

**Universitatea Tehnica de Constructii din Bucuresti**

**Facultatea de Hidrotehnica**

**Specializarea: Automatica si Informatica Aplicata**

**Proiect la Programarea calculatoarelor si limbaje de Programare 2**

**Alistar Ilinca**

**CUPRINS**

INTRODUCERE

Motivarea alegerii temei

Obiectivele propuse in cadrul lucrarii

Structura lucrarii

Tehnologii utilizate

STUDIU DE CAZ

Implementarea pas cu pas

Realizarea interfetei grafice (UI)

Functia principala (onButtonClicked())

Evaluarea expresiilor cu QJSEngine

Implementarea functiilor speciale

Functii de memorie si conversie Deg/Rad

Integrarea resurselor (.ico, .rc, .qrc)

Exportul aplicatiei cu windeployqt

CONCLUZII SI CONTRIBUTII PERSONALE

Concluzii

Contributii personale

Perspective de viitor

BIBLIOGRAFIE

ANEXE

**Introducere**

**Motivarea alegerii temei**

Am ales tema acestui proiect deoarece ideea de a crea un calculator mi s-a parut atat practica, cat si interesanta. Am fost mereu curioasa cum functioneaza, de fapt, o aplicatie care stie sa faca calculi, nu doar cele simple, ci si expresii mai complexe care implica functii matematice, constante sau simboluri. Voiam sa inteleg ce se intampla in spate, cum interpreteaza codul comenzile date printr-o interfata, si cum transforma o expresie introdusa de utilizator intr-un rezultat corect.

A fost o provocare placuta dar obositoare sa imbin partea tehnica, logica si procesarea matematica cu partea creativa, mai ales cand am inceput sa personalizez aplicatia. Mi-a placut ideea de a adauga o tema roz, butoane aranjate intuitiv si chiar o iconita proprie, pentru ca am simtit ca fac ceva al meu, nu doar un proiect generic. Alegerea acestei teme m-a motivat sa invat lucruri noi despre Qt si despre cum pot transforma un concept simplu intr-o aplicatie completa si functionala.

**Obiectivele propuse in cadrul lucrarii**

Scopul principal al acestui proiect a fost realizarea unui calculator stiintific care sa poata evalua expresii matematice complexe intr-un mod cat mai intuitiv pentru utilizator. Desi nu era o cerinta obligatorie in tema, am ales sa folosesc o interfata grafica tocmai pentru a ma provoca sa ies din zona de confort si sa invat mai mult despre dezvoltarea aplicatiilor cu Qt. Am vrut sa transform un proiect tehnic intr-o experienta completa, care sa combine partea de logica matematica cu o prezentare vizuala prietenoasa.

Printre obiectivele concrete s-au numarat: includerea functiilor trigonometrice, logaritmice si exponentiale, lucrul cu constante precum π si e, dar si implementarea functiilor de memorie (MC = *Memory Clear* - sterge valoarea memorata, MR = *Memory Add* - adauga la valoarea memorata, M+ = *Memory Recall* - afiseaza valoarea memorata in campul de calcul). Am urmarit ca aplicatia sa fie adaptabila permitand comutarea intre grade si radiani – si usor de folosit chiar si de catre cineva care nu are experienta in domeniul matematicii avansate.

Un alt obiectiv important a fost personalizarea vizuala a aplicatiei: am dorit ca proiectul meu sa nu fie doar functional, ci si placut de folosit. Astfel, am creat o tema roz, am adaugat o iconita proprie si am aranjat butoanele intr-un layout coerent si ordonat. In final, am urmarit sa export aplicatia intr-o forma portabila, care sa poata rula pe orice sistem Windows fara alte instalari – ca un produs final, complet.

