PROIECT FINAL C++

CUPRINS

1 Descriere....2
1.1 Recomandari....2
1.2 Criterii de evaluare:....3
2 Specificatiile programului....4
2.1 Modulele programului....4
2.2 Modulul Calcule....5
2.2.1 Operatiile aritmetice....5
2.2.2 Calculele combinatoriale....5
2.2.3 Valori de test....5
2.2.4 Sirurile numerice....6
2.2.4.1 Clasa de baza Sir....6
2.2.4.2 Clasa Fibonacci....7
2.2.4.3 Clasa Lucas....7
2.2.4.4 Clasa Pell....8
2.2.4.5 Clasa Triangular....8
2.2.4.6 Clasa Pentagonal....8
2.2.4.7 Clasa Prime....9
2.2.4.8 Clasa TestPrim....10
2.2.4.9 Valori de test....10
2.2.5 Structuri pentru exceptii....10
2.3 Modulul UI....11
2.3.1 Clasa Comanda....12
2.3.2 Clasa ComandaCalcule....12
2.3.3 Clasa CmdAritmetice....12
2.3.4 Clasa CmdCombinatorial1....13
2.3.5 Clasa CmdCombinatorial2....13
2.3.6 Clasa CmdSiruri....14
2.3.7 Clasa CmdExit....14
2.3.8 Clasa Meniu....14
2.3.9 Clasa AfiseazaComanda....16
2.3.10 Clasa MeniuOperatiiAritmetice....16
2.3.11 Clasa MeniuCombinatorial....17
2.3.12 Clasa MeniuSiruri....17
2.3.13 Clasa MeniuPrincipal....17
2.4 Modulul Calculator....18
2.4.1 Clasa Aplicatie....18
2.5 Functia main....18
3 Directoarele si fisierele proiectului....19

1 DESCRIERE

Proiectul final consta in implementarea unui calculator folosind toate modelele de programare suportate de C++ si toate conceptele predate la curs folosind specificatiile acestui document. In elaborarea acestor specificatii am urmarit aplicarea tuturor conceptelor cu o cantitate minima de cod scris si efort de implementare.

1.1 Recomandari

Pentru a putea implementa cu succes aces proiect este necesara parcurgerea practica a tuturor laboratoarelor, aplicarea ***etapelor dezvoltarii programelor***: `design` (intelegerea designului in cazul de fata), `implementare`, `testare`, `depanare` pentru fiecare functionalitate adaugata la proiect. De asemenea recomand folosirea ***spiralei*** ca metodologie de implementare a proiectului[1].

In mod practic aceasta inseamna adaugarea unei noi functionalitati (metoda, clasa, functie) la proiect numai dupa ce ***versiunea curenta se compileaza cu succes si rezultatele obtinute sunt corecte***. Veti avea astfel intregul proces sub cotrol.

Functionalitatile vor fi implementate in ordinea prezentarii lor in acest document.

Stergeti fisierele si directoarele temporare inainte de trimiterea arhivei cu proiectul, fie in stadiul final, fie in stadii intermediare. Directoarele temporare sunt: `Debug`, `Release`, `ipch`.Fisere temorare sunt: `*.sdf`, `*.suo`.

Scopul principal al acestui proiect este sa va asiste in asimilarea conceptelor si sa depasiti nivelul de dezvoltator incepator C++. Daca aveti dificultati sau neclaritati in oricare dintre etape[2], incercati sa revizuiti conceptele si sintaxa din suportul de curs, revedeti laboratorul pentru acea lectie. Daca nu puteti gasi o solutie in maxim o ora va rog sa ma contactati prin e-mail pentru a clarifica problema cat mai eficient.

Daca aveti erori de compilare: cititi mesajul de eroare, localizati zona de cod unde apare acest mesaj cu dublu click pe mesaj, revizuiti sintaxa codului, cautati pe internet documentatia pentru codul de eroare afisat in fereastra *Output* din Visual C++. Daca nu ati putut rezolva eroarea trimiteti arhiva prin e-mail inainte de a adauga alt cod.

Daca descoperiti erori in documentatie

Am implementat personal acest proiect folosind specificatiile, este posibil ca anumite erori sa existe in document cu toate ca am verificat textul. Daca descoperiti astfel de erori, va rog sa-mi trimiteti detaliile pentru a le putea corecta cat mai rapid.

