaugare informatii, Stergere informatii dupa nume/ID, Regasire informatii (dupa Nume sau ID) si Cautare Informatii (dupa valoare – e.g. 27).

Precizari:

- Daca se incearca adaugarea de informatii cu un nume deja existent, se vor produce urmatoarele: Pentru tipul text se va alipi (apenda) noua valoare la informatia stocata; Pentru tipul numeric si matematic se vor insuma valorile; Pentru tipul adresa se va returna eroare.
- Performanta cautarii nu este cruciala pentru aceasta cerinta.

Tema 25.

Teatru modern: Un teatru modern ofera mai multe spectacole pentru public. Spectacolele pot fi de diferite tipuri (e.g. opera, teatru, circ). Unele tipuri sunt adecvate doar pentru adulti. Fiecare spectacol are un gen (comedie, drama, etc.). Spectacolele de circ sunt alcatuite din mai multe numere (prestatii). Unele numere pot face uz de animale vii. Spectatorii au anumite trasaturi (nume, varsta) si anumite Pofte (e.g. “opera comedie”) si anumite exigente (e.g. nu doresc animale vii). Un spectator poate avea mai multe pofte la un moment dat, el fiind multumit cu satisfacerea oricareia dintre ele. Spectacolele au mai multe prezentari pe parcursul unei luni (momente cand ruleaza). Fiecare prezentare dispune de un numar fix de locuri. Lucratorul de la ghiseu al teatrului modern are sarcina de a sugerea fiecarui client individual care soseste lista de prezentari cu spectacole care l-ar multumi pe client. Cerinta presupune redactarea unui program in acest scop.

Precizari:

- Toate spectacolele au o durata si un nume.

- Un act de opera are un solist principal. Solistul principal are un nume si poate fi Tenor, Bariton sau Bass. Unele opere pot avea mai multe acte (2-7), cu durate diferite, specificate pentru fiecare act in parte. Intre acte se ia o pauza fixa specifica operei.
- Spectacolele de circ sunt toate comedii. Intre numerele unui spectacol de circ nu se ia pauza. Unele numere de circ pot fi adecvate doar pentru adulti caz in care intreg spectacolul este astfel adecvat. Numerele care presupun animale vii precizeaza ce animale sunt folosite (lei, ursi, pisici, pinguini).
- Teatrul are un singur act. Unele piese de teatru sunt interactive (presupun participarea publicului). In acestea, pe langa durata fixa exista si o durata variabila dependenta de numarul de spectatori (un minut pentru fiecare spectator adult si doua pentru fiecare copil sub 16 ani).
- O pofta a clientilor cuprinde una sau mai multe din urmatoarele cerinte: (i) spectacolul sa fie de un anumit gen (comedie, drama, actiune); (ii) sa fie de un anumit tip (teatru, opera, circ); (iii) sa dureze minim un anumit timp (cu/fara pauze); (iv) sa fie circ care sa includa un anumit gen de animale vii (e.g. pinguini); (v) sa fie spectacol de teatru interactiv; (vi) daca este opera sa inglobeze un anumit tip de voce (e.g. Tenor).
- Exigentele clientilor sunt unele dintre urmatoarele: (i) sa nu fie o anumita combinatie de (tip, gen) – e.g. opera dramatica; (ii) sa dureze (cu tot cu eventualele pauze) maxim un anumit timp; (iii) sa nu presupuna animale vii periculoase; (iv) sa fie maxim un numar (e.g. 20) de spectatori. Pot exista mai multe exigente de acelasi tip. Suplimentar, spectacolul trebuie sa fie potrivit pentru categoria de varsta a clientului.
- Pe langa operatia la ghiseu, programul va permite si introducerea spectacolelor si prezentarilor.

Tema 26.

Criptografie: In Criptografie una din activitatile cel mai adesea efectuate presupune diverse operatii aritmetice (+,-,*,/) in diferite Corpuri (nu Grupuri). Scopul programului aferent acestei cerinte este sa permiteti operatii cat mai generice peste Corpuri. Astfel, un Corp va putea fi definit de grupul aditiv si grupul multiplicativ. Un grup va putea fi definit de un tip elementar (R, Z, **Q**, Z/nZ, numere complexe) sau de explicitarea regulilor de compunere (pentru grupuri finite). Dandu-se o specificatie pentru un Corp de catre utilizator sa se: Determine daca este valida, Implementeze operatiile aritmetice in acel Corp. Suplimentar, dandu-se doua Corpuri, sa se determine daca sunt izomorfe.

Precizari:

- Grupul aditiv si cel multiplicativ au aceeasi multime.
- Pentru doua elemente dintr-un corp trebuie implementata si comparatia pentru egalitate (sunt sau nu egale).
- Programul trebuie sa proceseze comenzi (de la tastatura) odata ce a citit corpurile de intrare.

Tema 27.

Hotel: Un hotel are mai multe tipuri de incaperi (camere, apartamente, sali de conferinta, restaurant, etc.). Restaurantul are o capacitate maxima fixa, mai mica decat capacitatea totala de cazare. Unele sali de conferinta sunt echipate cu proiector. Unele sali sunt dotate cu mese individuale pentru participanti (4-8 persoane) iar altele sunt organizate tip scena. La Hotel sosesc diferite cereri pentru organizarea de evenimente in grup. Fiecare grup are un numar de participanti (ce trebuiesc cazati in camere si apartamente) si o perioada de sedere (1-10 zile). Unii dintre participanti vor dori sa ia micul dejun la restaurant, altii in camera (aleg la inceput). Unele grupuri vor dori sa rezerve si o sala de un anumit tip (scena/individuala) cu capacitate suficienta pe parcursul sederii lor. Cererile sosesc pe rand. Sa se determine pentru fiecare cerere cea mai apropiata perioada de inceputul de an (ziua 1) cand ea poate fi onorata si sa se tipareasca detaliile (ce camere au fost rezervate, ce sala, etc.). De asemenea pot sosi cereri de anulare a unei rezervari.

