FA Laborator

Raluca Laura Portase, Eugen-Richard Ardelean

September 28, 2025

Contents

1 Ghid de laborator **6**
1.1 Despre laborator . . . 6
1.2 Formatul laboratorului . . . 6
1.2.1 Reguli . . . 6
1.2.2 Notare . . . 6
1.2.3 Predare teme . . . 7
1.2.4 Evaluarea complexității algoritmilor . . . 7
1.2.5 Termene de predare . . . 8
1.2.6 Tentativa de fraudare . . . 8
1.3 Transferul între cadre didactice . . . 8
1.4 Bibliografie sugerată . . . 9
1.5 **Profiler** . . . 9

2 Sesiune introductiva **9**
2.1 Microsoft Visual Studio . . . 9
2.1.1 Instalare . . . 9
2.1.2 Utilizare . . . 10
2.2 JetBrains CLion . . . 13
2.2.1 Instalare . . . 13
2.2.2 Utilizare . . . 13
2.3 C/C++ . . . 15
2.3.1 Citire/scriere fișiere . . . 15
2.3.2 Generare cazuri de testare . . . 15
2.3.3 Generare grafice . . . 16
2.3.4 Microsoft Office Excel . . . 16
2.3.5 Exercițiu . . . 19

3 Tema nr. 1: Analiza și Compararea Metodelor Directe de Sortare **19**
3.1 Implementare . . . 19
3.2 Cerințe minimale pentru notare . . . 20
3.3 Cerințe . . . 20
3.3.1 Implementarea metodelor de sortare (5.5p) . . . 20

3.3.2 Analiză algoritmilor pentru caz mediu statistic (1.5p – 0.5 per algoritm) . . . 20
3.3.3 Analiză in caz favorabil și defavorabil (3p - câte 0.5p pentru fiecare caz a fiecărui algoritm) . . . 21
3.3.4 Bonus: Insertion sort prin inserție binară (0.5p) . . . 21

4 Tema Nr. 2: Analiza și Compararea a două metode de construire a structurii de date Heap: "De jos în sus" (Bottom-up) vs. "De sus în jos" (Top-down) 21
4.1 Implementare . . . 21
4.2 Cerințe minimale pentru notare . . . 21
4.3 Cerințe . . . 22
4.3.1 Analiza comparativă a *unuia* din algoritmii de sortare din L1 (la alegere) în versiune *iterativă* vs *recursivă*. Analiza se va efectua atât din *perspectiva numărului de operații, cât și a timpului de rulare*(2p) . . . 22
4.3.2 Implementarea metodei bottom-up de construire a unui heap (2p) . . . 22
4.3.3 Implementarea metodei top-down de construire a unui heap (2p) . . . 23
4.3.4 Analiza comparativă a celor două metode de construire în cazul mediu statistic (2p) . . . 23
4.3.5 Analiza comparativă a metodelor de construcție în cazul defavorabil (1p) . . . 23
4.3.6 Implementarea și exemplificarea corectitudinii algoritmului heapsort (1p) . . . 23

5 Tema Nr. 3: Analiza și compararea metodelor avansate de sortare – HeapSort și QuickSort / QuickSelect 23
5.1 Implementare . . . 24
5.2 Cerințe minimale pentru notare . . . 24
5.3 Cerințe . . . 24
5.3.1 QuickSort: implementare (2p) . . . 24
5.3.2 QuickSort: evaluare în caz mediu statistic, favorabil si defavorabil (3p) . . . 24
5.3.3 QuickSort și HeapSort: analiză comparativă a cazului mediu statistic (2p) . . . 25
5.3.4 Implementarea hibridizării quicksort-ului (1p) . . . 25
5.3.5 Determinare a unui prag optim în hibridizare + motivație (grafice/măsuratori) (1p) . . . 25
5.3.6 Analiză comparativă (între *quicksort și quicksort hibridizat*) din punct de vedere a numărului de operații și a timpului efectiv de execuție (1p) . . . 25
5.3.7 Bonus: QuickSelect - Randomized-Select (0.5p) . . . 26