1 Concepte discutate in prima lectie a cursului.
2 Si in special in intelegerea designului.

1.2 Criterii de evaluare:

1. Implementarea corecta a functionalitatii. Aceasta presupune compilarea cu succes a proiectului si obtinerea mesajelor de functionare normala si de eroare specificate si obtinerea rezultatelor corecte.

2. Minimizarea directivelor de `include` in fisierele header si sursa si folosirea corecta a spatiilor de nume. Acesta este un aspect important in activitatea practica de programare. Atunci cand este posibil se folosesc declaratiile anticipate.

3. Stilul de scriere a codului: folosirea constantelor simbolice si enumerate, denumirea variabilelor si constantelor, spatiile, identarea, alinierea acoladelor.

4. Organizarea corecta a directoarelor conform specificatiilor.

5. Stergerea fisierelor si directoarelor temporare inainte de trimiterea arhivei.

2 SPECIFICATIILE PROGRAMULUI

Sa se scrie un program care efectueaza urmatoarele calcule: operatii aritmetice intregi (`adunare`, `scadere`, `inmultire`, `impartire`, `modulo`), calcule combinatoriale (`factorial`, `aranjamente`, `combinari`) si siruri numerice (`fibonacci`, `lucas`, `pell`, `triangular`, `pentagonal` si `prime`).

Programul va avea o interfata utilizator cu mesaje text care prezinta utilizatorului o lista de comenzi si cere selectarea indexului comenzii in mod asemanator cu meniurile telefonice.

Cele trei categorii de calcule sunt grupate in trei liste de comenzi numite meniuri. Aceste meniuri sunt afisate in lista initiala numita meniu principal.

Dupa introducerea indexului comenzii, programul valideaza aceasta valoare si reafiseaza lista de comenzi in cazul in care indexul este invalid.

Daca indexul este valid, programul executa acea comanda solicitand valorile de calcul in intervalul specificat de utilizator. Rezultatele sunt afisate pe ecran si programul asteapta apasarea tastei enter pentru a continua. In acest mod, utilizatorul are posibilitatea citirii rezultatelor. Dupa executie programul afiseaza meniul principal.

Daca apar erori de calcul (impartire la zero, depasirea intervalului de valori etc) programul afiseaza informatia de eroare in locul rezultatului si, dupa ce utilizatorul apasa tasta enter, se afiseaza din nou meniul principal.

Proiectul se transmite intr-o arhiva zip stergand in prealabil directoarele si fisierele temporare.

2.1 Modulele programului

Programul are 3 module: `Calculator`, `Calcule` si `UI` reprezentate in Fig 2.1. Modulul `Calcule` contine codul pentru functionalitatea de calcule. Modulul `UI` contine functionalitatea pentru interfata utilizator. Modulul `Calculator` contine codul apelat din functia main.

Fiecare modul are asociat un `namespace`. Spatiul de nume `Calculator` este containerul pentru cele doua subspatii `Calcule` si `UI`. De asemenea, fisierele fiecarui modul se afla in directoarele cu numele spatiilor de nume.

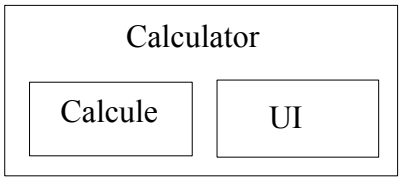

Fig 2.1:

2.2 Modulul Calcule

2.2.1 Operatiile aritmetice

Operatiile aritmetice sunt definite ca functii template cu un singur parametru tip generic in fisierul: `operatii.h`. Pentru impartire si modulo se ridica o exceptie de tipul `ImpartireLaZero` daca al doilea parametru este 0.

2.2.2 Calculele combinatoriale

Pentru calculele combinatoriale se folosesc functii declarate in fisierul `combinatorial.h` si definite in fisierulul `combinatorial.cpp`.

`int Factorial(int n)` – care calculeaza n! cu formula:

$$n! = \begin{cases} 1 & n = 0 \\ n*(n-1)*(n-2)*\cdots*2*1 & n \in [1-12] \end{cases}$$

folosind o implementare nerecursiva. Daca n nu este in intervalul specificat se ridica o exceptie de tipul `InvalidN`.