Precizari:

- Camerele obisnuite au capacitate maxima 2.
- Apartamentele au capacitatea 4.
- Nu se cazeaza participanti din grupuri diferite in aceeasi camera.
- Rezervarea include pentru fiecare participant (nume) camera in care este cazat (numarul). Participantii nu au preferinte la cuplajul in camere.
- In cazul anularii unei rezervari anuleaza rezervarea tuturor elementelor (camere, sala, etc.).
- Ar fi dragutz daca rezervarile ar fi anulate rapid – bonus!

Tema 28.

Spalatorie: Exista mai multe tipuri de haine (Pantaloni, Rochii, Camasi, Paltoane, Gegci, Costume) cu greutati diferite, printre altele. La o Spalatorie, aceste tipuri de haine sunt Supuse unui proces tehnologic de curatatorie realizat cu niste echipamente (Masini). El presupune o serie de etape: Spalare, Stoarcere, Uscare si Calcare. Hainele usoare (pantaloni, rochii, camasi) se pot spala intr-o masina obisnuita. Hainele grele (Paltoane, Gegci) necesita alt echipament special la Spalare. Paltoanele si Gegicile nu se Storc si nu se Calca. Unele haine necesita Spalare peste o anumita temperatura, altele dimpotriva (sub o anumita temperatura). Hainele au culori (deschise/inchise). Ele nu se amesteca la Spalare. Pentru a urma intregul proces, hainele trebuie trecute prin mai multe masini. Este recomandat ca Masinile sa lucreze cand sunt umplute peste jumatate din la capacitatea maxima. Scopul programului este implementarea fluxului tehnologic: (i) preluarea de haine de la clienti; (ii) plasarea lor in masini; (iii) pornirea masinilor cand sunt suficient de pline; (iv) preluarea hainelor de la o masina si mutarea catre urmatoarea (Spalat, Stors, Uscat, Calcat). Hainele sunt preluate in ordinea sosirii clientilor.

Precizari:

- Spalatoria poate prelua mai multe articole de la un client la un moment dat (e.g. 2 pantaloni, 3 gegci, etc.).

- Capacitatea masinilor de Spalat si Stors este determinata de greutate, a celor de Uscat de numarul de articole iar Calcatul se realizeaza manual (avand capacitate 1 articol).
- Costumele se spala separat de alte categorii de haine in masinile obisnuite dar se pot spala impreuna cu Gegci sau Paltoane.
- La Spalare se utilizeaza o cantitate de detergent determinata astfel: Pentru camasi si rochii – o valoare fixa unica tuturor articolelor; pentru Pantaloni – dublul cantitatii fixe dar doar daca sunt de culoare inchisa; pentru restul se folosesc 100g detergent / Kg de articol.
- Fiecare masina are o durata de functionare a programului (de spalat, stors, uscat) fixata.
- Durata calcatului este dependenta de greutatea obietului si de tipul sau. Pentru camasi este 120 secunde / Kg, pentru pantaloni 90 secunde / Kg, iar pentru costume suma dintre durata pentru sacou (150 s/Kg) si pentru pantaloni (idem mai sus).
- Clientii au rabdare oricat de multa.
- O haina „tine minte” prin ce a trecut, astfel incat, la finalul procesului tehnologic sa poata afisa operatiunile (detaliat) la care a fost supusa.
- Hainele se distribuie in prima Masina potrivita disponibila (fara un algoritm special).
- Cand toate hainele aferente comenzii unui client au fost procesate, este afisat pentru acea comanda totalul de detergent folosit si timpul total petrecut in Masini in timp ce lucrau (adica nu cat erau plasate acolo fara ca masina sa functioneze).

Obs: Autorul Temelor 29 –30 este Lect. Dr. Daniel Dragulici

Tema 29.

Joc tanc in mod text: doua tancuri conduse fiecare de cate un jucator se deplaseaza printr-un labirint si trag unul in altul. Proiectul contine:
* Fisierele "engine" (".h" si ".cpp") care descriu engine-ul 2d al jocului; rolul engine-lui este doar de a gestiona pozitia unitatilor pe tabla si interactiunea (schimbul de informatii) dintre ele; comportamentul specific al fiecarei unitati si modul sau specific de reactie vor fi definite ulterior, in cadrul unor clase derivate (metodele "action" si behaviour" - a se veda mai jos). Fisierele vor contine doua clase:
- clasa "engine", care desemneaza engine-ul propriuzis; ea are:
clasa "unit" (a se vedea mai jos) ca clasa friend;
membrii data privati:
"unit* unittab[MAXNC][MAXNL]" = tabla de joc;
"unit* unitvect[MAXNU]" = lista unitatilor de pe tabla;
"int nc,nl,nu" = nr. liniilor, coloanelor, unitatilor existente;
(eventual alti membrii data privati)
metode publice:

"void init(int,int)" = init. "nc", "nl" (eventual si alti membrii data) cu val. date ca param.; init. "unittab" cu comp. NULL; se apeleaza o singura data, la inceput;

"int getnl()", "int getnc()", "int getnu()" = ret."nl", "nc", resp."nu";

"int validxy(int,int)" = test. daca parametrii reprezinta o poz. Valida pe tabla "unittab" (deci daca sunt intre 0 si "nc-1",resp."nl-1");

"int validpoz(int)" = testeaza daca parametrul reprezinta o poz. Valida in lista "unitvect" (deci daca este intre 0 si "nu-1");

"unit* getunitxy(int,int)" = ret. adr. unitatii de pe o poz. data in "unittab" (NULL daca poz. e libera); pozitia se presupune valida;

"unit* getunitpoz(int)" = ret. adr. unitatii de pe o poz. data in "unitvect"; pozitia se presupune valida;

"void next()" = efect. un pas (trece de la o configuratie a tablei la urmatoarea); e cea mai importanta metoda; algoritmul sau este:

- tuturor unitatilor din "unitvect" le apeleaza metoda "action";
- tuturor unitatilor din "unitvect" le apeleaza metoda "react";
- fiec. unitate moarta ("!alive") din "unitvect" este eliminata din "unitvect", poz. ocupata de ea pe tabla "unittab" devine NULL, apoi este dezalocata cu "delete" (iar "nu" scade corespunzator);

clasa mai poate avea si alte met.publice, care sa furnizeze inf.despre ea;

- clasa "unit", desemnand nucleul comun al unitatilor gestionate de engine; nu se va instantia (in obiecte) direct ci doar prin descendenti; ea are:

clasa "engine" ca clasa friend;

membru data public:

"static engine motor" = engine-ul in care se afla unitatea; fiind static, va fi automat acelasi pentru toate unitatile; deci,in program vom declara/aloca doar unitati, nu si un engine (vom lucra cu obiectul "motor" al clasei "unit" sau al descendentilor sai);

membrii data privati:

"int alive" = daca mai este viu (0/1); constructorul il inti. cu 1;

"int x,y,poz" = poz. unitatii in engine-ul sau "motor", (in "unittab", resp. "unitvect"); la mutarea unitatii pe alta poz. (in "unittab" sau "unitvect") se vor actualiza si acesti "x", "y", "poz" (sa concorde ce stie "motor" si ce stie fiecare unitate despre aceste pozitii);

"int message[MAXNM]" = lista cu mesaje (intregi) provenite de la alte unitati; prin asemenea mesaje unitatile isi comunica actiunile efectuate una asupra alteia; "message" poate fi si alocat dinamic;

"int nm" = nr. mesajelor primite si neprelucrate aflate in "message";

metoda privata:

"void react()" = reactia la actiunile celorlalte unitati asupra sa: pt. fiecare mesaj din "message", daca e inca viu ("alive"), apeleaza "behaviour"; in final pune "nm=0" (mesajele au fost prelucrate);

metode protected:

"unit(int,int)" = constructorul (fiind protected si nu public, nu vom putea instantia direct clasa "unit" ci doar descendenti ai sai - in constructorul lor se va apela acest "unit"); parametrii sunt pozitia (pres.valida si lib.)in"motor",pe"unittab",unde va fi pusa; actiuni:

- daca "unitvect" e plin sau poz. pe "unittab" e ocupata, se iese;
- inser."this" la sfar. lui "unitvect" si in poz. data pe "unittab";

- "nm" devine 0, "alive" devine 1;
- diverse alte actiuni;

"virtual ~unit()" = destructorul; daca "message" este alocat dinamic, il dezaloca (altfel poate avea corp vid); faptul ca este virtual va permite dezalocarea uniforma a descendentilor;

"void die()" = "alive" devine 0 (dar nu apeleaza destructorul, obiectul va fi elim. cu "delete" mai tarziu, de metoda "next" a lui "motor");

"void move(int dx,int dy)" = se pres. ca "x+dx", "y+dy" e o poz. valida si libera in "motor" pe"unittab"; efect: muta unitatea in poz."x+dx", "y+dy" (actualiz."unittab"-ul lui "motor" si "x", "y" ale unitatii);

"void sendmes(int pm,unit& pu)"= trimite mesajul "pm" unitatii "pu"; actiune: daca lista"message" a lui "pu" e plina nu face nimic, altfel adauga "pm" la sfarsitul ei;

"virtual void action()=0" = met. virt. pura exprimand ce face unitatea la un pas al lui "motor"; de ex., poate contine apeluri "sendmes" sau "die"; este cea mai importanta metoda, ea implem. AI-ul unitatii;

"virtual void behaviour(int)=0" = met. virt. pura exprimand comportam. unitatii fata de un anumit mesaj primit; de ex. poate apela "die";

metode publice:

"int getx()", "int gety()" = ret. "x", resp. "y";

"virtual void* gettip()=0" = metoda virtuala abstracta ce ret. adresa unei zone de mem. ce contine inf. referitoare la unitate (de ex. ce trebuie desenat pe ecran atunci cand o afisam).

Clasele "engine" si "unit" pot avea si alti membrii/metode, dar ei tb.sa fie de utilit.generala, nu legate de specificul jocului "tanc". Studiati posib. de a scrie un engine ce suporta mai multe unitati pe o aceeasi pozitie !

* Fisierele "joctanc" (".h" si ".cpp") care descriu aplicarea engine-ului de mai sus la jocul "tanc". Ele contin trei descendenti ai clasei"unit" (toate vor mosteni insa acelasi engine static"motor").Pe langa ele, fisierul".cpp" poate contine si alte definitii cu vizibilitate limitata la el (ele nu vor aparea si in fisierul ".h") - de ex. un tip enumerare cu mesajele pe care si le pot trimite unitatile cu "sendmes", printre ele "KILL". Clasele sunt:

- clasa "zid"; o unitate "zid" nu se deplaseaza si nu poate fi distrusa; din obiecte "zid" plasate pe tabla comuna "motor" se construieste labirintul; clasa mosteneste public pe "unit" si are in plus:

membru data privat:

"static unsigned char nume" = contine caracterul 177; adresa acestei zone va fi returnata de "gettip" iar continutul ei va fi folosit la afisare;

metode protected mostenite ca virtuale abstracte si redefinite:

"void action()", "void behaviour(int)" = nu fac nimic (corp vid);

metode publice:

"zid(int,int)" = constructorul; param. sunt poz. (pres. valida si libera) unde se va plasa zidul (folosind constructorul "unit");

metoda publica mostenita ca virtuala abstracta si redefinita:

"void* gettip()" = returneaza "&nume"

- clasa "bomba"; o unitate "bomba" este un proiectil tras de un tanc; el se misca constant pe o acceasi dir.(oriz. sau vert.), data la crearea sa,

pana loveste alta unitate sau iese de pe tabla, iar atunci moare ("die");
daca a intalnit alta unitate, inainte de a muri ii trimite cu "sendmes"
mesajul "KILL"; clasa mosteneste public pe "unit" si are in plus:
membri data privati:
"static unsigned char nume" = contine caracterul '.'; adresa acestei zone
va fi returnata de "gettip" iar continutul ei va fi folosit la afisare;
"int dirx, diry" = directia de deplasare; pot fi -1, 0 sau 1;
metode protected mostenite ca virtuale abstracte si redefinite:
"void action()" = algoritmul sau este:
- calc. coord. "xn=getx()+dirx, yn=gety()+diry" unde ar trebui sa se deplaseze ("getx" si "gety" sunt mostenite de la "unit");
- daca poz."xn,yn"invalida, atunci"die"(a iesit de pe tabla),altfel: daca poz. e libera (NULL), se muta acolo ("move"), altfel trimite unitatii de acolo mesajul "KILL" si apoi "die";

"void behaviour(int)" = actiune: daca parametrul e "KILL" atunci "die";
metode publice:
"bomba(int,int,int,int)" = constructorul; param. sunt poz. (pres. valida si libera) unde se va plasa bomba (folosind constructorul "unit") si directia de deplasare (cu care se vor initializa "dirx", "diry");
metoda publica mostenita ca virtuala abstracta si redefinita:
"void* gettip()" = returneaza "&nume".