6 Tema Nr. 4: Interclasarea eficientă a k șiruri ordonate **26**
6.1 Implementare 26
6.2 Cerințe minimale pentru notare 26
6.3 Cerințe 27
6.3.1 Demo pentru generarea a k liste aleatoare sortate de dimensiuni diferite (având în total n elemente, unde n și k sunt date) și interclasarea a 2 liste (5p) 27
6.3.2 Adaptare operațiilor de *min-heap* pe structura nouă și interclasarea a k liste (3p) 27
6.3.3 Evaluarea algoritmului în cazul mediu statistic (2p) . . . 27

7 Tema Nr. 5: Căutarea în tabele de dispersie **28**
7.1 Implementare 28
7.1.1 Hashing (se referă la tabela de dispersie (hash table)) . . 28
7.2 Cerințe minimale pentru notare 29
7.3 Cerințe 29
7.3.1 Implementarea operației de inserare și căutare folosind structura de date cerută + *demo (dimensiune 10)* (5p) . . 29
7.3.2 Evaluarea operației de căutare pentru un singur factor de umplere 95% (2p) 29
7.3.3 Completarea evaluării pentru toți factorii de umplere (2p) 30
7.3.4 Implementare operației de ștergere într-o tabelă de dispersie, *demo (dimensiune 10)* și evaluarea operației de căutare după ștergerea unor elemente (1p) 30

8 Tema Nr. 6: Arbori Multicăi **31**
8.1 Implementare 31
8.2 Cerințe 31
8.2.1 Implementare a parcurgerii iterative și recursive a unui arbore binar în *O(n)* și cu *memorie aditională constantă* (3p) 31
8.2.2 Implementarea transformărilor între diferite reprezentări . 32
8.2.3 Implementarea corectă la pretty-print la *R1* (2p) 32
8.2.4 Implementarea corectă la *T1 (din R1 în R2)* și pretty-print la *R2* (1p) + *T1* în timp liniar (1p) 32
8.2.5 Implementarea corectă la *T2* (din R2 în R3) și pretty-print la *R3* (2p) + *T2* în timp liniar (1p) 32

9 Tema Nr. 7: Statistici dinamice de ordine **32**
9.1 Implementare 32
9.2 Cerințe 33
9.2.1 BUILD_TREE: implementare corectă și eficientă (5p) . . 33
9.2.2 OS_SELECT: implementare corectă și eficientă (1p) . . . 33
9.2.3 OS_DELETE: implementare corectă și eficientă (2p) . . . 33
9.2.4 Evaluarea operațiilor de management - BUILD, SELECT, DELETE (2p) 34

9.2.5 Bonus: Implementarea utilizând AVL / arbori roșu și negru (1p) . . . 34

10 Tema Nr. 8: Mulțimi disjuncte 34
10.1 Implementare . . . 34
10.2 Cerințe . . . 35
10.2.1 Implementare corectă a MAKE_SET, UNION și FIND_SET (5p) . . . 35
10.2.2 Implementarea corectă și eficientă a algoritmului lui Kruskal (2p) . . . 35
10.2.3 Evaluarea operațiilor pe mulțimi disjuncte (MAKE, UNION, FIND) folosind algoritmului lui Kruskal (3p) . . . 35

11 Tema Nr. 9: Căutarea în lățime (BFS) 36
11.1 Introducere . . . 36
11.2 Structura proiectului . . . 36
11.2.1 Inițializarea proiectului pe Windows, cu Visual Studio . . 36
11.2.2 Inițializarea proiectului pe Linux și Mac . . . 36
11.3 Funcționare . . . 37
11.3.1 Exemplu: comanda `neighb` . . . 38
11.3.2 Exemplu: comenzile `bfs` și `bfs_step` . . . 39
11.3.3 Exemplu: comanda `bfs_tree` . . . 39
11.3.4 Exemplu: comanda `path` . . . 39
11.3.5 Structuri de date folosite . . . 40
11.4 Cerințe . . . 41
11.4.1 Determinarea vecinilor unei celule (2p) . . . 41
11.4.2 Implementarea algoritmului BFS (3p) . . . 41
11.4.3 Afișarea arborelui BFS (2p) . . . 42
11.4.4 Evaluarea performanței algoritmului BFS (3p) . . . 42
11.4.5 Bonus: Determinarea celui mai scurt drum (0.5p) . . . 42
11.4.6 Bonus: Unde poate ajunge un cal pe tablă? (0.5p) . . . 42