Valori de test: 0! = 1! = 1, 3! = 6

`int Aranjamente(int n, int k)` – calculeaza aranjamente de n luate cate k cu urmatoarea formula:

$$A_n^k = \frac{n!}{(n-k)!}, \quad n \in [0-12], \quad 0 \le k \le n$$

Daca n nu este in intervalul specificat se ridica o exceptie de tipul `InvalidN`, iar daca valoarea lui k nu este in interval se ridica o exceptie de tipul `InvalidK`.

`int Combinari(int n, int k)` – calculeaza combinari de n luate cate k cu urmatoarea formula:

$$C_n^k = \frac{A_n^k}{k!} \quad n \in [0-12], \quad 0 \le k \le n$$

Daca n nu este in intervalul specificat se ridica o exceptie de tipul `InvalidN`, iar daca valoarea lui k nu este in interval se ridica o exceptie de tipul `InvalidK`.

2.2.3 Valori de test

n	k	A_n^k	C_n^k
4	1	4	4
4	2	12	6
4	3	24	4
4	4	24	1

2.2.4 Sirurile numerice

Pentru implementarea sirurilor numerice se va folosi programarea obiectuala. Toate clasele vor fi declarate in fisierul **siruri.h**, metodele acestor clase vor fi definite in fisierul **siruri.cpp**.

Pentru simplificarea implementarii, functionalitatea comuna tuturor sirurilor numerice va fi implementata in clasa de baza **Sir**. Algoritmii de calcul pentru elementele sirurilor numerice se implementeaza in clasele derivate: **Fibonacci**, **Lucas**, **Pell**, **Triangular**, **Pentagonal** si **Prime**. Pentru sirul numerelor prime se va folosi o clasa auxiliara **TestPrim**.

2.2.4.1 Clasa de baza Sir

Definitii de tip:

Aceasta clasa foloseste un container de tip vector pentru elementele sirurilor de tip **unsigned int**. Pentru simplificarea implementarii se adauga urmatoarele definitii de tip in fisierul **siruri.h** inainte de declaratia clasei:

```
typedef unsigned int Uint;
typedef vector<Uint> TVint;
typedef vector<Uint>::const_iterator TIterator;
```

Atribute

Clasa are doua atribute cu nivel de acces **protected** (pentru a putea fi accesate din clasele derivate):

```
Uint _count;   // numarul de elemente
TVint _elemente; // vectorul de elemente
```

Metode:

Toate metodele, cu exceptia cazului cand se specifica explicit alt nivel de acces, au nivelul de acces **public**.

Sir() – initializeaza valoarea lui **_count** cu **0**.

virtual ~Sir() – necesar deoarece este clasa de baza.

void CalculeazaValori(int index) – metoda ***virtual pura*** cu nivel de acces **protected** suprascrisa de clasele derivate pentru calcularea valorilor elementelor.

Uint operator [] (int index) – operatorul de indexare:

```
daca index < 0 ridica o exceptie de tipul ParametruInAfaraIntervalului
_count = index + 1
CalculeazaValori(index)
returneaza valoarea elementului de la pozitia index
```

Sir& operator () (int index) – operatorul functie[3]

```
apeleaza operator [] (index)
returneaza referinta la instanta curenta
```

friend ostream& operator << (ostream &out, const Sir &sir) – operatorul de scriere

```
afiseaza elementele sirului pentru index in intervalul [0 - count) in urmatorul format:
```

3 este folosit in expresii de forma: **cout << sir(index) << endl;** unde sir este o instanta a unei clase derivate

```
scrie urmatoarele mesaje (index si valoare sunt separate printr-un tab):
Index    Valoare
--------------------
Apoi scrie elementele vectorului pe cate o linie in formatul index valoare separate
printr-un tab.
```

Important

Elementele trebuie adaugate in vector folosind metoda **`push_back`** pentru a permite redimensionarea vectorului.

