Proiectul 1

Veți rezolva unul dintre subiectele de mai jos. Subiectele 1-2 sunt mai “clasice” (adică seamănă mai mult cu ce ați avea de rezolvat la un colocviu), iar subiectele 3-5 sunt mai atipice, dar sper interesante :).

# 1 Spălătorie automată

Veți implementa clasele necesare funcționării unui magazin de tip **laundromat**.

Descriere:

Spălătoria are mai multe mașini de spălat, n la număr, dar nu mai mult decât 5, deoarece spațiul actual poate fi îmbunătățit.

Despre fiecare mașină (**class MasinaSpalat**) cunoaștem dacă este sau nu liberă, durata până la terminarea spălării curente (minute și secunde -> **class Durata**), numărul de capsule de detergent pe care le conține (va folosi o capsulă pentru fiecare spălare).

Utilizatorii vor folosi clasa **Spalatorie** pentru a gestiona afacerea:

1. Crearea unei spalatorii

*int* main() {

Masina masini[5] = {

Masina(*true*, Durata(0, 0), 3),

Masina(*true*, Durata(0, 0), 2),

Masina(*true*, Durata(0, 0), 5),

};

Spalatorie spalatorie(

3,

masini

);

}

1. Metoda si functie de listare a masinilor de spalat:

spalatorie.listeazaMasini(); *// = metoda care listeaza informatii despre fiecare masina de spalat din spalatorie:*

*/\**

*\* Exemplu de output a metodei:*

*\**

*0. Masina este libera*

*1. Masina este neutilizabila! Nu mai are capsule!*

*2. Masina este ocupata pentru inca 1:20*

*\*/*

listeazaMasiniLibere(spalatorie); *// functie globala care afiseaza doar masinile libere. Afiseaza indicele lor:*

*/\**

*\* Exemplu de output a functiei*

*\**

*Urmatoarele masini sunt libere: 0 2*

*\*/*

1. Metoda pentru adaugarea unei spalari pentru o anumita masina de spalat

spalatorie.adaugaJob(

1,

Durata(1, 20)

); *// va decrementa numarul de capsule, va retine noua durata, dar si ca nu mai este libera maxina*

* Daca masina de spalat nu ar mai avea capsule de detergent, s-ar afisa mesajul:

“**Masina 1 nu mai are capsule!”***, unde 1 este indicele* masinii.

1. Metoda care actualizeaza datele despre fiecare masina (daca a terminat spalarea, daca mai are capsule, daca este libera), daca trece o anumita durata de timp.

spalatorie.actualizeazaTimp(Durata(1, 0));

1. Metodele for fi folosite intr-o succesiune logica, precum:

// declararea spalatoriei

Masina masini[5] = {

Masina(*true*, Durata(0, 0), 3),

Masina(*true*, Durata(0, 0), 2),

Masina(*true*, Durata(0, 0), 5),

};

Spalatorie spalatorie(

3,

masini

);

// listare initiala

spalatorie.listeazaMasini();

listeazaMasiniLibere(spalatorie);

spalatorie.adaugaJob(

1,

Durata(1, 20)

);

spalatorie.listeazaMasini();

spalatorie.actualizeazaTimp(Durata(1, 19));

spalatorie.listeazaMasini();

spalatorie.actualizeazaTimp(Durata(1, 0));

spalatorie.listeazaMasini(); *// metoda*

*// adaugam inca o spalare la masina 1, ca sa ramana fara capsule*

spalatorie.adaugaJob(

1,

Durata(0, 35)

);

spalatorie.actualizeazaTimp(Durata(3, 0));

*// Incercam sa adaugam inca o spalare. Ar trebui sa afiseze mesajul "Masina 1 nu mai are capsule!"*

spalatorie.adaugaJob(

1,

Durata(0, 35)

);

*// Ceea ce se poate observa si din listarea masinilor:*

spalatorie.listeazaMasini();

* Output-ul acestui exemplu de rulare va fi:

Spalatoria in prezent:

0. Masina este libera

1. Masina este libera

2. Masina este libera

Urmatoarele masini sunt libere: 0 1 2

Spalatoria in prezent:

0. Masina este libera

1. Masina este ocupata pentru inca 1:20

2. Masina este libera

Spalatoria in prezent:

0. Masina este libera

1. Masina este ocupata pentru inca 0:01

2. Masina este libera

Spalatoria in prezent:

0. Masina este libera

1. Masina este libera

2. Masina este libera

Masina 1 nu mai are capsule!