**Structura proiectului**

Aplicatia „Calculator Roz cu Expresii” a fost construita folosind framework-ul Qt, avand la baza o structura clasica pentru aplicatii de tip GUI (Graphical User Interface). Proiectul este organizat modular, fiecare fisier avand un rol bine definit, ceea ce a permis o dezvoltare logica si coerenta a aplicatiei.

Fisierul main.cpp reprezinta punctul de intrare al aplicatiei. Aici se initializeaza obiectul principal QApplication, se creeaza fereastra principala (MainWindow) si se seteaza titlul si iconita aplicatiei. Tot in acest fisier este apelata metoda show(), care afiseaza interfata grafica utilizatorului.

Functionalitatea principala a aplicatiei este definita in fisierele mainwindow.h si mainwindow.cpp. Acestea contin clasa MainWindow, care gestioneaza toate elementele de interfata si logica asociata fiecarui buton. In mod special, toate butoanele sunt conectate la o singura functie (onButtonClicked()), care decide actiunea potrivita in functie de textul afisat pe buton. Tot aici se implementeaza si functiile speciale ale aplicatiei, cum ar fi manipularea memoriei, conversia dintre grade si radiani, sau interpretarea simbolurilor matematice.

Layout-ul grafic al aplicatiei este definit in fisierul mainwindow.ui, creat cu ajutorul Qt Designer. Acest fisier permite pozitionarea vizuala a tuturor elementelor din interfata: butoanele pentru cifre, operatii, functii matematice si campul de afisare a expresiei.

Pentru a adauga resurse externe, cum ar fi imagini sau iconite, a fost utilizat fisierul resources.qrc. In cadrul acestuia sunt incluse toate resursele necesare aplicatiei, care sunt apoi accesate in cod printr-un prefix special (:/). Pentru ca aplicatia sa aiba o iconita personalizata inclusa direct in fisierul executabil, a fost creat si un fisier appicon.rc. Acesta este legat in fisierul .pro al proiectului si permite compilatorului sa includa imaginea .ico in .exe.

Structura aplicatiei este simpla, dar eficienta, si respecta bunele practici de separare a interfetei de logica. Aceasta organizare a permis adaugarea treptata de functionalitati, testarea independenta a componentelor si, in final, exportul complet al aplicatiei intr-un format executabil portabil.

**Tehnologii utilizate**

Pentru realizarea acestui proiect am folosit limbajul de programare C++, impreuna cu mediul de dezvoltare Qt Creator 16.0.1 (Enterprise). Am ales aceasta varianta deoarece ofera un spatiu complet de lucru pentru construirea aplicatiilor cu interfata grafica, fiind usor de invatat si de folosit.

Interfata calculatorului a fost creata cu ajutorul unei unelte vizuale numite Qt Designer, inclusa in Qt Creator. Aceasta mi-a permis sa adaug butoanele si campul de afisare intr-un mod intuitiv, fara sa scriu manual tot codul pentru aspectul aplicatiei. Am folosit un sistem de organizare pe randuri si coloane pentru ca toate elementele sa fie aliniate frumos.

Pentru partea de calcul, am folosit o functionalitate oferita de Qt care poate interpreta expresii matematice scrise ca text. Astfel, utilizatorul poate scrie de exemplu 2+3\*4, iar aplicatia va intelege si va afisa rezultatul. Acest lucru a fost posibil datorita unei componente numite QJSEngine, care interpreteaza automat expresii intr-un mod similar cu JavaScript.

Iconita aplicatiei si alte fisiere, cum ar fi imaginea pentru pictograma, au fost adaugate printr-un fisier special de tip .qrc, in care am inclus toate resursele necesare. De asemenea, pentru ca aplicatia sa aiba o iconita vizibila si atunci cand este copiata pe desktop sau in alt folder, am folosit un fisier de tip .rc, care leaga imaginea .ico de aplicatia finala.

