Laborator Programare Orientata pe Obiecte

Proiect 2

Toate proiectele de la punctul 1 pana la 12 vor urmari urmatoarele proprietati:

Proiectele se vor testa pe obiecte de tip numere complexe si intregi implementate de voi si pe numere mari intregi/rationale/reale, pe care le veti obtine de la colegii vostri care au implementat deja aceste tipuri de data ca proiectul 1. Se va considera valid proiectul doar in momentul in care reusiti sa integrati unul dintre proiectele colegilor vostri in proiectul vostru :) .

Proiectele 13, 14, 16, 17 si 18 vor trebui sa implementeze clasa suport ca si clasa de tip Singleton.

Setari Code::Blocks dorite: Project -> Built Options -> bifati *Have g++ follow the C++11 ISO C++ language standard [-std=c++11]*.

1. Lista dublu inlantuita de obiecte ce respecta o interfata specifica. Functionalitate:
    a. supraincarcare *operator*[] pentru a obtine un obiect la o pozitie specificata
    b. metode de adaugare a unui nou obiect la final, la inceput si in interiorul listei la o pozitie specificata; lista se indexeaza de la 0.
    c. metode de stergere a obiectului de la final, de la inceput si din interiorul listei la o pozitie specificata.
    d. metoda de obtinere a dimensiunii listei
    e. metoda de sortare a obiectelor

    Interfata obiectelor:
    a. contine metode necesare afisarii (.ToString ())
    b. contine o metoda de comparare a doua obiecte

2. Coada de obiecte ce respecta o interfata specifica. Coada trebuie implementata folosind lista dublu inlantuita ce este detaliata la punctul 1 (mostenire). Functionalitate:
    a. metode de adaugare a unui element la finalul cozii
    b. metode de obtinere a elementului din varful cozii
    c. metode de stergere a elementului din varful cozii
    d. metoda de obtinere a dimensiunii cozii
    e. desi este implementata mostenind clasa de liste, clasa de cozi trebuie sa ascunda functionalitatea listei

    Intefata obiectelor:
    a. contine metode necesare afisarii (.ToString ())

3. Stiva de obiecte ce respecta o interfata specifica. Stiva trebuie implementata folosind lista dublu inlantuita ce este detaliata la punctul 1 (mostenire). Functionalitate:
    a. metode de adaugare a unui element in varful stivei
    b. metode de obtinere a elementului din varful stivei
    c. metode de stergere a elementului din varful stivei
    d. metoda de obtinere a dimensiunii stivei
    e. desi este implementata mostenind clasa de liste, clasa de cozi trebuie sa ascunda functionalitatea listei

    Interfata obiectelor:
    b. contine metode necesare afisarii (.ToString ())

4. Vector de obiecte ce respecta o interfata specifica. Functionalitatea acestui obiect este urmatoarea: Initial vectorul va aloca dinamic o zona continua de memorie de o dimensiune initial fixa. Intern vom considera doua zone de memorie:
    a. *zona alocata* care va reprezenta toata memoria alocata initial
    b. *zona folosita* care va reprezenta toata memoria utilizata de obiectele continute de vector. Aceasta zona este pastrata la inceputul zonei de

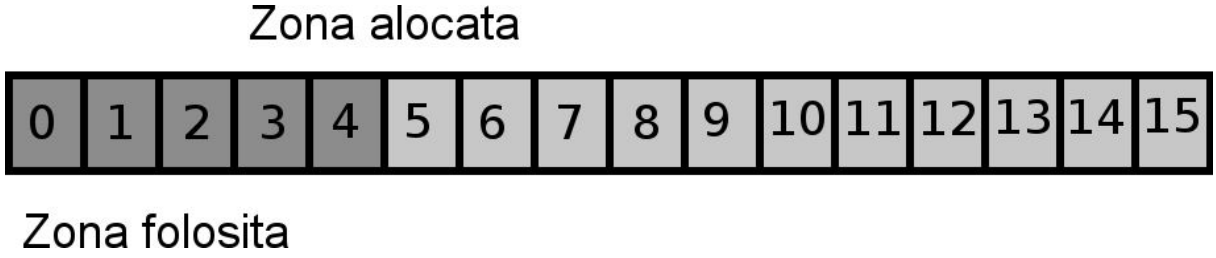

Figura 1

memorie alocate. Initial, cand vectorul este gol, zona de memorie folosita are lungimea 0.

