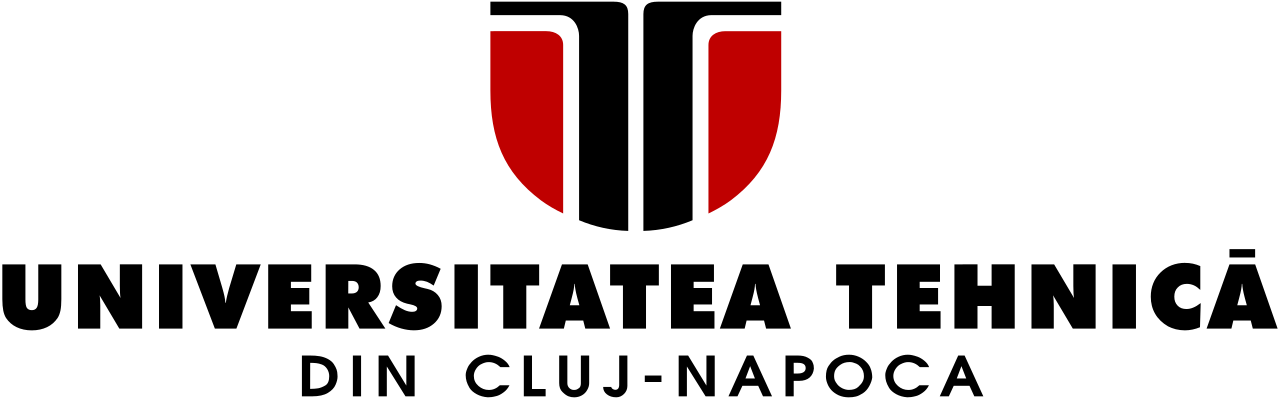
****

**Măsurarea Timpului de Execuție a Proceselor în Diferite Limbaje de Programare**

**(C, C++, Java)**

Structura Sistemelor de Calcul

Student: Lupu Daniel Ionuț

Profesor Coord. Dr. Ing. Neagu Mădălin

Cuprins

[1. Introducere 3](#_Toc124855202)

[1.1 Context 3](#_Toc124855203)

[1.2 Specificații 3](#_Toc124855204)

[1.3 Obiective 4](#_Toc124855205)

[2. Studiu bibliografic 5](#_Toc124855206)

[3. Design 8](#_Toc124855207)

[3.1. C 8](#_Toc124855208)

[3.2. C++ 9](#_Toc124855209)

[3.3. Java 10](#_Toc124855210)

[4. Implementare 12](#_Toc124855211)

[4.1 C 12](#_Toc124855212)

[4.2 C++ 13](#_Toc124855213)

[4.3 Java 14](#_Toc124855214)

[4.4 GUI 15](#_Toc124855215)

[5. Testare 16](#_Toc124855216)

[6. Concluzii 16](#_Toc124855217)

# 1. Introducere

## 1.1 Context

Scopul proiectului de la Structura Sistemelor de Calcul este de a compara în mod direct diferite limbaje de programare prin performanța lor, adică timpul de rulare măsurat pentru fiecare set de instrucțiuni care urmează să fie testat de acest program.

Acum câteva decenii, alegerea limbajului de programare potrivit era crucială pentru anumite programe mai complexe. Diferența de timp era foarte mare și, având în vedere că existau și alte opțiuni, nu era eficient să folosim limbajul care nu este potrivit pentru nevoile specifice programului si programatorului. În ziua de azi, sistemele de calcul și limbajele de programare au ajuns să fie atât de performante și eficiente încât aceste deosebiri de timp nu mai sunt la fel de considerabile, decât în cazuri care sunt de o mare amploare. Totuși, pentru cele mai bune rezultate încă este recomandat să ne informăm înainte de a incepe să lucrăm pe anumite aplicații sau programe.