2.2.4.2 Clasa Fibonacci

Este derivata public din clasa **`Sir`**. Are urmatorul atribut public pentru valoarea ultimului index valid:

```
enum { MAX_FIB = 47 }
```

Suprascrie metoda **`void CalculeazaValori(int index);`**

- elementele sirului se calculeaza cu formula:

$$F(n)=\begin{cases} n & n=0,1 \\ F(n-1)+F(n-2) & n\leq 47 \end{cases}$$

```
daca index > MAX_FIB
     ridica exceptia ParametruInAfaraIntervalului

daca dimensiunea vectorului este 0
     se adauga primele doua elemente in vector

apoi pornind de la dimensiunea curenta cat timp dimensiunea < count
     se adauga elementele folosind ultima expresie din formula
```

2.2.4.3 Clasa Lucas

Este derivata public din clasa **`Sir`**. Are urmatorul atribut public pentru valoarea ultimului index valid:

```
enum { MAX_LUCAS = 46 }
```

Suprascrie metoda **`void CalculeazaValori(int index);`**

- elementele sirului se calculeaza cu formula:

$$L(n)=\begin{cases} 2 & n=0 \\ 1 & n=1 \\ L(n-1)+L(n-2) & n\leq 46 \end{cases}$$

```
daca index > MAX_ LUCAS
     ridica exceptia ParametruInAfaraIntervalului

daca dimensiunea vectorului este 0
     se adauga primele doua elemente in vector

apoi pornind de la dimensiunea curenta cat timp dimensiunea < count
     se adauga elementele folosind ultima expresie din formula
```

2.2.4.4 Clasa Pell

Este derivata public din clasa **Sir**. Are urmatorul atribut public pentru valoarea ultimului index valid:

```
enum { MAX_PELL = 26 }
```

Suprascrie metoda **void CalculeazaValori(int index):**

- elementele sirului se calculeaza cu formula:

$$Pell(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ 2*P(n-1)+P(n-2) & n\leq 26\end{cases}$$

```
daca index > MAX_PELL
     ridica exceptia ParametruInAfaraIntervalului


daca dimensiunea vectorului este 0
     se adauga primele doua elemente in vector

apoi pornind de la dimensiunea curenta cat timp dimensiunea < count
     se adauga elementele folosind ultima expresie din formula
```

2.2.4.5 Clasa Triangular

Este derivata public din clasa **Sir**. Are urmatorul atribut public pentru valoarea ultimului index valid:

```
enum { MAX_TRIANGULAR = 1024 }
```

Suprascrie metoda **void CalculeazaValori(int index):**

- elementele sirului se calculeaza cu formula:

$$T(n)=\frac{n*(n+1)}{2}\quad n\in[0-1024]$$

```
daca index > MAX_TRIANGULAR
     ridica exceptia ParametruInAfaraIntervalului


daca dimensiunea vectorului este 0
     se adauga primele doua elemente in vector

apoi pornind de la dimensiunea curenta cat timp dimensiunea < count
     se adauga elementele folosind ultima expresie din formula
```

2.2.4.6 Clasa Pentagonal

Este derivata public din clasa **Sir**. Are urmatorul atribut public pentru valoarea ultimului index valid:

```
enum { MAX_PENTAGONAL = 1024 }
```

Suprascrie metoda **void CalculeazaValori(int index):**

- elementele sirului se calculeaza cu formula:

$$P(n)=\frac{n*(3*n-1)}{2}\quad n\in[0-1024]$$

```
daca index > MAX_PENTAGONAL
     ridica exceptia ParametruInAfaraIntervalului


daca dimensiunea vectorului este 0
     se adauga primele doua elemente in vector

apoi pornind de la dimensiunea curenta cat timp dimensiunea < count
     se adauga elementele folosind ultima expresie din formula
```

2.2.4.7 Clasa Prime

Este derivata public din clasa **Sir**. Are urmatorul atribut public pentru valoarea ultimului index valid:

```
enum { MAX_PRIME = 1024 }
```

Suprascrie metoda **void CalculeazaValori(int index):**

Se cauta urmatorul numar prim:

```
daca index > MAX_PENTAGONAL
     ridica exceptia ParametruInAfaraIntervalului


daca dimensiunea vectorului este 0
      se adauga primele doua numere prime in vector: 2 si 3

pornind de la testPrim[4] = valoarea ulimului element[5] + 2[6],
     cat timp dimensiunea vectorului < count, se incrementeaza testPrim

          it[7] = find_if[8] de la inceput la sfarsit, cu testPrim se cauta un divizor


          daca valoarea iteratorului[9] returnat de algoritm  > testPrim
               se adauga testPrim in vector
```