- clasa "tanc"; o unitate "tanc" se poate misca (oriz. sau vert.) sau poate trage, in fct. de o comanda (char) primita de la intrare (tastatura);
clasa mosteneste public pe "unit" si are in plus:
membru data public:
"static char cc" = comanda curenta data spre exec.; fiind static, este acelasi pt. toate tancurile; ideea: de la tastatura se cit."tanc::cc", apoi toate tancurile vor trata (la "action") aceasta comanda dar doar acela care o va recunoaste va face ceva; la inceput se init. cu 0;
membri data privati:
"unsigned char nume" = contine numele tancului(setat de constructor);adr. lui va fi ret. de "gettip" iar continutul lui va fi fol. la afisare;
"char stg,dr,sus,jos,foc" = comenzile proprii pentru deplasare si foc; vor fi setate de constructor;
"char pc" = ultima comanda de deplasare executata de tanc (va fi folosita in cazul unei comenzi foc, deoarece bombei i se va da aceeasi dir. in care tancul s-a deplasat ultima oara);
metode protected mostenite ca virtuale abstracte si redefinite:
"void action()" = algoritmul sau este:
- testeaza "cc" (care e comun tuturor tancurilor); doar daca este egala cu "stg", "dr", "sus", "jos" sau "foc" propriu face ceva;
- daca este egala cu "stg", "dr", "sus" sau "jos" iar poz. in care ar trebuie sa se mute este valida si libera,efectueaza deplasarea respectiva ("move") si seteaza "pc=cc";
- daca este egala cu "foc": determina in functie de "pc" dir. in care va trage, apoi determina coord."bx,by" unde trebuie creata o bomba ("new bomb(.....)"); daca pozitia "bx,by" e valida, atunci:

- daca e si libera, creaza o bomba in acea pozitie si cu dir.
de deplasare stabilita mai inainte in functie de"pc";
- daca nu e lib.,trimite "KILL" unitatii de acolo (cu"sendmes");
"void behaviour(int)" = actiune: daca parametrul e "KILL" atunci "die";
metode publice:
"tanc(char,char,char,char,char,char,int,int)" = constructorul;parametrii:
numele, comenzile de depl.si foc, poz.pe tabla (pres. valida si lib.);
creaza tancul in poz.resp.(cu constructorul "unit") si init. membrii;
metoda publica mostenita ca virtuala abstracta si redefinita:
"void* gettip()" = returneaza "&nume".

* Fisierele "input" (".h" si ".cpp") care descriu interfata prin care se cit. date de la intrare. Contin def.clasei"input". Ea va avea o singura instant. numita "intrare", in prog. princ., care va fi interf. de intrare - de la ea se va citi de fiecare data comanda "tanc::cc". In plus, fisierul ".cpp" poate contine si alte definitii cu vizibilitate limitata la el (ele nu vor aparea si in fis. ".h"). Clasa are (cel putin) urmatoarele metode publice:
"void init()" = init.interfetei;poate contine orice (chiar si corpul vid);
se va apela o singura data, la inceput;
"void close()" = inchiderea interfetei; poate contine orice (chiar si corpul vid); se va apela o singura data, la sfarsit;
"char getc()" = testeaza daca s-a testat un caracter; daca nu, ret. 0;
daca da, ret. caracterul respectiv (un fel de "kbhit"+"getch"); trebuie scrisa a.i. daca se apasa o tasta speciala sa se ret. doar caracterul nenul de la sfarsitul secventei emise - de ex. stim ca tastele sgeata stg, dr, sus, jos emit caracterul nul, urmat de resp.'K', 'M', 'H', 'P';
Clasa "input" tb. scrisa indep. de celelalte fisiere din proiect, a.i. daca vrem sa modif.jocul pt.ca datele sa nu mai fie citite de la tastatura ci de la mouse sau un port de comunic. sa fie necesara doar rescr.clasei "input".

* Fisierele "outputt" (".h" si ".cpp") care descriu interf. prin care se afis. desfasurarea jocului pe ecran. Contin def. clasei "output". Ea va avea o singura instantiere numita "iesire", in prog. princ., care se va ocupa cu afisarea pe ecran a desfasurarii jocului. In plus, fis.".cpp" poate contine si alte def. cu vizib. limitata la el (ele nu vor aparea si in fis. ".h")
Clasa are (cel putin) urmatoarele:
membru data privat:
"engine& motor" = (referinta la) engine-ul jocului afisat; e initializat de constructor si e singura cale comunicare cu jocul(orice informare cu priv.la starea jocului se face prin metodele publice ale lui "motor");
metode publice:
"output(engine& pm)" = constructorul; asigneaza "motor" cu "pm"(si atat);
"void init()" = initializeaza modul de afisare text ("textmode(c4350)", _setcursortype(_NOCURSOR)", "clrscr()", etc., vezi "conio.h"), desen. chenarul (caractere 219) si labirintul - "zid"-urile nu dispar si nu se misca niciodata,deci pot fi desen.acum;se poate afla poz.lor parcurgand "unitvect"-ul lui "motor" si testand cu "gettip" tipul unitatilor;
aceasta metoda se va apela o singura data, la inceputul programului;
"void close()" = restaureaza modul de afisare text ("textmode(C80)",

"_setcursortype(_NORMALCURSOR)", "clrscr()", etc); aceasta metoda se va apela o singura data, la sfarsitul programului;
"void afisare()" = reactualizeaza informatia de pe ecran conform starii actuale a lui "motor" (afisaza unitatile la noile lor pozitii);pt. a nu redesena de fiecare data si "zid"-urile, se va proceda astfel: in fis. "outputt.cpp" declaram doua entitati cu vizibilitate limitata la el:
"static int pictvect[MAXNU][2], npict;"
avand semnificatia:
"npict" = nr. unitatilor <> "zid" afisate ultima data; initial e 0 (asa il va seta metoda "init");
"pictvect" = lista ceulelor de pe ecran ocupate ult.data de unitati <>"zid"; "pictvect[i][0]", "pictvect[i][1]" = poz.pe ecran a cel.i;
iar metoda "afisare" lucreaza astfel:
- pt.0<=i<npict pune pe ecran in poz."picvetc[i][0]","pictvect[i][1]" un ' ' (adica sterge unitatile <> "zid" afisate ultima data);
- seteaza "npict" cu 0;
- parcurge "unitvect"-ul lui "motor" si pt. fiec.unitate testeaza cu "gettip" daca e <> "zid"; daca da, deseneaza informatia sa (aflata cu "gettip") pe ecran la poz. aflata cu "getx", "gety" si introd. aceasta pozitie la sfarsitul lui "pictvect", incrementand "npict".