Seturile de instrucțiuni testate vor fi programe care măsoara: alocarea de memorie, accesul la memorie (static si dinamic), crearea unui thread, thread context switch și thread migration. Pentru acestea se vor realiza o serie de benchmark-uri în fiecare limbaj de programare care urmează să fie analizat.

## 1.2 Specificații

Timpul de rulare va fi măsurat cu ajutorul unor comenzi care măsoara strict timpul de completare al instrucțiunilor, astfel încât comenzile suplimentare din fiecare limbaj de programare să nu reprezinte un factor în plus care trebuie măsurat. Acestea vor acoperi exact timpul concordant performanței limbajului. Toate testele vor fi desfășurate pe laptop-ul meu personal, iar acestea vor fi în număr cât mai mare, rezultând într-un timp mediu pentru a obține rezultate cât mai apropiate de adevăr.

Componente relevante pentru benchmark:

Procesor: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz (8 CPUs) ~ 1.50 GHz

Memorie RAM: 16,0 GB (15,8 GB utilizabil)

Sistem de operare: Windows 10 Pro 64-bit (10.0, Build 19044)

## 1.3 Obiective

Obiectivul principal este de a vedea cu exactitate care sunt diferențele între limbajele de programare, până la nivel de milisecunde, astfel rezultând într-o decizie cât mai bună pentru a alege limbajul de programare potrivit pentru necesitățile prezentate de fiecare utilizator. Aceștia pot câștiga timp valoros în intermediul unor programe lungi, unde timpul de rulare rezultă în diferențe considerabile.

# 2. Studiu bibliografic

Testele de performanță se vor realiza în trei limbaje de programare: C, C++ și Java. În continuare voi elabora sumar asupra acestora, prezentând când și cum s-au format, pentru ce utilizare, de către cine etc.

C este un limbaj de programare standardizat. Este implementat pe majoritatea platformelor de calcul existente azi, și este cel mai popular limbaj de programare pentru scrierea de software de sistem. Este apreciat pentru eficiența codului obiect generat de compilatoarele C, și pentru portabilitatea sa. A fost dezvoltat la începutul anilor 1970 de Ken Thompson și Dennis Ritchie, care aveau nevoie de un limbaj simplu și portabil pentru scrierea nucleului sistemului de operare UNIX. Sintaxa limbajului a stat la baza multor limbaje create ulterior și încă populare azi: C++, Java, JavaScript, C#, D. C este un limbaj de programare relativ minimalist ce operează în strânsă legătură cu hardware-ul, fiind cel mai apropiat de limbajul de asamblare față de majoritatea celorlalte limbaje de programare. Este prezentat uneori ca „asamblor portabil”, făcându-se astfel diferențele principale față de limbajele de asamblare: codul unui program poate fi compilat și rulat pe aproape orice tip de mașină (calculator), asemănător altor limbaje de programare, în timp ce limbajele de asamblare sunt specifice unui anumit model de mașină. Limbajul C aparține clasei limbajelor de nivel scăzut sau de nivel mediu, aceasta indicând strânsa legătură între interoperabilitate și echipamentul hardware. A fost creat având drept scop important de a face ca programele mari să poată fi scrise mai ușor și cu mai puține erori în paradigma programării procedurale, dar fără a pune obstacole în scrierea compilatorului de C, care este încărcat de caracteristicile complexe ale limbajului.