Am folosit versiunea pentru Windows a Qt Creator, impreuna cu MinGW 64-bit, care este un program ce ajuta la „construirea” aplicatiei finale. Pentru a face aplicatia usor de deschis pe orice calculator, am folosit comanda windeployqt, care adauga automat toate fisierele de care are nevoie programul ca sa ruleze fara probleme.

Toate aceste instrumente m-au ajutat sa creez o aplicatie complet functionala, cu un aspect placut, care poate fi folosita de oricine, fara alte instalari.

**STUDIU DE CAZ – Tema aleasa**

Proiectul s-a concentrat pe dezvoltarea unei aplicatii care permite rezolvarea expresiilor matematice direct din interfata, intr-un mod intuitiv si placut vizual. In locul unei abordari simple, am integrat functii avansate, precum cele trigonometrice, lucrul cu constante si memorarea valorilor, pentru a oferi o experienta cat mai completa utilizatorului.

Accentul a fost pus nu doar pe functionalitate, ci si pe interactiunea prietenoasa cu aplicatia. Am ales sa personalizez interfata intr-un mod diferit fata de calculatoarele standard, optand pentru o tema roz, placuta si mai apropiata de stilul meu personal. Totul este construit astfel incat aplicatia sa fie usor de inteles, indiferent de nivelul de experienta al utilizatorului

**Implementarea pas cu pas**

Realizarea aplicatiei a presupus mai multe etape, fiecare cu rolul ei. Am inceput cu proiectarea interfetei grafice, folosind Qt Designer pentru a aseza toate butoanele intr-un mod clar si aerisit. Apoi am trecut la logica din spate, adica modul in care aplicatia intelege ce vrea utilizatorul sa calculeze si cum returneaza rezultatul.

Pentru interpretarea expresiilor am folosit un mecanism care le transforma intr-un format inteles de program si le calculeaza automat. In plus, am adaugat functii speciale precum conversia din grade in radiani, memorarea valorilor, functii matematice precum sin, cos, log si altele. Dupa ce toate functionalitatile au fost puse cap la cap, am trecut la partea de personalizare am adaugat o iconita, am modificat culorile si, la final, am exportat aplicatia intr-un format care sa poata fi folosit pe orice calculator.

**Realizarea interfetei grafice (UI)**

Interfata aplicatiei a fost creata folosind Qt Designer, un instrument vizual care permite asezarea elementelor direct pe ecran, fara sa fie nevoie sa scrii cod pentru fiecare buton sau camp. Am folosit un layout de tip grila (Grid Layout) pentru a organiza toate butoanele in mod clar si simetric, astfel incat aplicatia sa fie usor de folosit.

In partea de sus a ferestrei am adaugat un camp de tip QLineEdit, unde se afiseaza expresia introdusa sau rezultatul calculului. Sub acesta, am grupat toate butoanele necesare: cifrele de la 0 la 9, operatorii matematici de baza (+, –, ×, ÷), functiile precum sin, cos, log, precum si butoanele pentru paranteze, memorie, constante si functii speciale (ex: eˣ, √, |x|). Fiecare buton a fost denumit sugestiv in Qt Designer, pentru a putea fi recunoscut usor in cod.

Am acordat atentie si aspectului vizual, alegand fonturi lizibile, dimensiuni potrivite pentru butoane si am aplicat o tema in nuante de roz, care sa diferentieze aplicatia de alte calculatoare clasice. Designul a fost gandit sa fie cat mai clar, aerisit si atractiv, astfel incat utilizatorul sa nu fie coplesit de numarul mare de functii disponibile. 

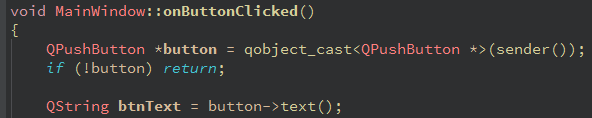
In figura de mai sus putem vedea interfata aplicatiei in Qt Designer. Aici se observa toate componentele grafice aranjate intr-un grid layout: butoane numerice, operatori matematici si functii stiintifice. Aceasta interfata a fost realizata vizual, fara cod, folosind modulul „Design” al Qt Creator.