Clasa "output" tb. scrisa indep. de celelalte fisiere din proiect, a.i. daca vrem sa modif.jocul pt.ca afisarea sa se faca altfel (de ex. in mod grafic, nu in mod text) sa fie necesara doar rescrierea clasei "output".

* Fisierul "tanc.cpp" cu programul principal; in esenta el are urm. structura:
- include "input.h", "outputt.h", "joctanc.h"
- se declara global obiectele: "input intrare;"
"output iesire(tanc::motor);"
(deoarece exista clasa"tanc",exista si ob.static "tanc::motor", care este de fapt "unit::motor" si este engine-ul comun tuturor unit.de orice tip);
- se initializeaza tabla, de ex: "tanc::motor.init(78,47);"
- se citesc de la tastatura numele si comenzile a doua tancuri, dupa care ele se creaza dinamic: "new tanc(....)"
"new tanc(....)"
- se gen. un labirint de ziduri facand o serie de apeluri: "new zid(....)" (cu diversi parametrii alesi inteligent, a.i. sa apara un desen frumos);
- se apeleaza: "intrare.init(); iesire.init();"
- se efectueaza ciclul: "do{
tanc::cc=intrare.getc();
tanc::motor.next();
iesire.afisare();
}while(tanc::cc!=27);"
- se apeleaza: "intrare.close(); iesire.close();".

Tema 30.

Jocul vietii, folosind engine-ul de la probl.anterioara. Proiectul contine:
* Fisierele "engine" (".h" si ".cpp") de la problema anterioera.

* Fisierele "jocv" (".h" si ".cpp") care descriu aplicarea engine-ului de mai sus la jocul vietii. Ele contin clasa "fiinta", descendenta a lui "unit" si clasele "planta", "ierbivor", "carnivor",descendente ale lui "fiinta"(toate vor mosteni acelasi engine static "motor"). Pe langa ele, fisierul ".cpp" poate contine si alte definitii cu vizibilitate limitata la el (ele nu vor aparea si in fisierul ".h") - de ex. tipul enumerare cu mesajele pe care si le pot trimite unitatile cu "sendmes", printre ele "KILL". Clasele sunt:
- clasa "fiinta", care def. caract. comune ale fiintelor; nu se va instantia direct ci doar prin descendenti; ea mosteneste public pe "unit" si are:
membri data privati:
"static int nrfiinte" = contor static (comun tuturor fiintelor) cu nr. total de fiinte existente in "motor"; initial este 0; la fiecare creare /distrugere a unei fiinte (apel "fiinta"/"~fiinta") creste/scade cu 1;
"int energieoptima" = pragul de en. de la care se schimba comportamentul (hranire/reproducere); e o constanta care depinde de fiecare tip de fiinta (va fi setat de constructorii descendentilor lui "fiinta");
"int timpramas" = timpul de viata ramas (la fiecare apel "action" scade cu 1 - vezi mai jos);
membru data protected:
"int energie" = energia curenta a fiintei (e protected ca sa poata fi mai usor modificata de catre descendenti);
metoda mostenita ca virtuala abstracta si redefinita ca privata:
"int action()" = algoritm:
- daca nu a trebuit sa se apere ("!apara") atunci:
daca "energie < energieoptima" atunci "mananca", altfel "inmulteste" (pt. metodele "apara", "mananca", "inmulteste" vezi mai jos);
- imbatraneste si flamanzeste: "timpramas" si "energie" scad cu 1;
- daca vreunul din "timpramas" sau "energie" a ajuns <=0, atunci "die";
metode protected:
"fiinta(int,int,int,int,int)" = constructorul (fiind protected, nu vom putea clasa "fiinta" direct, ci doar prin descendenti);parametrii sunt: poz. x, y in "motor" (furnizata constructorului "unit"),energia optima, durata vietii (cu ea se init."timpramas"), energia initiala (cu ea se init. "energie"); dupa init. membrilor data increm. pe "nrfiinte" cu 1;
"~fiinta()" = destructorul; decrementeaza pe "nrfiinte" cu 1;
metode protected virtuale si pure:
"virtual int apara()=0" = actiunea de aparare; in descendenti va ret. 1 daca fiinta a trebuit sa efectueze o manevra de aparare si 0 altfel;
"virtual void mananca()=0" = actiunea de hranire;
"virtual void inmulteste()=0" = actiunea de inmultire;
"virtual void behaviour(int)=0"=met.vir.mosten.de la "unit"si nedef.inca;
metode publice:
"static int getnrfiinte()" = returneaza "nrfiinte";
"int getenergie()" = ret."energie"(e folosita pt.a afla en.curenta atunci cand nu ne aflam intr-un loc unde putem accesa direct membrul"energie");
"virtual void* gettip()=0" = met.virt.mosten.de la "unit" si nedef. inca;
- clasa "planta"; o unitate "planta" nu se deplaseaza, nu se apara, se