C++ este un limbaj de programare general, compilat. Este un limbaj multi-paradigmă, cu verificarea statică a tipului variabilelor ce suportă programare procedurală, abstractizare a datelor, programare orientată pe obiecte. În anii 1990, C++ a devenit unul dintre cele mai populare limbaje de programare comerciale, rămânând astfel până azi. Bjarne Stroustrup de la Bell Labs a dezvoltat acest limbaj (inițial denumit C cu clase) în anii 1980, ca o serie de îmbunătățiri ale limbajului C. Acestea au început cu adăugarea noțiunii de clase, apoi de funcții virtuale, suprascrierea operatorilor, moștenire multiplă (engleză multiple inheritance), șabloane (engleză template) și excepții. Limbajul de programare C++ a fost standardizat în 1998 ca și ISO 14882:1998, versiunea curentă fiind din 2003, ISO 14882:2003. În 1982, numele limbajului a fost schimbat de la C cu clase la C++. Au fost adaugate noi facilități, inclusiv funcții virtuale, supraîncărcarea operatorilor și a funcțiilor, referințe, constante, alocare dinamică, un control al tipului mai puternic și noua variantă de comentariu pe un singur rând (liniile care încep cu caracterele '//'). În 1985 a fost lansată prima ediție a cărții "The C++ Programming Language" (Limbajul de programare C++), oferind informații importante despre limbaj, care încă nu era un standard oficial. În 1989 a fost lansată versiunea 2.0 a C++. Au apărut acum moștenirea multiplă, clase abstracte, funcții statice, funcții constante și membri protected. În 1990 o altă carte a fost lansată, oferind suport pentru standarde viitoare. Ultimele adăugări includeau template-uri, excepții, spații de nume (namespace-uri) și tipul boolean. O dată cu evoluția limbajului C++, a evoluat și o bibliotecă standard. Prima adăugire a fost biblioteca de intrări/ieșiri (I/O stream), care oferea facilități pentru a înlocui funcțiile tradiționale C cum ar fi printf și scanf. Mai târziu, printre cele mai semnificative adăugari la biblioteca standard a fost STL (Standard Template Library) (Biblioteca de formate standard). După ani de lucru, un comitet ANSI-ISO a standardizat C++ în 1998 (ISO/IEC 14882:1998).

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor ʼ90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor. Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java. Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

Utilizarea variabilelor dinamice reprezintă o tehnică numită alocare dinamică. Ea se implementează prin intermediul pointer-ilor și are avantajul că folosește doar atâta memorie cât este necesară. Prin alocare statică înţelegem faptul că memoria este alocată automat încă de la pornirea execuţiei instrucţiunilor unei funcţii, iar eliberarea memoriei se face tot automat la părăsirea funcţiei.

Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces.

Thread switching este o tehnică care comută de la un thread la altul în cadrul aceluiași proces. Acesta este foarte eficient și are un cost mult mai mic de implementare pentru că schimbă doar unele identități și resurse, precum contorul programelor, regiștrii și pointerii de stack.

Thread migration asigură trecerea unui thread de la o sarcină la alta, pentru a asigura o funcționare corectă.

# 3. Design

## 3.1. C

Pentru partea de măsurare a performanței în limbajul de programare C vom folosi anumite funcții din librăria time.h. Această librărie este un header file care se află deja în librăriile standard din C. Conține funcții cu declarări legate de timp și dată pentru a ajuta la manipularea unităților de timp. Noi vom folosi funcția clock() care ne returnează numărul de clock-uri care s-au procesat până în momentul prezent. Nu acceptă parametrii, iar tipul returnat este de clock\_t. Vom apela această funcție fix înainte de a ajunge la codul pe care vrem să îl testăm și o vom reapela fix după ce această porțiune de cod se termină. Pentru a afla câte clock-uri a durat procesul nostru vom face diferența dintre cele două rezultate pe care le-am obținut. Valoarea la care am ajuns acum este în clock-uri, dar pentru a obține această valoare în secunde, vom împărți rezultatul la CLOCKS\_PER\_SEC, astfel rezultând timpul final.



Cod preluat de pe:

<https://stackoverflow.com/questions/1688133/how-can-i-test-the-performance-of-a-c-function>

## 3.2. C++

La fel ca la măsurarea performanței în C, vom realiza o procedură similară și în C++. Ne vom folosi de librăria chrono, care are ca funcționalitate manipularea elementelor de timp. Toate elementele din acest header nu sunt definite direct în std namespace, dar se pot apela cu std::chrono. Folosim funcția high\_resolution\_clock, cu parametrul now pentru a obține momentul curent. În aceeași manieră ca la programul din C, vom apela funcția fix înainte și fix după apelul funcției pe care vrem să o testăm. La final, ca să obținem timpul care a durat pentru funcția respectivă, scădem valoarea de la primul apel al funcției din valoarea obținută la al doilea apel. Rezultatul returnat este măsurat în milisecunde, iar dacă vrem să trecem în secunde înmulțim cu 103.