Vectorul trebuie sa contina o metoda de adaugare a unui nou obiect la finalul lui. La apelarea acesteia se extinde zona folosita si se adauga noul obiect.

O imagine face cat 1000 de cuvinte, astfel ca o sa urmarim un exemplu cu Figura 1. Consideram ca vectorul pastreaza valori de tip *int*. Se aloca initial (in mod dinamic), 16 zone continue de memorie de 4 octeti fiecare. Presupunem ca primele 5 zone sunt folosite, iar restul libere in momentul actual. Intern se pastreaza informatia ca s-au ocupat 5 pozitii din vector. Cand se apeleaza metoda de adaugare a unui nou obiect se ia blocul cu indicele 5 din Figura (este indexat de la 0), i se atribuie valoarea obiectului si se modifica informatia interna care pastreaza numarul de pozitii folosite din vector.

De asemenea, este necesar ca vectorul sa prezinte urmatoarele functionalitati:

a. supraincarcare *operator*[]
b. metode de adaugare un nou obiect la final.
c. metode de stergere a unui obiect de la final si din interior, la o pozitie specificata. Metoda de stergere se va implementa cu o alunecare la stanga
d. metoda de sortare a obiectelor
e. metoda de cautare binara/secventiala a unui obiect in interior
f. daca memoria zonei folosite devine egala cu cea a zonei alocate si se incearca introducerea unui nou obiect, se va aloca o alta zona de memorie, de dimensiune dubla fata de cea actuala, se vor copia

elementele din locatia curenta in cea noua si se elibereaza memoria zonei curente.

Interfata obiectelor interne trebuie sa respecte urmatoarele proprietati:

a. contine metode necesare afisarii (.ToString ())
b. contine o metoda de comparare a doua obiecte

5. Matrice ale carei elemente trebuie sa respecte o interfata. Functionalitate:
   a. supraincarcare operator [][] dublu
   b. supraincarcare operatori *, +, -, /, >, <, ==, !=

   Interfata obiectelor din matrice trebuie sa prezinte urmatorele proprietati:
   a. contine metode necesare afisarii (.ToString ())
   b. contine metode pentru inmultire, adunare, scadere, impartire si comparare

6. Coada de prioritati implementata folosind structura de heap. Obiectele continute de coada de prioritati trebuie sa respecte o interfata specifica. Functionalitate:
   a. metoda de adaugare a unui obiect nou
   b. obtinerea si eliminarea primului element din coada de prioritati

   Interfata obiectelor din matrice trebuie sa prezinte urmatoarele proprietati:
   a. contine o metoda de comparare

7. Tabele de dispersie (Hash Table) ale carei elemente prezinta o interfata. Modelul de memorie poate fi asemanator celui prezentat la proiectul 4. In momentul in care se depaseste dimensiunea tabelei de dispersie trebuie sa se aloce o noua zona de memorie de dimensiune mai mare si sa se readauge toate obiectele folosind functia de hashing. Functionalitate:
   a. supraincare *operator* [] care sa raspunda la intrebarea daca exista valoarea in tabela
   b. contine metode de adaugare, stergere, apartenenta

   Interfata obiectelor componente:
   a. contine o metoda de hashing a obiectului care trebuie sa intoarca o valoare intreaga pozitiva pe 32 de biti

8. Tabele de dispersie (Map) care pastreaza informatie ca pereche de cheie si valoare. Valoarea este necesar sa respecte o interfata. Modelul de memorie poate fi asemanator celui prezentat la proiectul 1. In momentul in care se depaseste dimensiunea tabelei de dispersie trebuie sa se aloce o noua zona de memorie de dimensiune mai mare si sa se readauge toate obiectele folosind functia de hashing. Functionalitate:
    a. supraincarcare $operator\,[](cheie)$ care sa raspunda la cererea de returnare a valorii asociate cheii
    b. contine metode de adaugare, stergere, apartenenta a cheii

   Interfata obiectelor componente:
    a. nu trebuie sa contina nicio metoda specifica, dar din ea trebuie mostenit orice obiect transmis ca valoare in pereche

   Cheia:
    a. o valoare intreaga pozitiva pe 32 de biti. Se va lasa in grija utilizatorului sa nu trimita aceeasi valoare pentru toate obiectele. O mica observatie: cheia nu reprezinta valoarea de hashing, ea este doar un identificator pentru obiectul valoare