hraneste cu viteza constanta ("energie" creste cu viteza constanta) si
gen. cate un singur fiu odata;clasa"planta"mosten.public pe"fiinta"si are:
membru data privat:
"static unsigned char nume" = contine caracterul 177; adresa acestei
zone va fi ret.de"gettip" iar continutul ei va fi folosit la afisare;
metode mostenite ca virtuale abstracte si redefinite ca private:
"int apara()" = ret. 0 (planta nu face manevre de aparare);
"void mananca()"=increm."energie"cu 2 (planta se hran. cu viteza const.);
"void inmulteste()" = testeaza pozitiile adiacente (pozitia sa se afla
cu "getx", "gety", mostenite de la "unit"); daca gaseste una valida
("validxy(...)") si libera ("getunitxy(...)==NULL") atunci creaza o
planta acolo ("new planta(...)") iar energia sa "energie" scade cu o
anumita fractie (de ex. 1/30); la un apel genereaza <= 1 fiu;
"void behaviour(int pm)" = daca "pm<0" atunci "energie" scade cu "-pm"
(un asemenea mesaj este trimis de un ierbivor care mananca din planta -
a se vedea mai jos); daca "pm==KILL" atunci "die";
metode publice:
"planta(int,int)"=constructorul; parametri: poz. x, y in "motor";apeleaza
"fiinta(x,y,100,100,50)"(const.pot fi schimb.la latit.programatorului);
"void* gettip()" = returneaza "&nume";
- clasa "ierbivor"; o unitate "ierbivor" se apara fugind de carnivore, se
se hraneste cautand si mancand plante si genereaza cate un singur fiu
odata; clasa "ierbivor" mosteneste public pe "fiinta" si are:
membru data privat:
"static unsigned char nume" = contine caracterul 'i'; adresa acestei
zone va fi ret. de "gettip" iar continutul ei va fi folosit la afisare;
metode mostenite ca virtuale abstracte si redefinite ca private:
"int apara()" = algoritm:
- testeaza un patrat 11 X 11 in jurul poz.sale (aflata cu"getx","gety")
si retine nr.si pozitia carnivorelor aflate in acest patrat; daca
nu gas. nici un carnivor iese si ret.0 (nu face manevre de aparare);
- pt. poz. sa si pentru fiecare poz. adiacenta ei care este valida si
libera calculeaza suma distantelor la carnivorele detectate mai sus
si o retine pe cea in care suma e maxima;
- daca poz. retinuta mai sus este diferita de cea curenta,se muta acolo
("move") si ret. 1, altfel nu se muta nicaieri si ret. 0;
"void mananca()" = algoritm:
- testeaza poz. adiacente pozitiei sale si daca gaseste o planta manaca
din ea, apoi iese ("return"); la un apel mananca din <= 1 planta;
mancatul din planta se face astfel:
- daca en.plantei(aflata cu"getenergie") este >10, atunci "energie"
proprie creste cu 10, iar plantei ii trimite(cu"sendmes")mesajul
-10 (ulterior functia "behaviour" a plantei va trata acest mesaj
prin decrementarea cu 10 a energiei ei - a se vedea mai sus);
- daca energia plantei este <= 10, atunci "energie" proprie creste
cu aceasta energie, iar plantei ii trimite mesajul "KILL";
(constanta 10 poate fi schimbata la latitudinea programatorului);

- (aici ajunge daca n-a gasit nici o planta mai inainte) testeaza
un patrat 11 X 11 in jurul poz. sale si retine nr. si poz. plantelor
aflate in acest patrat;daca nu gaseste nici o planta iese("return");
- pt. pentru fiecare poz. adiacenta poz. sale care este valida si
libera calculeaza minimul distantelor la plantele detectate mai sus
si o retine pe cea in care acest minim e minim; s-ar putea sa nu
existe poz.adiacente valide si lib. si atunci nu retine nici o poz.;
- daca mai sus s-a retinut o pozitie, se muta acolo ("move");
"void inmulteste()" = testeaza pozitiile adiacente celei curente si
daca gaseste una valida si libera atunci genereaza un ierbivor acolo
("new ierbivor(...)") iar energia sa "energie" scade cu o anumita
fractie (de ex. 1/30); la un apel genereaza <= 1 fiu;
"void behaviour(int pm)" = daca "pm==KILL" atunci "die";
metode publice:
"ierbivor(int,int)"=constructorul; parametri: poz.x, y in"motor";apeleaza
"fiinta(x,y,100,100,50)"(const.pot fi schimb.la latit. programatorului);
"void* gettip()" = returneaza "&nume";
- clasa "carnivor"; o unitate "carnivor" nu se apara, se hraneste cautand si
mancand ierbivore si genereaza cate un singur fiu odata; clasa "carnivor"
mosteneste public pe "fiinta" si are:
membru data privat:
"static unsigned char nume" = contine caracterul 'c'; adresa acestei
zone va fi ret. de "gettip" iar continutul ei va fi folosit la afisare;
metode mostenite ca virtuale abstracte si redefinite ca private:
"int apara()" = returneaza 0;
"void mananca()" = algoritm:
-test.poz.adiacente si daca gaseste un ierbivor il manaca(i.e."energie"
proprie creste cu en. ierbivorului (aflata cu"getenergie"),iar lui ii
trimite "KILL"),apoi iese("return");la un apel mananca <= 1 ierbivor;
-(aici ajunge daca n-a gasit ierbivori mai inainte) testeaza un patrat
11 X 11 in jurul poz. sale si retine nr. si poz. ierbiv. aflati in
acest patrat; daca nu gaseste nici unul iese ("return");
-pt. pentru fiecare poz. adiacenta care este valida si libera calc.
minimul distantelor la ierbiv. detectati mai sus si o retine pe cea
in care acest minim e minim; s-ar putea sa nu existe poz.adiacente
valide si libere si atunci nu retine nici o pozitie;
-daca mai sus s-a retinut o pozitie, se muta acolo ("move");
"void inmulteste()" = testeaza pozitiile adiacente celei curente si
daca gaseste una valida si libera atunci genereaza un carnivor acolo
("new carnivor(...)") iar energia sa "energie" scade cu o anumita
fractie (de ex. 1/30); la un apel genereaza <= 1 fiu;
"void behaviour(int pm)" = corp vid (carnivorii nu vor primi mesaje);
metode publice:
"carnivor(int,int)"=constructorul; parametri:poz. x, y in"motor";apeleaza
"fiinta(x,y,100,200,50)"(const.pot fi schimb.la latit. programatorului);
"void* gettip()" = returneaza "&nume".
* Fisierele "outputjv" (".h"si".cpp") asemanatoare fisierelor "outputt" de la

problema anterioara, cu urmatoarele diferente:
- metoda "init" doar deseneaza conturul tablei;
- metoda "afisare" sterge si reafis.toate unitatile (aici nu exista unitati nemuritoare ca unitatile "zid");in plus,dupa fiecare reafis.a tablei,scrie dedesubt nr.generatiei respective;pt.numarare se def.in fis."outputjv.cpp" un contor static, vizibil doar in acest fisier; "init" il initializ. cu 0.