**Functia principala (onButtonClicked())**

Toate butoanele din aplicatie sunt legate la o singura functie principala numita onButtonClicked(). Aceasta functie are rolul de a detecta ce buton a fost apasat si de a actiona corespunzator in functie de textul acestuia. Astfel, indiferent ca este vorba de un numar, o operatie matematica sau o functie speciala, totul este tratat intr-un singur loc in cod.

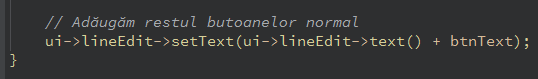
Pentru a obtine textul butonului apasat, functia identifica sursa evenimentului (butonul apasat) folosind sender() si il transforma intr-un obiect de tip QPushButton. Apoi citeste textul afisat pe buton si decide ce trebuie adaugat in campul expresiei (QLineEdit). Daca butonul este, de exemplu, „C”, expresia este stearsa; daca este „=”, atunci se face evaluarea. In cazul butoanelor speciale precum „Deg/Rad” sau cele pentru memorie, functia actualizeaza modul de lucru sau valoarea stocata.

Aceasta abordare a simplificat mult codul, in loc de a avea o functie separata pentru fiecare buton, am centralizat toata logica, ceea ce a facut aplicatia mai usor de intretinut si modificat. De asemenea, functia gestioneaza si cazurile speciale precum inchiderea modulului absolut („|x|”) sau adaugarea unui element nou intr-o expresie deja scrisa.



*Acesta este fragmentul de inceput, initializarea butonului apasat.*

La inceputul functiei onButtonClicked(), este identificat butonul care a fost apasat. Se foloseste sender() pentru a obtine obiectul care a generat evenimentul si se transforma intr-un QPushButton. Apoi se extrage textul de pe buton, care va fi folosit pentru a decide ce actiune trebuie executata mai departe (ex: adaugare caracter, evaluare expresie, functie speciala etc.).



*Fragmentul de final - adaugarea valorilor normale*

Daca butonul apasat nu este unul special (C, =, sin, memorie etc.), textul de pe acesta este adaugat direct in expresia curenta afisata in campul principal (QLineEdit). Astfel, utilizatorul poate construi pas cu pas expresii matematice complete, apasand butoanele din interfata.

**Evaluarea expresiilor cu QJSEngine**

Pentru a calcula rezultatul expresiilor introduse de utilizator, am folosit o componenta inclusa in Qt numita QJSEngine. Aceasta permite interpretarea expresiilor scrise ca text, intr-un mod asemanator cu JavaScript. Practic, orice sir de caractere care respecta regulile unui calcul matematic este evaluat automat, iar rezultatul este returnat.

Inainte ca expresia sa fie trimisa catre motorul de calcul, aceasta este prelucrata. De exemplu, simbolurile vizuale precum × si ÷ sunt inlocuite cu \* si **/**, iar constantele ca π si e sunt inlocuite cu valorile lor numerice. De asemenea, am adaugat reguli pentru functii precum sin, cos, tan, ln, log si pentru operatii speciale cum ar fi radicalul, puterea, inversul sau modulul.

Dupa ce toate aceste transformari sunt facute, expresia este trimisa la QJSEngine care o interpreteaza si returneaza rezultatul. Daca expresia este gresita sau incompleta, motorul returneaza o eroare, iar in aplicatie se afiseaza mesajul „Eroare”. In caz contrar, rezultatul este afisat in campul de text.

**Implementarea functiilor speciale**

Pe langa operatiile obisnuite, aplicatia include si o serie de functii matematice mai avansate, cum ar fi radicalul √, functiile trigonometrice (sin, cos, tan, cot), logaritmul (log, ln), exponentiala (eˣ), valoarea absoluta |x|, puterea ^ si inversul (1/x). Acestea au fost adaugate ca butoane separate in interfata si fiecare are un comportament specific tratat in functia onButtonClicked().