9. Arbori binari de cautare ce se vor comporta ca structura de la punctul 8 (Hash Table). Obiectele continute de arbore trebuie sa respecte o interfata. Functionalitate:
    a. supraincarcare $operator\,[]$ care sa raspunda la intrebarea daca exista obiectul in arbore
    b. metode de adaugare, stergere, apartenenta

   Interfata:
    a. contine o metoda de comparare

10. Arbori binari de cautare ce se vor comporta ca structura de la punctul 9 (Map). Obiectele continute de arbore trebuie sa respecte o interfata specifica. Functionalitate:
    a. supraincarcare $operator\,[]\,(cheie)$ care sa raspunda la cererea de returnare a valorii asociate cheii

b. contine metode de adaugare, stergere, apartenenta a cheii

Interfata:

a. Nu trebuie sa contina nicio metoda specifica, dar din ea trebuie mostenit orice obiect transmis ca valoare in pereche

Cheia:

a. o valoare intreaga pozitiva pe 32 de biti. Se va lasa in grija utilizatorului sa nu trimita aceeasi valoare pentru toate obiectele. O mica observatie: cheia nu reprezinta valoarea de hashing, ea este doar un identificator pentru obiectul valoare

11. Implementare structura de date asemanatoare celei de la punctul 9 folosind un arbore binar de cautare echilibrat (AVL, Red Black Tree, Treap).

12. Implementare structura de date asemanatoare celei de la punctul 10 folosind un arbore binar de cautare echilibrat (AVL, Red Black Tree, Treap)

13. Creati un editor de text. Nu este importanta interfata grafica (evident). Considerati ca in momentul in care vreti sa editati textul de pe ecran, veti edita un string pastrat intern. Editorul de text trebuie sa prezinte urmatoarele operatii de baza:
    a. Operatia de inserare a unui substring in interiorul stringului vostru de editor. Considerati ca cititi executia aceasta de la fluxul de intrare, cin (in locul editarii bine cunoscute cu interfata grafica ce se gaseste in gedit, vim sau notepad++ :)). Query-ul va avea forma:
       INSERT <pozitie> <sir de caractere>
    b. Operatia de stergere a unui substring undeva in interiorul textului vostru. La fel ca la operatia a, query-ul de citire va avea forma:
       ERASE <capat stanga> <capat dreapta>
    c. Operatia de transformare a unui substring in alt string. La fel ca la operatia a, query-ul de citire va avea forma:
       EDIT <capat stanga> <capat dreapta> <string de inlocuit>

De asemenea, editorul vostru va trebui sa contina doua operatii de nivel inalt care vor reprezenta si scopul acestei probleme:

a. Operatia de Undo care va sterge ultima operatie de baza executata pana in acel moment. De exemplu, dupa doua insert-uri, un erase si inca un insert, doua operatii succesive de undo vor face revert la ultima operatie de insert si la penultima operatie de erase, astfel incat string-ul vostru va ajunge la starea dupa doua insert-uri. Query-ul operatiei va reprezenta doar sirul:
   UNDO
b. Operatia de Redo, considerata ca inversa operatiei de Undo. Va re-exuta ultima operatie la care s-a facut Undo. Daca in exemplul anterior, dupa doua operatii de Undo se vor face doua operatii de Redo stringul de editor va reveni la starea de dupa doua insert-uri, un erase si un insert. Query-ul operatiei va reprezenta doar sirul:
   REDO

14. Creati un alocator de memorie. Alocatorul va trebui sa aloce un spatiu continuu, destul de mare, de octeti in memoria RAM a calculatorului (alocare dinamica clasica). Dupa alocarea acestei zone de memorie, va trebui sa returnati adrese din interiorul ei la cerere si sa stergeti zone din interior la cerere.
    a. Operatia de alocare de memorie va trebui sa returneze un pointer intern zonei alocate de voi initial spre o zona continua ce contine n octeti liberi nealocati pana in acel moment (n este de forma 2k,k10). Programul va trebui sa contina o functie cu urmatorul prototip:
       void* malloc (int numar_octeti);
    a. Operatia de dealocare de memorie va primi o adresa de memorie la care s-a efectuat o alocare la un pas anterior folosind operatia de la punctul a. Aceasta va avea scopul de a semnala programului vostru ca memoria de la adresa respectiva este libera si evident va putea fi alocata la un pas ulterior. Programul vostru va trebui sa contina o functie cu urmatorul prototip:
       void free (void* adresa_alocata_anterior);