Spalatoria in prezent:

0. Masina este libera

1. Masina este neutilizabila! Nu mai are capsule!

2. Masina este libera

Process finished with exit code 0

# 

# 2 Listă de așteptare

Veți implementa clasele necesare funcționării unui aplicatii de tip lista de asteptare (clientul face o comanda si primeste un numar), care va avea diverse actiuni: comanda, anulare comanda, gestionare plata (clientul primeste banii inapoi). Actiunile vor fi mereu acompaniate de mesaje clare (afisate pe consola).

Descriere:

Aveti un vecin antreprenor care si-a deschis in timpul liber un magazin de clatite. Din cauza popularității afacerii, intampinati zilnic o problema: se formeaza o coada atat de mare, nu reusiti sa va intrati in bloc fara sa va ingramaditi (despre modalitatile legale in care a obtinut permisul de a deschide afacerea fara sa aiba o intrare speciala a clientilor, nu vom discuta ;).)

Tocmai de aceea, aplicatia voastra va gestiona comenzile si va putea raspunde la intrebari precum cat va dura pana un anumit client poate ridica comanda, care este lista actuala de comenzi etc.

Pentru a scrie cod elegant, ati decis sa creati o clasa **ListaDeAsteptare** care va contine date despre cele n **Comenzi** din prezent.

Despre fiecare comanda cunoastem **numele clientului**, **numarul de telefon**, **durata de pregatire** (un numar dintre 3 si 5, ales aleator, cu o metoda aleasa de voi), dar si un **id** unic, care va fi afisat la listarea tuturor comenzilor.

Aplicatia va putea fi utilizata datorita urmatoarelor functionalitati:

1. Crearea unei instante a listei de asteptare, pentru a putea primi comenzi

*int* main() {

*// 1. crearea unei instante de lista de asteptare*

*// Putem fie sa adaugam cateva comenzi*

Comanda comenzi[100] = {

Comanda("Stefanescu", "0733222744", 3, 3),

Comanda("Mihailescu", "0752222745", 4, 4),

Comanda("Popescu", "0799222746", 3, 6)

};

ListaDeAsteptare lista(3, comenzi); *// totalComenzi, adica id-ul unic al urmatoarei comenzi,*

*// il vom putea gasi in constructor ca fiind maximul+1 dintre id-urile fiecarei comande (max(3,4,6)=6 => totalComenzi=6+1=7)*

*// sau putem sa folosim constructorul default, iar numarComenzi si totalComenzi vor incepe de la 0:*

ListaDeAsteptare lista2;

}

1. Adaugarea unei noi comenzi

lista.adaugaComanda("Prastie",

"0786001002"); *// idUnic si minutePregatire vor fi generate corespunzator in interiorul clasei*

* Include si afisarea timpului minim pana la ridicarea comenzii si a timpului de pregatire. (mesaj de genul in X minute ne vom putea ocupa de comanda dumneavoastra, care va putea fi ridicata in X+Y minute - X=timpul de asteptare, Y=timpul de pregatire aleator)

1. Afisarea listei de asteptare cu id-urile comenzilor

lista.afisareOrdine();

// va afisa “Ordinea comenzilor: 3 4 6 7”

* Id-urile vor trebui sa fie unice. O modalitate de a face asta este folosirea unei numar **totalComenzi** in clasa ListaDeAsteptare care va incepe de la valoarea 0 si va creste cu 1 pentru fiecare comanda, ceea de ne ofera numarul urmatoarei comenzi plasate.