Pentru ca aceste functii sa poata fi evaluate corect de QJSEngine, aplicatia le transforma intr-un format pe care motorul de calcul il poate intelege. De exemplu, √(x) devine Math.sqrt(x), ln(x) devine Math.log(x), iar cot(x) este rescris ca 1/Math.tan(x). In cazul valorii absolute, expresia |x| este inlocuita cu Math.abs(x), iar puterea x^y este convertita in x\*\*y, asa cum se scrie in JavaScript.

Am avut grija sa tratez si cazurile mai speciale, cum ar fi functia eˣ, care este reprezentata intern ca e^x, sau functia de invers (1/x), care este inconjurata de paranteze pentru a evita erorile. De asemenea, am folosit expresii regulate (QRegularExpression) pentru a detecta si inlocui automat anumite sabloane din expresie, cum ar fi √25 sau cot(45). Aceste functii speciale au fost gandite astfel incat utilizatorul sa nu fie nevoit sa scrie manual formulele, ci doar sa apese butonul dorit, iar aplicatia se ocupa de restul.

**Functii de memorie si conversie Deg/Rad**

Pentru a oferi o experienta completa, aplicatia include si functii de memorie, asemanatoare cu cele intalnite pe calculatoarele stiintifice clasice. Astfel, utilizatorul are la dispozitie butoanele MC (Memory Clear), MR (Memory Recall) si M+ (Memory Add). Prin aceste functii, o valoare poate fi salvata temporar in memorie si folosita ulterior in alte calcule, fara a fi reintrodusa manual.

Implementarea acestor functii a fost simpla si eficienta. Aplicatia retine un singur numar in memorie, stocat intr-o variabila interna. Cand utilizatorul apasa M+, valoarea curenta este adunata la cea deja salvata. MR insereaza valoarea memorata in campul expresiei, iar MC reseteaza memoria la zero. Toate aceste actiuni sunt tratate in functia principala onButtonClicked() si sunt accesibile din orice moment in timpul folosirii aplicatiei.

In plus, am adaugat un buton pentru comutarea intre grade si radiani (Deg/Rad), extrem de util in cazul functiilor trigonometrice. In mod implicit, QJSEngine interpreteaza valorile in radiani. Atunci cand utilizatorul activeaza modul „Deg”, aplicatia transforma automat valorile din grade in radiani inainte de evaluare. De exemplu, sin(30) este transformat in Math.sin(30 \* Math.PI / 180). Astfel, utilizatorul poate lucra intuitiv in grade, iar rezultatul va fi corect indiferent de modul ales. Aceste functii suplimentare adauga flexibilitate aplicatiei si o fac mai apropiata de nevoile reale ale utilizatorilor, fie ca este vorba de calcule simple sau de operatii mai avansate.

**Integrarea resurselor (.ico, .rc, .qrc)**

Am vrut ca aplicatia mea sa fie personalizata si sa aiba o nota care sa ma reprezinte, nu doar din punct de vedere functional, ci si vizual. Asa ca am adaugat o iconita proprie, cu un stil placut si potrivit cu tema roz aleasa. Iconita apare atat in coltul ferestrei, cat si pe bara de activitati, iar cand aplicatia este rulata ca fisier executabil (.exe), imaginea se vede si in Windows, in locul pictogramei standard.

Pentru a adauga o iconita personalizata aplicatiei, am incercat initial sa folosesc un fisier .png prin includerea acestuia in fisierul de resurse .qrc si apelarea lui in cod cu o cale de tip ***:/icons/calculator.png***. Din pacate, aceasta varianta nu a functionat – iconita nu se afisa nici in bara de titlu, nici in alte locuri, fara sa fie un motiv evident, iar solutiile standard nu au avut efect.