4 **testPrim** este o instanta de tipul **TestPrim**.
5 valoarea ultimului element se acceseaza apeland: **_elemente.back()**
6 se incrementeaza cu 2 deoarece cu exceptia lui 2 toate numerele prime sunt impare
7 it are tipul **TIterator** declarat in header.
8 Se cauta divizori in vector pentru in intervalul $[3-\sqrt{(n)}]$
9 se foloseste indirectarea pentru iterator

2.2.4.8 Clasa TestPrim

Aceasta clasa este folosita ca si *predicat unar* pentru algoritmul `for_each` si implementeaza suplimentar operatorii de conversie, comparare si incrementare pentru a simplifica deplasarea la urmatoarea valoare. Este declarata in fisierul `siruri.h` si metodele sunt definite in `siruri.cpp`.

Atribute

```
Uint _val;          // valoarea care este testata
double   _stopVal;  // valoarea de oprire a testarii sqrt(_val)
```

Metode

`TestPrim(Uint val)` – constructor initializeaza `_val` si `_stopVal` cu valoarea val respectiv `sqrt(val)`[10].

`TestPrim& operator ++()` – operatorul de incrementare, incrementeaza `_val` cu 2 si ***actualizeaza*** `_stopVal = sqrt(val)`.

`bool operator () (Uint divizor)` – operatorul functie, returneaza `true` daca:

`divizor > _stopVal sau 0 == _val % divizor`.

`operator Uint ()` – operatorul de conversie, returneaza _val, este folosit la adaugarea elementelor in sir.

`friend bool operator > (Uint val, const TestPrim &src)` – operatorul de comparare, returneaza `true` daca: `val > src._stopVal`.

2.2.4.9 Valori de test

index	Fibonacci	Lucas	Pell	Triangular	Pentagonal	Prime
0	0	2	0	0	0	2
1	1	1	1	1	1	3
2	1	3	2	3	5	5
3	2	4	5	6	12	7
4	3	7	12	10	22	11
5	5	11	29	15	35	13

2.2.5 Structuri pentru exceptii

Pentru simplitate, exceptiile sunt doar declarate ca structuri in fisierul `exceptii.h`:

```
struct ImpartireLaZero {};

struct ParametruInAfaraIntervalului {};

struct InvalidK {};

struct InvalidN {};
```

10 este necesara folosirea unui cast: `_stopVal = sqrt(double(_val))`

2.3 Modulul UI

Rolul acestui modul este de a realiza interactiunea cu utilizatorul: afiseaza mesaje informative, citeste datele introduse de utilizator apeleaza functionalitatea de calcul si afiseaza rezultatele sau mesajele de eroare.

Pentru implementarea acestui modul se foloseste ierarhia de clase prezentata in Fig 2.2:

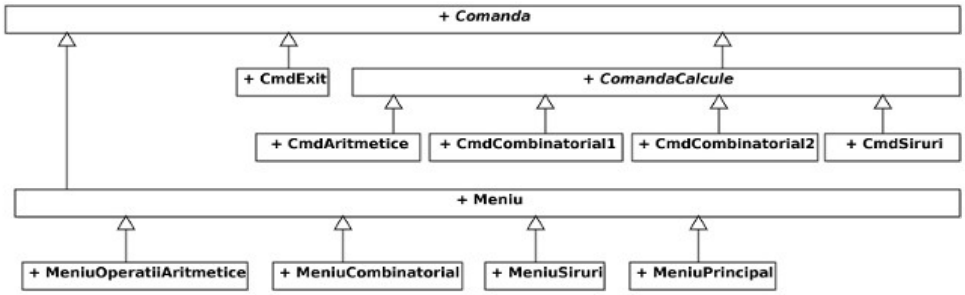

Fig 2.2

La prima vedere poate parea o arhitectura complexa, insa, prin separarea functionalitatilor si folosirea unei clase pentru fiecare responsabilitate, efortul de implementare este redus.

Clasa abstracta de baza `Comanda` grupeaza elementele comune tuturor comenzilor: fiecare comanda are un *nume* si o *actiune* care trebuie executata.

Clasa `ComandaCalcule` permite utilizatorului citirea rezultatelor dupa efectuarea calculelor prin afisarea unui mesaj pe ecran si asteptarea tastei enter. Toate clasele care executa comenzi de calcule sunt derivate din aceasta clasa. Rolul acestor clase este acela de a afisa mesaje de informare, citi valori de la tastatura si afisare rezultate.