Cod preluat de pe: <https://godbolt.org/z/oe5cMd>

## 3.3. Java

Pentru partea de Java, putem folosi instant.now() ca funcție pentru a obține timpul curent. Astfel, în mod complet analog vom trece prin același proces ca la C și C++. Apelăm instant.now(), apelăm funcția pe care dorim să o testăm și apelăm iarăși instant.now(). În final, facem diferența între ultimul apel și primul, iar acesta este rezultatul. De asemenea, mai sunt și alte moduri de a obține performanța unei funcții în java. Funcțiile System.nanoTime(), System.currentTimeMillis() se pot folosi exact ca instant.now() și obtinem rezultatele în diferite unități de timp, în funcție de cum preferăm. Încă o metodă este folosirea instanței StopWatch, cu funcțiile start(), stop(), getTime(). În acest fel nu mai suntem nevoiți să facem diferența dintre două valori, deoarece funcția face direct acest lucru.





Cod preluat de pe:

<https://www.tutorialspoint.com/how-to-measure-execution-time-for-a-java-method>

# 4. Implementare

## 4.1 C

Programele din C au fost scrise si rulate in CodeBlocks, cu compilatorul MinGW. Toate operatiile care trebuiau masurate au fost scrise in proiecte separate, pentru a putea extrage executabilul. In cod s-a masurat timpul de executie cu functia clock() din libraria time.h, iar acesta a fost exportat intr-un fisier de tip text cu un FILE pointer si cu fopen in modul write. Motivul pentru care am folosit fisiere text este pentru a putea folosi valorile in interfata din Java, astfel incat acestea sa fie afisate la apasarea unui buton. Rezultatele au fost salvate in variabile de tip double, pentru a avea precizie cat mai mare la benchmark.







## 4.2 C++

Rularea codurilor pentru C++ a fost realizata tot in CodeBlocks si cu compilatorul MinGW. Pentru inregistrarea timpilor de executie a fost folosita libraria chrono.h. Rezultatele au fost salvate la fel ca in C, in fisiere, cu variabile de tip double, pentru a putea transfera informatia catre interfata GUI din Java.





## 4.3 Java

Pentru partea de Java, codul a fost scris si rulat in IntelliJ IDE. Aici nu am mai folosit scrierea rezultatelor in fisiere, deoarece le putem trimite direct la interfata prin settere-le de la casutele de text care afiseaza rezultatul. A fost folosita functia System.nanoTime() pentru a obtine timpul de executie in nanosecunde si au fost adaugate operatii pentru a transforma rezultatul in secunde. A fost utilizata si o variabila de tip DecimalFormat pentru a putea formata rezultatul obtinut in urma executiei intr-o forma de double cu precizie de 12 zecimale.