Am decis sa convertesc imaginea in format .ico, care este mai potrivit pentru aplicatiile Windows. Apoi am creat un fisier appicon.rc, in care am declarat iconita, si l-am inclus atat in proiect, cat si in fisierul .pro. Dupa aceste modificari, iconita a inceput sa se afiseze corect – atat in bara aplicatiei, cat si la nivelul fisierului .exe. Chiar daca a parut un detaliu minor, aceasta modificare a facut ca aplicatia sa para mai finisata si mai profesionista.

**Exportul aplicatiei cu windeployqt**

Dupa ce aplicatia a fost complet functionala si personalizata, a urmat partea de export — adica transformarea proiectului intr-un fisier executabil care sa poata fi rulat pe orice calculator, fara a avea Qt instalat. Teoretic, acest proces ar fi trebuit sa fie simplu, dar pentru mine a fost, sincer, cea mai obositoare parte a proiectului. Am folosit comanda windeployqt, care se ocupa de copierea automata a tuturor fisierelor necesare langa .exe. Problema a fost ca desi faceam totul corect, aplicatia tot nu functiona cum trebuie. Am petrecut ore intregi incercand sa-mi dau seama ce lipseste, cautand pe forumuri, reluand pasii din documentatie, modificand iar si iar.

La final, dupa multa frustrare, am descoperit ca lucram pe un drive gresit F: in loc de C: si de aceea unele comenzi pur si simplu nu aveau efect sau nu gaseau fisierele necesare. Desi pare un detaliu mic, acel „F” mi-a dat multe batai de cap. Abia dupa ce am refacut totul in C:, exportul a functionat, iar aplicatia a rulat perfect, cu iconita, fara erori si complet portabila. Chiar daca a fost o parte complicata, am invatat multe despre cum functioneaza distribuirea unei aplicatii reale, nu doar scrierea ei. A fost o lectie de rabdare, atentie la detalii si perseverenta.

### ****CONCLUZII****

### Desi proiectul a fost, fara indoiala, fascinant, provocator si chiar amuzant pe alocuri, trebuie sa recunosc cu toata sinceritatea ca nu l-as mai repeta prea curand. A fost genul acela de proiect care iti testeaza rabdarea, logica si simtul umorului in acelasi timp. Dupa tot acest efort, simt ca merit un pic de timp pentru recuperare emotionala.

### Totusi, experienta a fost extrem de valoroasa si cred ca m-a pregatit, intr-un mod foarte realist, pentru meseria de viitor inginer. Am inteles ca, in viata reala, uneori programul nu merge ore intregi dintr-un motiv aparent banal, iar alteori, cand ceva functioneaza din prima… nu mai ai curajul sa-l atingi. Daca merge, il lasi asa. Nu-l mai atingi. Nimic. Sub nicio forma.

### Am invatat nu doar cum se scrie o aplicatie functionala, ci si cum sa o impachetezi, sa o personalizezi si mai ales cum sa nu intri in panica atunci cand terminalul refuza sa coopereze. Proiectul mi-a dat incredere, dar si un reality check: uneori lucrurile nu merg pentru ca… esti pe F: in loc de C:.

### Pe viitor, daca ar fi sa continui acest proiect, m-as concentra pe imbunatatirea designului si poate pe optimizarea codului — acum ca stiu cum functioneaza totul, as sti exact unde sa umblu si ce sa simplific. De asemenea, as putea adauga mici imbunatatiri pentru claritate si accesibilitate, dar recunosc sincer: in acest moment, sunt multumita ca merge si ca... are iconita.