1. Eliminarea unei comenzi, care va determina inclusiv reafisarea automata a listei de asteptare updatata (deoarece fiecare dintre comenzile ramase vor fi la stanga)

// afiseaza “Comanda 4 a fost eliminata cu succes.”

lista.eliminaComanda(

4); *// va elimina comanda cu id-ul unic egal cu 4. Daca nu exista, va afisa un mesaj corespunzator.*

// afiseaza

// “Nu exista aceasta comanda.”

// “Comenzile existente: 3 6 8”

lista.eliminaComanda(

1001); *// va elimina comanda cu id-ul unic egal cu 1001. Daca nu exista, va afisa un mesaj corespunzator.*

1. O metoda care va permite derularea cu un numar de minute

// Nu puteti derula cu mai mult de 8 minute, deoarece depaseste timpul total!Timp de 3 minute,

// lista are urmatoarele modifi cari:

lista.adaugaTimp(10);

// Timp de 3 minute, lista are urmatoarele modificari:

// Comanda cu id-ul 6 poate fi ridicata chiar acum!

lista.adaugaTimp(3);

// Timp de 1 minute, lista are urmatoarele modificari:

// Comanda cu id-ul 8 este in curs de pregatire si va fi gata in 4 minute!

lista.adaugaTimp(1);

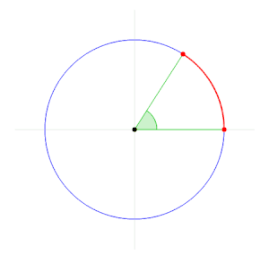
* Atentie ca daca introduceti un numar mare, aplicatia va stii sa afiseze comenzile care au fost ridicate in aceasta perioada de timp. Gasiti si o modalitate de a nu permite folosirea unui numar de minute care ar depasi durata tuturor comenzilor din prezent.

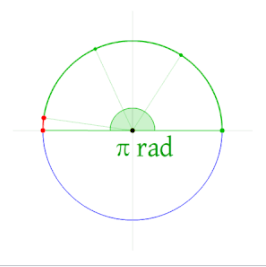
# 

# 3 Puțină trigonometrie dezaburită

Scopul este implementarea unei singure clase de la zero, în care vom include constructori, getteri, setteri, functii membre si operatori (**nu sunt necesari**, deoarece vom invata despre ei in laboratorul 4).

1. Unghiurile sunt folosit des în lumea reală - arhitectură, aranjamente florale, monitoare, pentru a nu aluneca laptopul de pe o masă înclinată, benzi de alergare etc). Precum majoritatea lucrurilor care au rezistat mult timp și măsurarea gradelor a trecut prin multe schimbări. Noi ne vom concentra pe două modalități clasice de a reprezenta unghiurile:
   1. Grade sexazegimale, minute și secunde de arc. Sau mai simplu, componentele notației **15°6‘30“** (15 grade, 6 minute, 30 secunde)
   2. Radiani (cel mai probabil mai ciudat definit.. Un radian este echivalent cu numărul de grade ale unui **unghi** care defineste un **arc** egal cu **raza cercului**:





Problema care apare odată cu notațiile multiple este că deși avem aplicații care fac calculele pentru noi, precum aplicații grafice diverse (CAD=Computer Aided Drawing, plotarea funcțiilor, crearea de animații) sau limbaje de programare diferite, notația poate să difere: de grad sexazegimal, ori notația de radiani, sau chiar notație centazimală (despre care nu discutăm).

Clasa Angle va avea scopul ca, de îndată ce am implementat-o corect, aceste calcule și conversii să fie mult mai ușor de realizat.

Mai exact, clasa va putea fi folosită în aceste moduri:

1. Crearea de unghiuri

| Angle a1 (23, 30, 10); // sexazegimal  Angle a2 (PI / 2); // unghi de 90 de grade, cu input in radiani  Angle a0; // exista si constructor default/neparametrizat |
| --- |

Hint: pe PI il puteti declara ca o constanta globala:

**const double PI = 3.141592653589793; // sau acos(0) \* 2;**

1. Operatii pe unghiuri

| Angle a3 = aduna(a1, a2); // unghiul a3 va avea 113 grade 30 minute 10 secunde    a3.print(); // afiseaza "113 30 10"  cout << a3.radians() << endl; // afiseaza aproximativ 1.9809971824 rad |
| --- |

1. Functii trigonometrice

| cout << a2.cos() << endl; // va afisa 0 |
| --- |

Indicii:

Scopul este ca instructiunile sa poate fi rulate iar afisarile sa coincida cu cele din comentariile de dupa fiecare instructiune in parte.