Se doreste ca acest alocator de memorie sa fie unic in tot programul si de preferat global motiv pentru care va trebui sa aveti urmatoarea functionalitate:

a. supraincarcare *operator new* global
b. supraincarcare *operator new*[] global
c. supraincarcare *operator delete* global
d. supraincarcare *operator delete*[] global

15. Creati un Smart Pointer.

Descriere: Un smart pointer este un pointer asemanator celui clasic cunoscut in limbajul C, cu singura exceptie ca este smart. Pastreaza informatie in legatura cu obiectele care au referinta spre el si stie sa nu stearga blocul de memorie la care adreseaza daca mai sunt obiecte care il folosesc.

Practic luam urmatorul scenari: Avem 5 obiecte care pastreaza un smart pointer catre un acelasi obiect alocat dinamic de interes pentru toate cele 5 obiecte. La un moment dat in evolutia programului unul din cele 5 obiecte va incerca sa stearga obiectul alocat dinamic deoarece nu mai este de interes pentru el. Smart pointer-ul va observa ca se incearca stergerea lui, dar in acelasi timp va fi constient ca mai sunt alte 4 obiecte pentru care mai este de interes. In acest moment smart pointer-ul nu va face nimic. Daca toate cele 5 obiecte vor incerca sa il stearga va trebui sa se stearga deoarece stie ca de abia atunci nu mai este de interes pentru niciun obiect din program.

Ca observatie practic, mare atentie la implementare pentru ca apar probleme (la copiere, atribuire, stergere).

Se va testa acest smart pointer pe aceleasi obiecte ce se testeaza si proiectele de 1 la 12 (specificat la inceputul fisierului).

16. Creati un manager de resurse. Consideram urmatorul scenariu:

Se da urmatorul joculet 2D (“Lost Garden Community - Free Game Graphics”): Figura 2

Se observa cei patru braduti numerotati care sunt identicii. Jocul pastreaza intern un vector de obiecte de tip bradut pentru fiecare astfel de obiect din lume. Fiecare obiect pastreaza informatii despre coordonate, collider dar si o imagine cu bradutul.

Problema: Fiecare obiect pastreaza o copie a aceleiasi imagini a bradutui. In cazul celor 4, se vor pastra 4 obiecte de tip imagine, dar care din punct vizual sunt identice. Din acest motiv se consuma foarte multa memorie in mod inutil.

Creati un manager de resurse pentru pentru managementul acestor resurse comune (imaginile). Manager-ul va pastra intern un dictionar (Map, proiectul 8, 10, 12) de perechi (label, pointer catre imagine). Obiectele de tip bradut vor pastra intern doar label-ul asociat imaginii de tip bradut si vor accesa acest manager de fiecare data cand va avea nevoie de imagine, transmitand label asociat ei. Acest manager trebuie sa poata fi folosit nu doar pentru obiectele de tip bradut, dar si pentru obiectele de tip copacei (din screenshot), roci (din partea de sus a screenshot-ului) si orice alt obiect din imagine.

Se va incerca simularea acestui scenariu folosind siruri de caractere in locul imaginilor efective si obiecte dummy in locul obiectelor propriu-zise din joc. Functionalitatea manager-ului fiind scopul acestui proiect.