### ****CONTRIBUTII PERSONALE****

### Tot proiectul a fost realizat de mine, cap - coada. Am proiectat interfata grafica, am scris codul, am adaugat functiile speciale, am testat totul si am exportat aplicatia. Nimic nu a fost preluat gata facut fiecare linie de cod, fiecare detaliu vizual si fiecare buton au fost lucrate, ajustate si testate de mine. Am incercat initial sa urmez cateva tutoriale online, dar foarte repede mi-am dat seama ca problemele mele erau mult mai specifice si mai incapatanate decat cele tratate in exemplele generice. Am apelat de cateva ori la ChatGPT pentru ajutor, dar de multe ori primeam tot aceleasi solutii generale care nu mergeau. Dupa cateva ore de incercari si mesaje in stilul „de ce nu functioneaza, ca am copiat tot bine?”, am ajuns sa ma dau cu capul de masa.

### Chiar daca au fost multe momente frustrante, am reusit in final sa rezolv problemele prin rabdare, incercari, mici pause de uitat in gol si mult debugging. Cred ca asta a fost cea mai valoroasa lectie: ca rezolvarile reale nu vin mereu dintr-un tutorial sau dintr-o sugestie AI, ci din faptul ca nu renunti pana nu vezi aplicatia functionand perfect.

### Faptul ca am dus totul pana la capat, ca am inteles ce fac si am reusit sa finalizez o aplicatie complet functionala, personalizata si portabila, e poate cea mai mare realizare pe care o iau din acest proiect.

### ****PERSPECTIVE DE VIITOR****

Daca as continua sa lucrez la aceasta aplicatie, as vrea sa adaug cateva functionalitati care mi-au lipsit in timp ce o foloseam si testam. In primul rand, un buton de Backspace ar fi extrem de util ar salva timp si nervi atunci cand gresesti o litera sau un simbol, ca sa nu fii nevoit sa stergi tot si sa o iei de la capat.

Mi-ar placea si un buton pentru schimbarea temei, astfel incat utilizatorul sa poata alege intre mai multe stiluri vizuale (nu toata lumea iubeste rozul ca mine — desi ar trebui). Aceasta functie ar face aplicatia mai flexibila si mai personalizabila.

De asemenea, daca as avea mai mult timp, as incerca sa imbunatatesc layout-ul interfetei. Acum functioneaza, dar trebuie sa recunosc ca ordinea unor butoane imi zgarie putin OCD-ul. Nu e totul asezat perfect si simetric, dar in acel moment...atat s-a putut. Cu putina reorganizare, cred ca aplicatia ar putea arata si mai bine. Chiar daca acum ma bucur ca totul functioneaza, stiu ca mereu e loc de mai bine — si cine stie, poate intr-o zi ma reapuc si o duc la versiunea 2.0, cu tot ce mi-am imaginat

**ANEXE**

Aceasta sectiune contine elemente vizuale relevante pentru aplicatia „Calculator Roz cu Expresii”. Am inclus capturi de ecran care ilustreaza interfata grafica finala, aspectul butoanelor in Qt Designer si iconita personalizata asociata aplicatiei. Aceste imagini ofera o perspectiva vizuala asupra organizarii aplicatiei, aspectului ei final si detaliilor de personalizare implementate.