* Fisierul "jv.cpp" cu programul principal; in esenta el are urm. structura:
- include "outputjv.h", "jocv.h"
- se declara global obiectul: "output iesire(fiinta::motor);"
- se initializeaza tabla, de ex: "fiinta::motor.init(78,47);"
- se gen. o configuratie initiala pe tabla prin apeluri "new planta(...)", "new ierbivor(...)", "new carnivor(...)";
- se apeleaza: "iesire.init();"
- se efectueaza ciclul: "do{
  fiinta::motor.next();
  iesire.afisare();
  delay(50);
 }while(fiinta::nrfiinte>0);
- se apeleaza: "iesire.close();".

TEST PRACTIC

(Problema Razboi Naval)

Autor: Mircea Digulescu

Enunt

Intr-un tinut indepartat doua mari puteri (Albastru si Rosu) se afla in mijlocul unei confruntari navale. Golful in care se desfasoara confruntarea este un dreptunghi de dimensiune N x N in care sunt situate Nave. Navele ambelor parti au ajuns in teatru de operatiuni inainte de declansarea confruntarii si deja ocupa anumite pozitii. Datorita vitezei cu care se desfasoara razboiul naval in zilele noastre, navele NU apuca sa se miste (sa isi schimbe pozitia) pe durata confruntarii. Toate Navele ocupa pozitii disjuncte (nu se intersecteaza). Ambele parti dispun de 4 categorii de Nave: Salupa, Submarin, Distrugator si Crucisator. Toate Navele au o Pozitie si o Forma (sunt alcatuite din mai multe patrate 1x1). Ele sunt echipate cu rachete de atac, avand eventual si alte abilitati speciale si sufera in mod diferit consecintele unui atac. Toate Navele dispun de un anumit nivel de Combustibil (intreg) si, ca urmare a operatiunilor de lupta, au suferit un anumit grad de avarie (0-100). O Nava care a ajuns la **Gradul de avarie 100** sau care a ajuns cu **Combustibilul la 0** se considera distrusa si nu mai participa la operatiuni.

Conflictul naval se desfasoara pe Randuri. In cadrul unui Rand, toate navele aleg (simultan) o actiune dintre cele pe care pot sa le intreprinda. Actiunea isi produce efectele la inceputul randului urmator. La finalul fiecarui Rand, o Nava consuma o cantitate de combustibil (in L) egala cu numarul de patratele al Formei sale (e.g. pentru Distrugator, 3 Litri) pentru intretinere. Actiunea de *Regenerare* consuma **5 Litrii** de combustibil si scade *Gradul de Avarie* cu **14** (dar nu la mai putin de 0). Lansarea unei rachete de atac reduce stocul de astfel de rachete cu **1**. Se poate lansa maxim 1 racheta de atac / Rand de catre o Nava.

Mai multe informatii despre fiecare categorie de Nava se regasesc in tabelul de mai jos:

Informatii / Nava	**Salupa**	**Submarin**	**Distrugator**	**Crucisator**
Forma	*	* *	* * *	** ** ** sau **** ** ** (simplu) (cu sistem aparare)
Resurse	Combustibil Rachete de atac	Combustibil Rachete de atac	Combustibil Rachete de atac	Combustibil Rachete de atac
Grad de avarie in cazul atacului cu rachete	+100	+50	+35	+20* * in caz ca nu sunt interceptate (vezi mai jos)
Actiuni posibile	Nimic Atac cu rachete	Nimic Atac cu rachete Autodistrugere	Nimic Atac cu rachete Regenerare	Nimic* Atac cu rachete* Regenerare*

Nivel initial de combustibil	55	225	300	N/A (difera de la nava la nava)

** - Suplimentar, pe langa actiunea principala, Crucisatoarele dotate cu aparare antiracheta, pot opta sa armeze acest sistem.*

Precizari suplimentare:

- O actiune a unei Nave este *disponibila* (poate fi efectuata) doar daca exista suficiente Resurse pentru efectuarea ei. Utilizatorul va alege *doar dintre actiunile disponibile!*
- O Nava cu grad de avarie de 70 sau mai mare, *nu poate lansa atacuri* acel Rand.
- Un Crucisator echipat cu sistem de aparare poate, la fiecare rand, sa opteze suplimentar sa Armeze acest sistem de aparare, pe langa actiunea obisnuita pe care o efectueaza. Armarea sistemului consuma **2 Litrii** de combustibil. Daca acest sistem este armat, toate rachetele lansate in respectivul rand care ar lovi acel Crucisator, au o sansa de 75% sa fie distruse, caz in care nu produc daune.
- Activarea mecanismului de Autodistrugere a unui Submarin conduce la distrugerea completa (avarie +100) a tuturor navelor ale caror forme se intersecteaza cu Zona Tinta. Zona tina este alcatuita din toate patratelele ale caror coordonate (X si Y) difera prin cel mult 2 de cele ale unui patratel din forma Submarinului.
- Regenerarea consuma **5 Litri de combustibil** si are drept efect reducerea gradului de avarie a respectivei Nave cu **14** (dar el nu poate scade sub 0).
- Lansarea unei rachete de catre o Nava presupune alegerea (de catre Utilizator, de la tastatura) a patratelului (coordonate X si Y) tinta. Daca Forma unei Nave cuprinde acel patratel, respectiva Nava se considera tintita si sufera (la randul urmator) consecintele specifice.

Lupta consta intr-o succesiune de Randuri, cat timp exista Nave nedistruse apartinand ambelor parti. Daca la inceputul unui Rand exista Nave nedistruse apartinand unei singure parti, acea parte se considera INVINGATOARE. Daca la finalul unui Rand nu ramane nici o Nava nedistrusa, lupta se considera REMIZA.