Important este sa luati pas cu pas fiecare instructiune.

Atentie ca puteti folosi functiile preimplementate in cmath pentru a calcula cos, dar ele au nevoie de radians ca input, deci va fi prima data nevoie de functia membra .radians() care calculeaza cati radiani are unghiul

1. Compunerea de clase

Vom crea o clasa Triunghi care va contine 3 unghiuri. Scopul vostru este sa creati o metoda simpla care primeste ca parametru lungimea celei mai mici dintre laturile triunghiului si raspunde la urmatoarele intrebari:

1. Care este unghiul opus celei mai mici laturi?
2. Care sunt lungimile tuturor celor 3 laturi?
3. Care este aria triunghiului?
4. Triunghiul este ascutit unghic?

Inca o observatie. Suntem smart coders, daca vrem sa rezolvam asta vom crea cate o metoda pentru fiecare cerinta, iar in metoda care afiseaza toate aceste informatii sa le utilizam pe rand.

Ar fi fost frumos prin liceu, dar si in cercetare, sa avem aplicatii care fac matematica si calculele pentru noi :). Cam asta implementam si acum.

Daca va loviti de matematica inca o data, ganditi-va ca scopul vostru este sa gasiti o solutie pentru a demonstra ca va puteti dezvolta mai bine (la finalul zilei veti putea face ceea ce nu puteati face la inceputul zilei ;) .), si aveti sansa de a cerceta si de a ma intreba pe mine cum puteti rezolva acest exercitiu sau ce va lipseste matematic.

# 4 Clasă care criptează și decriptează un mesaj

Vom folosi o metoda super simplista pentru a cripta un mesaj. Acest sistem de criptare va avea sens doar in cazul in care **cheia** este privata.

Mai exact, consideram ca doua persoane au decis de la inceput ca aleg o "cheie", un string de maxim 42 de caractere.

De indata ce cheia a fost aleasa, criptarea va avea loc prin aplicarea operatiei XOR (^ in C++) caracter cu caracter, asupra mesajelor trimise de cele două persoane.

**Spre exemplu**

Cream sistemul de criptare cu cheia "POO justice".

De indata ce am ales aceasta cheie, fiecare mesaj "transmis" va fi ixorat (adică i se aplica XOR) cu cheia. In cazul in care **cheia este mai scurta** decat mesajul, cheia este reluata circular de la inceput:

| Cheie | P | O | O | ' ' | j | u | … |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ASCII cheie | 01001000 | 01100001 | 01110000 | 01110000 | 01111001 | 00100000 | 01001100 |
| Mesaj criptat (cheie[i] ^ mesaj[i]) | 00011000 | 00101110 | 00111111 | 01010000 | 00010011 | 01010101 | 00111111 |
| ASCII mesaj | 01010000 | 01001111 | 01001111 | 00100000 | 01101010 | 01110101 | 01110011 |
| Mesaj necriptat | H | a | p | p | y | ' ' | … |

Pentru a verifica daca criptarea a fost corectă, am implementat o functie care afiseaza tabelul de mai sus:

| void afisareBinara(char c) {  for (int i = 7; i >= 0; --i) {  if (((1 << i) & c) != 0) {  cout << 1;  } else {  cout << 0;  }  }  }    void testPrint(char original[], char cheie[], char criptat[]) {  // deoarece nu stim sigur daca ^ a determinat sau nu valoarea 00000000 ('\0') in timpul operatiei,  // vom folosi strlen(original)  cout << " | ";  for (int i = 0; i < strlen(original); ++i) {  afisareBinara(cheie[i % strlen(cheie)]);  cout << " | ";  }  cout << '\n';  cout << " | ";  for (int i = 0; i < strlen(original); ++i) {  afisareBinara(criptat[i]);  cout << " | ";  }  cout << '\n';  cout << " | ";  for (int i = 0; i < strlen(original); ++i) {  afisareBinara(original[i]);  cout << " | ";  }  cout << '\n';  } |
| --- |

Daca va intrebati, motivul pentru care operatia folosita este XOR (^) este pentru ca operatia este inversabila. Mai exact

Daca

mesaj[i] ^ cheie[i] == z

Atunzi

z ^ cheie[i] == mesaj[i]

Așadar dacă aplicăm „ixorarea” de două ori putem recupera mesajele originale. Bineînțeles, doar cei care au acces la cheia folosite pot face asta.