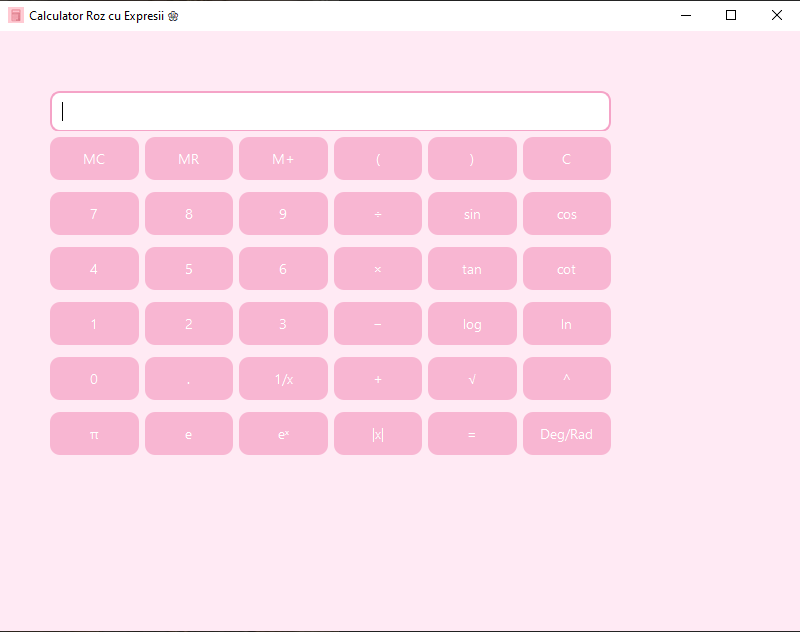


Figura de mai sus prezinta spectul final al aplicatiei „Calculator Roz cu Expresii”. Se poate observa tema personalizata in nuante de roz, interfata prietenoasa cu utilizatorul, si organizarea clara a butoanelor specifice functiilor matematice si stiintifice. Personalizarea designului a avut rolul de a face aplicatia mai placuta vizual si mai distincta.



Capturile de ecran ilustreaza modul in care apare aplicatia finala dupa rularea si exportul corect, inclusiv adaugarea iconitei personalizate. In dreapta este reprezentarea aplicatiei fixata in bara de activitati (taskbar), iar in stanga este scurtatura de pe desktop, denumita sugestiv „Calculator ROZ”. Acest nivel de personalizare a fost posibil prin adaugarea unui fisier .ico in resursele aplicatiei si legarea sa printr-un fisier .rc, asigurand astfel o identitate vizuala completa aplicatiei. Iconita roz reflecta tema generala si intentia de a oferi un design unic si prietenos.





Evaluarea expresiilor complexe si utilizarea modului Radiani

In cele doua imagini de mai sus se poate observa ca aplicatia rezolva corect expresii matematice complexe, care combina functii precum logaritm in baza 10 si functii trigonometrice. In acest caz, a fost calculata expresia 2.33 × log(9) − tan(π), iar rezultatul a fost afisat corect.

Butonul de comutare Deg/Rad a fost setat in modul Radiani, esential pentru functionarea corecta a functiei tan(π). La prima rulare a aplicatiei, daca butonul afiseaza „Deg”, inseamna ca aplicatia este deja in modul Radiani, iar apasarea butonului schimba modulul in Grade. Comportamentul este gandit intuitiv: textul de pe buton indica modulul in care vei intra dupa ce il apesi

**BIBLIOGRAFIE**

* Qt Documentation – Develop with Qt
* <https://doc.qt.io>
* Qt Widgets Calculator Example – Oficial Qt Example
* <https://doc.qt.io/qt-6/qtwidgets-widgets-calculator-example.html>
* Qt Forum – Community support and developer discussions
* <https://forum.qt.io>
* Stack Overflow – QJSEngine usage and Qt-related issues
* <https://stackoverflow.com>
* W3Schools – Math and JavaScript syntax overview
* <https://www.w3schools.com>
* GeeksforGeeks – C++ programming basics and examples
* <https://www.geeksforgeeks.org>
* Microsoft Learn – Using Windows Resource Files (.rc)
* <https://learn.microsoft.com/en-us/windows/win32/menurc/using-resources>
* YouTube – Creating a calculator using Qt (Code, Tech, and Tutorials)
* <https://www.youtube.com/watch?v=H2ud-ATLIdI>
* YouTube – Creating a calculator with Qt Creator (Sciber)
* <https://www.youtube.com/watch?v=cXojtB8vS2E>
* YouTube – C++ GUI Programming with Qt (Derek Banas)
* <https://www.youtube.com/watch?v=txGRU7OrTZo>
* ChatGPT (OpenAI) – Asistenta tehnica, corectare erori, optimizari cod si structurare documentatie
* <https://chat.openai.com>