Desfasurarea unui Rand:

In cadrul unui rand, urmatoarele au loc, in ordine:

1. Se afiseaza pe ecran numarul de Nave apartinand fiecarei parti care inca nu sunt distruse.
2. Regenerarea produce efecte asupra Navelor care au ales aceasta actiune Randul anterior.
3. Atacurile isi produc efectele, eventual ducand la distrugerea unor Nave. De asemenea, Navele ramase fara combustibil sunt distruse in acest pas.
4. Fiecare Nava (a fiecarei parti), **inca nedistrusa**, opteaza pentru FIX o actiune in randul curent (mai putin Crucisator echipat cu sistem de aparare care poate, suplimentar opta sa activeze si acest sistem). Actiunea dorita este selectata de utilizator de la tastatura, *dintre cele disponibile*. Odata aleasa o actiune (pentru o Nava), se introduc si eventualii parametrii suplimentari (e.g. coordonatele X,Y ale tintei atacului).
5. Navele consuma cantitatea de combustibil specifica pentru intretinere.

Cerinta

Realizati un program in C++ care, pornind de la o configuratie initiala a campului de lupta (citita din fisier sau de la tastatura), simuleaza o lupta navala, conform specificatiilor din enunt, pe baza actiunilor alese de utilizator de la tastatura pentru fiecare Nava la fiecare rand. La final afisati rezultatul luptei (Partea INVINGATOARE / REMIZA).

Aspectele pentru care nu exista o cerinta explicita (e.g. formatul datelor de intrare/interactiunii cu utilizatorul) ramane la latitudinea voastra. Formatul in care se citesc datele de intrare si structura interactiuni cu utilizatorul este la latitudinea voastra. Exemplul prezentat nu are caracter normativ in aceasta privinta.

In realizarea programului este obligatorie folosirea urmatoarelor concepte de OOP: **Clase**, **Mostenire (Derivare)**, **Incapsulare**, **Alocare dinamica a memoriei (cu new)**, **Functii virtuale**.

Suplimentar, folositi cel putin 2 din urmatoarele 5 concepte de OOP: Accesibilitatea (ascunderea) membrilor, Membrii static/const, Supraincarcarea functiilor, Supraincarcarea operatorilor, Conversii.

Atentie: Conceptele folosite trebuie sa fie aplicate in mod adecvat in rezolvarea cerintei problemei.

Evaluarea va tine cont de urmatoarele elemente ale programului realizat: Corectitudine, Calitatea Design-ului, Folosire adecvata si utila a Conceptelor de OOP, Calitatea Codului Sursa (Claritate, etc.).

Atentie: Sursele care nu se compileaza, vor fi notate cu **1p**, indiferent de continut. Este recomandat sa specificati in comentarii mediul de dezvoltare folosit.

Recomandari

In realizarea programului, comisia recomanda sa proiectati urmatoarele clase:

- **Forma** – clasa care reprezinta o colectie de blocuri ce alcatuiesc forma unei nave
- **Nava** – clasa *de baza* pentru Nave
- **Clase derivate pentru fiecare nava in parte**
- **Actiune** – clasa ce reprezinta o actiune aleasa de utilizator pentru o anumita Nava

Avand in vedere ca testul vizeaza evaluarea cunostintelor de programare OOP, este *suficient* sa determinati ce nava(e) se afla la o coordonata parcurgand de fiecare data toate Navele din teatrul de operatiuni, fara a tine vreo structura de date speciala pentru imbunatatirea performantei.

Precizari finale

- Timpul efectiv de lucru, inclusiv citirea enuntului este de **90 de minute**.
- Accesul la Internet este **interzis**.

Exemplu

Exemplu de date de intrare pentru configuratia initiala (1 submarin, 1 distrugator, 2 crucisatorare):

Rosu X 0 Y 1 Submarin 10 racheteatac **|** Albastru X 1 Y 2 Crucisator false 500 Litri 30 racheteatac **|** Rosu Crucisator X 1 Y 6 true 400 Litri 20 rachateatac **|** Albastru Distrugator X 10 Y 6 10 racheteatac **.**

Exemplu de interactiune cu Utilizatorul:

Randul 1: Computer: Albastru – 2 Nave, Rosu – 2 Nave; **Computer: Alegeti actiune pentru Submarin Rosu (X=0,Y=0) – Atac, Autodistrugere, Nimic**; **Utilizator: Autodistrugere;** Computer: Alegeti actiune pentru Crucisator Albastru (X=1,Y=2) – Atac, Regenerare, Nimic; Utilizator: Atac; Computer: Coordonatele tinta (X, Y)? Utilizator: 1 5; **Computer: Alegeti actiune pentru Crucisator Rosu (X=1,Y=6) – Atac, Regenerare, Nimic; Utilizator: Atac; Computer: Coordonatele tinta (X, Y)? Utilizator: 10 6; Computer: Doriti sa activati sistemul de aparare al Crucistatorului? Utilizator: Da**. Computer: Alegeti actiune pentru Distrugator Albastru (X=10,Y=6) – Atac, Regenerare, Nimic; Utilizator: Nimic.

Randul 2: Computer: Albastru – 1 Nave, Rosu – 1 Nave; **Computer: Alegeti actiune pentru Crucisator Rosu (X=1,Y=6) – Atac, Regenerare, Nimic; Utilizator: Atac; Computer: Coordonatele tinta (X, Y)? Utilizator: 10 6; Computer: Doriti sa activati sistemul de aparare al Crucistatorului? Utilizator: Nu**. Computer: Alegeti actiune pentru Distrugator Albastru (X=10,Y=6) – Atac, Regenerare, Nimic; Utilizator: Atac. Computer: Coordonatele tinta (X, Y)? Utilizator: 1 6.

Randul 3: Computer: Albastru – 1 Nave, Rosu – 1 Nave; **Computer: Alegeti actiune pentru Crucisator Rosu (X=1,Y=6) – Atac, Regenerare, Nimic; Utilizator: Atac; Computer: Coordonatele tinta (X, Y)? Utilizator: 10 6; Computer: Doriti sa activati sistemul de aparare al Crucistatorului? Utilizator: Da**. Computer: Alegeti actiune pentru Distrugator Albastru (X=10,Y=6) – Regenerare, Nimic; Utilizator: Nimic.

Randul 4: Computer: Albastru – 0 Nave, Rosu – 1 Nava. **ROSU ESTE INVINGATOR**.