Clasele derivate din clasa `ComandaCalcule` sunt clase `template` pentru a putea transfera functiile sau clasele de calcul cu tipuri diferite de date.

- `CmdAritmetice` conecteaza functiile de calcul aritmetic cu interfata utilizator. Se foloseste cate o instanta pentru fiecare functie.
- `CmdCombinatorial1` conecteaza functia Factorial cu interfata utilizator.
- `CmdCombinatorial2` conecteaza functiile *Aranjamente* si *Combinari* cu interfata utilizator. Se foloseste cate o instanta pentru fiecare functie.
- `CmdSiruri` conecteaza sirurile numerice cu interfata utilizator. Se foloseste cate o instanta pentru fiecare tip de sir numeric.
- `CmdExit` permite navigarea catre meniul anterior sau terminarea programului in cazul meniului principal.
- `Meniu` permite gruparea mai multor comenzi.
- `MeniuOperatiiAritmetice` grupeaza comenzile pentru operatiile aritmetice.
- `MeniuCombinatorial` grupeaza comenzile pentru calculele combinatoriale.
- `MeniuSiruri` grupeaza comenzile pentru sirurile numerice.
- `MeniuPrincipal` grupeaza sub-meniurile pentru operatii aritmetice, calcule combinatoriale si siruri numerice.

2.3.1 Clasa Comanda

Pentru aceasta clasa se folosesc fisierele: **comanda.h/cpp**

Atribute private:

const string _nume

Metode publice:

Comanda(const string &nume) – constructor - initializeaza atributul _nume

virtual ~Comanda() – destructor

const string& Nume() const – returneaza numele comenzii.

void AsteaptaEnter() – metoda *virtuala* suprascrisa de clasele derivate pentru asteptarea tastei enter care permite citirea valorilor rezultatelor.

void Execute() - metoda *virtual pura* implementata de clasele derivate.

2.3.2 Clasa ComandaCalcule

Pentru aceasta clasa se folosesc fisierele: **comanda.h/cpp**

Atribute private:

const string _nume

Metode publice:

ComandaCalcule(const string &nume) – constructor initializeaza atributul _nume

void AsteaptaEnter() – suprascrie metoda cu urmatorul cod:

```
cout << "Apasati Enter pentru a continua: ";
cin.ignore(1, '\n');
char ch;
cin.get(ch);
```

2.3.3 Clasa CmdAritmetice

Pentru aceasta clasa se foloseste fisierul: **cmdaritmetice.h**

Aceasta clasa este o clasa template cu urmatoarea lista de parametri generici:

template<int Operatie(int, int)>

Metode publice:

CmdAritmetice(const string &nume)

void Execute():

```
afiseaza mesajul: Introduceti doua numere intregi (x, y).
declara doua variabile intregi
citeste valorile variabilelor (se afiseaza un promt inainte x: respectiv y: )
apeleaza functia Operatie(x,y):
cout << x << " " << Nume() << " " << y  << " = "  << Operatie (x,y) <<endl;
```

2.3.4 Clasa CmdCombinatorial1

Pentru aceasta clasa se foloseste fisierul: **cmdcombinatorial.h**

Aceasta clasa este o clasa template cu urmatoarea lista de parametri generici:

template <int Operatie(int), int MAX, int MIN = 0>

Metode publice:

CmdCombinatorial1(const string &nume)

void Execute():

```
afiseaza mesajul: Introduceti un numar intreg in intervalul [ MIN, MAX] inlocuind MIN si
MAX cu valorile primite in lista de parametri generici
declara declara o variabila intreaga
citeste valoarea variabilei
apeleaza functia Operatie(n):
cout <<  Nume() << "(" << n << ") = "  << Operatie(n) << endl;
```

2.3.5 Clasa CmdCombinatorial2

Pentru aceasta clasa se foloseste fisierul: **cmdcombinatorial.h**

Aceasta clasa este o clasa template cu urmatoarea lista de parametri generici:

template <int Operatie(int, int), int MAX, int MIN = 0>

Metode publice:

CmdCombinatorial2(const string &nume)

void Execute():

```
afiseaza mesajul: Introduceti doua numere intregi (n, k, n > k) in intervalul [MIN , MAX]
inlocuind MIN si MAX cu valorile primite in lista de parametri generici
declara doua variabile intregi
citeste valorile variabilelor (se afiseaza un promt inainte x: respectiv y: )
apeleaza functia Operatie(n,k):
cout << Nume() << "(" << n << ", " << k << ") = " << Operatie(n,k) << endl;
```

2.3.6 Clasa CmdSiruri

Pentru aceasta clasa se foloseste fisierul: **cmdsiruri.h**

Aceasta clasa este o clasa template cu urmatoarea lista de parametri generici:

```
template<typename SirNumere, int MAX, int MIN=0>
```

Atribute private:

```
SirNumere _sir;
```

Metode publice:

```
CmdSiruri(const string &nume)
```

```
void Execute():
```

```
afiseaza mesajul: Introduceti un numar intreg in intervalul [ MIN, MAX] inlocuind MIN si MAX cu valorile primite in lista de parametri generici

declara declara o variabila intreaga

citeste valoarea variabilei

apeleaza operatorul functie pentru atributul _sir:

cout << "Sirul "  << Nume() << " [0 - " << n << "]:\n" << _sir(n) << endl;
```

2.3.7 Clasa CmdExit

Pentru aceasta clasa se folosesc fisierele: **cmdexit.h/cpp**

Atribute private:

```
bool _exit
```

Metode publice:

CmdExit(const string& text = "Inapoi") – initializeaza atributul **_exit** cu **false**.

void Execute() – seteaza **_exit** cu **true.**

bool Exit() – returneaza valoarea atributului **_exit**.

2.3.8 Clasa Meniu

Pentru aceasta clasa se folosesc fisierele: **meniu.h/cpp**

Meniul este o comanda speciala care contine un vector de alte comenzi si permite adaugarea comenzilor, afisarea indexului si numelui comenzilor, citirea indexului comenzii de la tastatura, executia comenzii selectate si afisearea rezultatelor sau mesajelor de eroare. De asemenea, meniul este responsabil pentru eliberarea memoriei ocupate de comenzi.

Definitii de tip:

```
typedef vector<Comanda*> TCommands;
typedef TCommands::size_type Index;
```

Atribute private:

`TCommands _comenzi` – vectorul de comenzi al meniului.

Metode publice:

```
Meniu(const string &nume)
```

`virtual ~Meniu()` – dealoca toti pointerii din vector folosind algoritmul `for_each` impreuna cu functia `void DeleteComanda(Comanda *&pcmd)` definita in fisierul `meniu.cpp` inainte de destructor.

`void Execute():`

```
executa:
    sterge ecranul
    afiseaza numele meniului
    afiseaza comenzile
cat timp CitesteIndex returneaza false

apeleaza comanda cu indexul specificat intr-un bloc try
trateaza exceptiile: ImpartireLaZero, InvalidK, InvalidN si ParametruInAfaraIntervalului
apeleaza AsteaptaEnter pentru comanda selectata
```

Pentru afisarea comenzilor se foloseste algoritmul `for_each` impreuna cu obiectul functie unar `AfiseazaComanda`.

Mesajele de eroare afisate in cazul exceptiilor sunt:

Exceptie	Mesaj
`ImpartireLaZero`	Impartire la zero.
`ParametruInAfaraIntervalului`	Parametru in afara intervalului.
`InvalidK`	Invalid k.
`InvalidN`	Invalid n.

Metode protected:

`void AdaugaComanda(Comanda *comanda)` – adauga o comanda in vector folosind `push_back`.

Metode private:

`void StergeEcran()` – sterge ecranul cu urmatoarea secventa:

```
#if defined(_WIN32) || defined(_WIN64)
     #define ERASE "cls"
#else
     #define ERASE "clear"
#endif
      system(ERASE)[11];
```

`bool IndexValid(Index index)` – returneaza true daca 0 <= index < _comenzi.size()

`bool CitesteIndex(Index &index)`:

```
reseteaza erorile din cin[12]
citeste valoarea indexului

daca cin.fail() == true
     returneaza false

returneaza IndexValid(index)
```

2.3.9 Clasa AfiseazaComanda

Pentru aceasta clasa se folosesc fisierele: **`meniu.h/cpp`**

Atribute private:

`int _index` - indexul comenzii.

Metode publice:

`AfiseazaComanda()` – initializeaza **`_index`** cu 0.