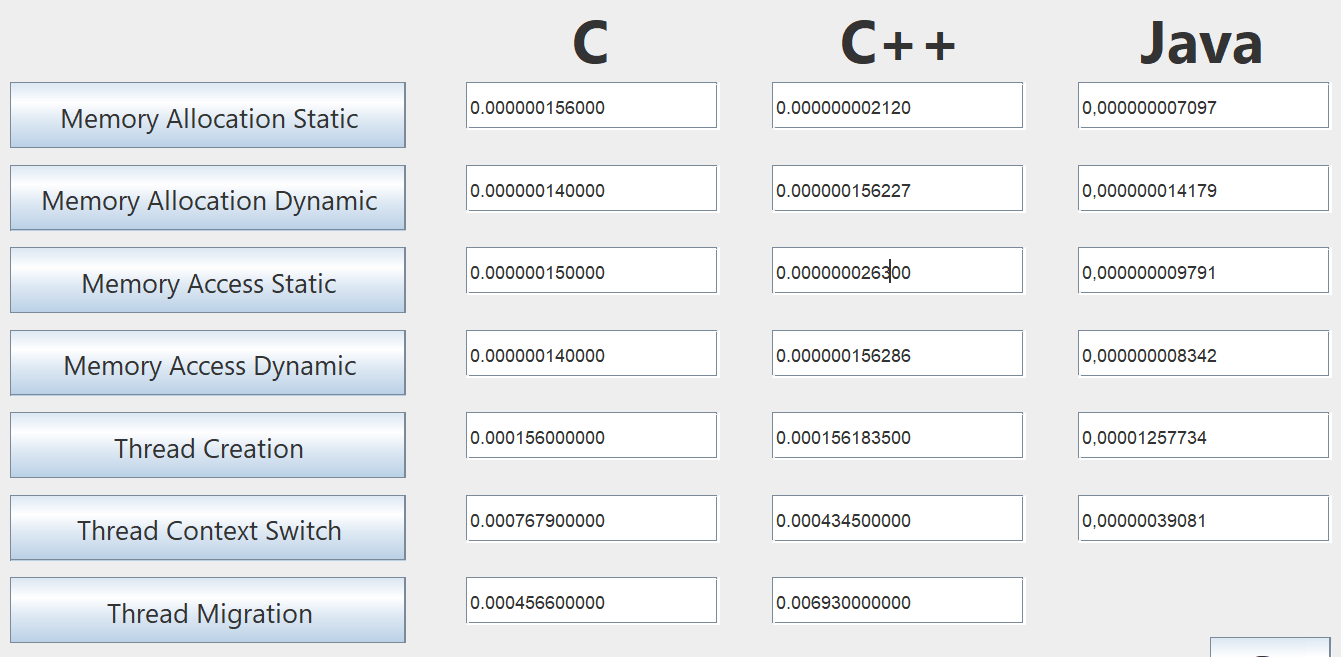
## 4.4 GUI

Designul interfetei grafice de utilizator a fost facut in Eclipse IDE cu absolute layout, urmand sa fie mutat intr-o clasa din proiectul Java din IntelliJ. Pentru a putea obtine benchmark-ul, intai apelam o functie care ruleaza fisierul executabil C/C++ care ne va genera fisierul text cu rezultatul aferent functiei, pornim un delay de o secunda, iar apoi citim numarul din fisier si il afisam in interfata. De asemenea, tratam si cazurile de eroare cu exceptii de tipul Exception.



# 5. Testare

In urma testarii, se pot vedea diferente intre timpurile de executare ale celor trei limbaje.



Aceste rezultate sunt in secunde. Se poate observa ca valorile obtinute in Java sunt cele mai mici, apoi cele din C++, iar cele din C sunt cele mai lente. De asemenea, mentionez ca dupa mai multe rulari ale programelor din Java, acestea isi imbunatatesc timpul de executie considerabil, fapt datorat refolosirii adreselor care au fost deja alocate inainte. Un alt caz interesant este obtinerea de timp de executie care tinde foarte tare spre 0 la alocarile statice din C si C++.

# 6. Concluzii

Concluzia finala a proiectului este ca, pe operatiile de alocare, accesare si threading, Java este pe primul loc ca performanta, urmat de C++, iar apoi de C. Cu toate acestea, fiecare limbaj are avantajele si dezavantajele lui. In ziua de azi unii factori mai importanti decat performanta sunt functionalitatile, librariile si plugin-urile, iar aceste limbaje dispun de foarte multe astfel de optiuni. Asadar, fiecare programator trebuie sa incerce si sa experimenteze cat mai multe unelte pe care le are la indemana, pentru a gasi ce este cel mai potrivit pentru munca lui.