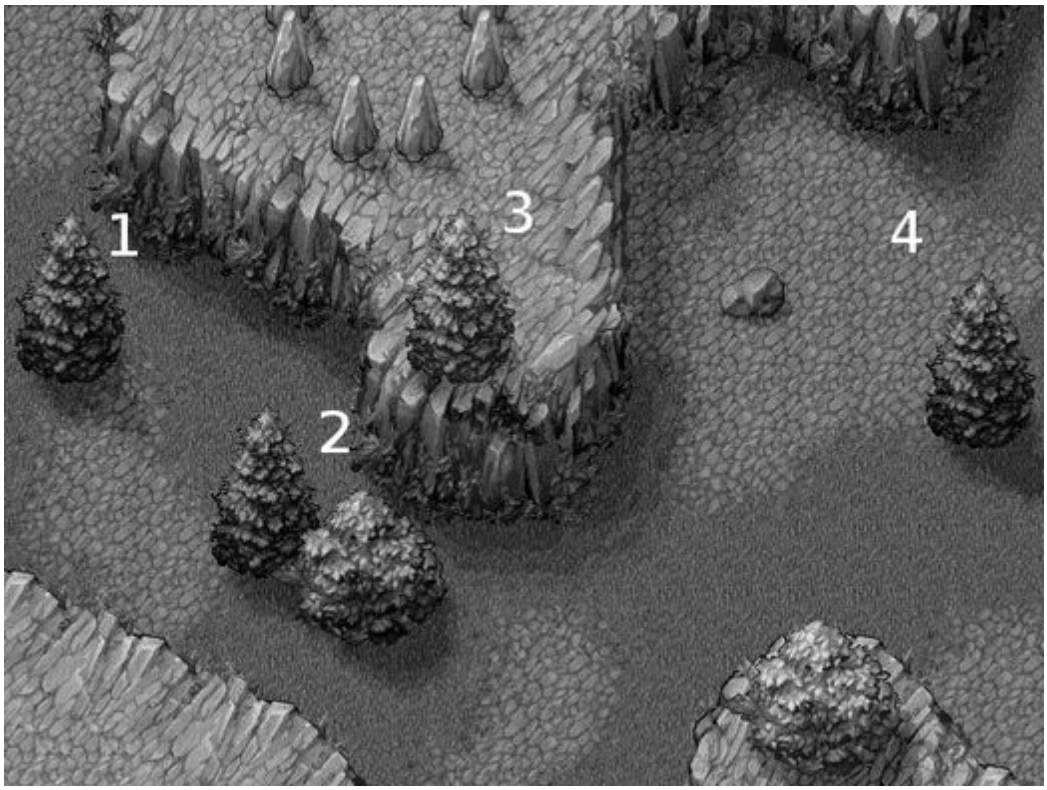

Figura 2

17. Creati un manager de input care sa functioneze in felul urmator. Inputul va fi considerat introducerea anumitor combinatii de taste de la tastatura (tasta sus, jos, stanga in ordinea aceasta). Obiectele se vor “conecta” la manager-ul de input in modul urmator:
    a. Se va trimite la manager doua informatii: un obiect de tip InputPreferences care va contine combinatia dorita si un pointer la o functie cu prototipul

        void Callback (const InputPreferences& inputPreferences);

        Aceasta functie poate fi o metoda a obiectelor ce se conecteaza la manager-ul de input, dar care va trebui sa respecte prototipul.
    b. In momentul in care combinatia de taste se satisface, manager-ul de input apeleaza functia transmisa prin pointer.
    c. O observatie importanta este ca manager-ul poate sa pastreze mai multe combinatii de taste si mai multe functii Callback pentru aceeasi combinatie de taste.

    Aceasta structura de clase este folosita la Input-ul de pe telefoanele mobile cu sistem de operare Android :) .

18. Creati un manager de evenimente care sa functioneze in felul urmator. Vor exista obiecte ce se conecteaza la manager-ul de evenimente printr-o pereche de (string nume_eveniment, functie* pointer). Functia trebuie sa respecte urmatorul prototip:

        void Callback (const EventArgs& args);

    Obiectele vor putea avea doua utilizari diferite.
    a. Sa se conecteze la manager-ul de evenimente
    b. Sa anunte manager-ul de realizarea unui eveniment

    Exemplu:

Presupunem evenimentul de “PlayerDead”. Daca player-ul a murit, player-ul anunta manager-ul ca s-a realizat evenimentul “PlayerDead”, cu argumentul

```
class PlayerDeadEventArgs : public EventArgs
{
private:
	Damage _damage;
	Player* player;
public:
	// getters
	// setters
}
```

In momentul in care s-a anuntat evenimentul, manager-ul de evenimente se uita in lista lui de obiecte conectate si se uita daca exista un obiect conectat la evenimentul “PlayerDead”. Daca exista un obiect conectat la acest eveniment se apeleaza functia Callback (const EventArgs& args) transmisa ca pointer cu argumentul de tip PlayerDeadEventArgs care mosteneste clasa EventArgs.

Trebuie sa aveti grija ca manager-ul de evenimente pastreaza un dictionar (Map, proiectele 8, 10 si 12) cu mai multe perechi de timp (string nume_eveniment, functie* pointer) si pot sa existe mai multe obiecte conectate (in concluzie si mai multe functii) la acelasi eveniment.