`void operator() (Comanda *comanda):`

```
cout << "   " << _index++ << " " << comanda->Nume() << endl;
```

2.3.10 Clasa MeniuOperatiiAritmetice

Pentru aceasta clasa se folosesc fisierele: **`meniuoperatiiaritmetice.h/cpp`**

11 trebuie inclus headerul **`cstdlib`**

12 se foloseste secventa :

```
     if(cin.fail())
     {
          cin.clear();
          cin.ignore(numeric_limits<std::streamsize>::max(), '\n');
      }
```

Metode publice:

`MeniuOperatiiAritmetice()`

```
apeleaza constructorul clasei de baza cu: Operatii Aritmetice Intregi
adauga commanda exit: AdaugaComanda(new CmdExit);
adauga comenzile aritmetice cu apeluri de forma:
    AdaugaComanda(new CmdAritmetice<operatie[13]>("simbol"));
```

2.3.11 Clasa MeniuCombinatorial

Pentru aceasta clasa se folosesc fisierele: **`meniucombinatorial.h/cpp`**

Metode publice:

`MeniuCombinatorial()`

```
apeleaza constructorul clasei de baza cu: Combinatorial
adauga commanda exit: AdaugaComanda(new CmdExit);
adauga comenzile combinatoriale cu apeluri de forma:
    AdaugaComanda(new CmdCombinatorial#<nume,MAX>("nume"))[14];
```

2.3.12 Clasa MeniuSiruri

Pentru aceasta clasa se folosesc fisierele: **`meniusiruri.h/cpp`**

Metode publice:

`MeniuSiruri()`

```
apeleaza constructorul clasei de baza cu: Siruri Numerice
adauga commanda exit: AdaugaComanda(new CmdExit);
adauga comenzile pentru siruri numerice cu apeluri de forma:
    AdaugaComanda(new CmdSiruri<Nume,Nume::MAX_INDEX_NUME>("Nume")[15]
```

2.3.13 Clasa MeniuPrincipal

Pentru aceasta clasa se folosesc fisierele: **`meniuprincipal.h/cpp`**

Atribute private:

`CmdExit *_pExit` – pointer la comanda de iesire din program[16].

Metode publice:

13 se inlocuieste operatie cu numele functiilor aritmetice

14 se inlocuieste **`#`** cu 1 sau 2, **`nume`** cu numele functiei combinatoriale, constanta **`MAX`** este declarata in **`combinatorial.h`** cu valoarea **`12`**

15 **`Nume`** se inlocuieste cu numele claselor sirurilor numerice Fibonacci, Lucas etc, **`MAX_INDEX_NUME`** se inlocuieste cu numele constantei enumerate in clasa respectiva.

16 Se foloseste declaratie anticipata in header.

MeniuPrincipal()

```
apeleaza constructorul clasei de baza cu: Proiect Final C++
_pExit = new CmdExit("Exit");
AdaugaComanda(_pExit)
adauga comenzile pentru submeniuri:
AdaugaComanda(new Sumbeniu);
```

bool Exit() – returneaza **_pExit->Exit()**

2.4 Modulul Calculator

2.4.1 Clasa Aplicatie

Instantiaza meniul principal si executa acest meniu cat timp nu s-a executat comanda exit a meniului principal.

Pentru aceasta clasa se folosesc fisierele: **aplicatie.h/cpp**

Metode publice:

void Run():

```
meniuPrincipal
repeta
    meniuPrincipal.Execute()
cat timp meniuPrincipal.Exit() este false
```

2.5 Functia main

```
se instantiaza app de tipul Aplicatie
se apeleaza app.Run()
se retuneaza 0
```

3 DIRECTOARELE SI FISIERELE PROIECTULUI

Director	**Fisierere**
Calculator	main.cpp aplicatie.h/cpp fisierele solutiei si proiectului create de Visual C++
Calculator\Calcule	combinatorial.h/cpp exceptii.h operatii.h siruri.h/cpp
Calculator\UI	cmdaritmetice.h cmdcombinatorial.h cmdexit.h/cpp cmdsiruri.h comanda.h/cpp meniucombinatorial.h/cpp meniu.h/cpp meniuoperatiiaritmetice.h/cpp meniuprincipal.h/cpp meniusiruri.h/cpp