**Urmeaza sa adaug si aici cateva cerinte mai clare. Am mers pe ideea de a avea cerinte cat mai clare si gradual nu vom descrie exact metodele pe care le folositi, ci faptul ca cititi anumite date si creati aplicatia care duce la rezultatul corect :).**

Descrierea claselor:

Aici vom avea o clasa denumita **Crypt**, care va fi initializata cu cheia secreta si va putea sa critpteze orice mesaj cu ajutorul acesteia.

De asemenea, pentru a invata totusi compunerea de clasei, vom avea o clasa **ConversatieCriptata** care contine un array/vector cu obiecte de tipul **MesajCriptat.**

ConversatieCriptata contine un obiecte de tipul Crypt, care va fi folosit pentru fiecare mesaj criptat, si lista de mesaje criptate.

Functionalitati de implementat

1. Vom putea crea un Crypt si o conversatie Criptata
2. Vom putea adauga un mesaj conversatie

* Va fi salvat in varianta criptata

1. Vom putea afisa unul dintre mesaje

* In acest caz, vom folosi Crypt pentru a decripta mesajul inainte de afisare

1. Vom putea actualiza cheia de criptare

* In acest caz, vom dori ca primul pas sa fie actualizarea fiecarui dintre mesajele retinute, prin a le cripta cu noua cheie

# 5 Clasa HangMan

Creați o clasa HangMan care va putea rula următorul cod:

int main() {

HangMan game{"Thug POO"};

game.printStats(); /\*

functia membra va afisa:

Guess \_ \_ \_ \_ \_ \_ \_

Tried letters:

You have 6 guesses left.

\*/

game.printUnknownWord(); // Va afisa: "\_ \_ \_ \_ \_ \_ \_"

game.quess('P'); // Va afisa: "You hit the spot! Now you only have to guess \_ \_ \_ \_ \_ P \_ \_"

game.quess('o'); // Va afisa: "You hit the spot! Now you only have to guess \_ \_ \_ \_ \_ P O O"

game.printStats(); /\* va afisa:

Guess \_ \_ \_ \_ \_ P O O

Tried letters: PO

You have 6 guesses left.

\*/

game.quess('z'); // "Unlucky! That was not it, but you still have 5 guesses left"

game.quess('t');

game.quess('h');

game.quess('u');

game.quess('g');

// dupa ultimul apel se va afisa: "Congratulations! You've guessed the WORD won the GAME"

// si mai puteti implementa o functie care verifica daca jocul s-a terminat:

cout << game.playerHasWon() << endl; // va afisa 1, deoarece jucatorul a castigat

}

Indicii:

Nu va conta performanta. Ideea in mare este ca in clasa putem retine cuvantul care trebuie ghicit si numarul de incercari ramase, plus orice modalitate simpla stiti de a retine care litera au fost incercate de catre jucator (i.e. s-a rulat game.guess('P')). Codul poate fi implementat in 80 linii de cod fara nicio problema.

Luati fiecare instructiune din main si deduceti ce aveti nevoie sa implementati in clasa HangMan ca apelul de functie sa mearga. Spre exemplu:

Linia de cod:

game.printUnknownWord(); // Va afisa: "\_ \_ \_ \_ \_ \_ \_"

Practic ne spune ca in clasa exista o metoda care nu returneaza nimic dar afiseaza mesajul de mai sus, in care fiecare \_ reprezinta una dintre cuvintele inca neghicite. Observam si ca exista un spatiu cand mesajul original contine spatiu :).