12 Tema Nr. 9: Tutorial 43
12.1 Configurare Proiect Visual Studio . . . 43
12.2 Eroare 'unsafe' în Visual Studio . . . 46
12.3 Eroare 'Assertion' în Visual Studio . . . 51
12.4 Eroare undefined în CLion . . . 52
12.5 CLion Visual Studio – Opțiunea 1 (mai lentă) . . . 53
12.6 CLion MinGW – Opțiunea 2 (mai rapidă, necesită consolă externă) 56
12.6.1 Eroare Clear în CLion . . . 56
12.6.2 CLion nu afișează grila . . . 58
12.7 Comanda de rulare pe Mac . . . 59

13 Tema Nr. 10: Căutare în adâncime (DFS) 59
13.1 Implementare 59
13.2 Cerințe 60
13.2.1 DFS (5p) 60
13.2.2 Sortare topologică (1p) 60
13.2.3 Tarjan (2p) 60
13.2.4 Analiza performanței pentru DFS (2p) 60

14 Rezolvarea erorilor 60
14.1 Profiler + VS Code error 60

References 63

1 Ghid de laborator

1.1 Despre laborator

În acest semestru, urmează să implementați o serie de algoritmi și să analizați corectitudinea și eficiența propriilor implementări. Implementările vor avea ca punct de pornire pseudo-codul de la curs și seminar. Le puteți scrie în C sau C++ sau un alt limbaj de programare pe care îl cunoașteți atâta timp cât implementați toate structurile de date de care aveți nevoie.

1.2 Formatul laboratorului

1.2.1 Reguli

- Prezența este obligatorie
- O absență poate fi recuperată (tema corespunzătoare poate fi predată în următoarea săptămână)
- Absența a 2-a poate fi recuperata în laboratorul special de la sfârșitul semestrului (contra cost) rezolvând teme suplimentare (NU se rezolva tema corespunzatoare sesiunii in care s-a inregistrat absenta)
- Dacă ai mai mult de 2 absențe **nu poţi participa la examen în sesiunea normală**
- **IN MOD EXCEPTIONAL**, poți participa la laborator în aceeași săptămână cu o altă grupă dacă mă anunți din timp pe email, si primesti acordul meu (a cadrului didactic de laborator de la care lipsesti); tema se prezinta tot cadrului didactic la sesiunile căruia ești înscris (tema se incarcă la timp, se prezintă data urmatoare când participi cu semigrupa ta)

1.2.2 Notare

- Nota de la laborator valorează **30%** din nota finală
- Nota de la laborator este compusă din două părți: Nota pe teme - 2/3 din nota de laborator (altfel spus **20%** din nota finală) și Colocviu - 1/3 din nota de laborator (altfel spus **10%** din nota finală)
- Teme
  - Fiecare temă are o pondere egală la calculul notei finale pe teme - media aritmetică
  - Fiecare temă are mai multe praguri de notare accesibile în documentul de cerințe
- Colocviu

  - Colocviul constă într-un test de laborator de tip closed book în ultima săptămână a semestrului (W14)
  - Colocviul va fi susținut de fiecare student pe calculatoarele disponibile în sala de laborator (nu pe laptopuri / calculatoare personale)

- Pentru a promova laboratorul și a fi acceptați în examen, trebuie să îndepliniți **ambele cerințe**:
  - Nota teme ≥ 5
  - Nota colocviu $\geq$ 5

1.2.3 Predare teme

- La discuția de evaluare se vor prezenta: **codul sursă, graficele** și un **exemplu de rulare**
- **Codul sursă** (ex: program.cpp) si **graficele** trebuie încărcate pe Moodle, într-o arhivă (ex: program.zip), **înainte de sesiunea de laborator**
- **Nu evaluăm** teme cu cod neindentat
- **Nu evaluăm** teme pentru care studentul nu poate explica algoritmul (algoritmii) utilizați
- Fiecare fișier sursă trebuie să conțină la început un comentariu cu următorul format:

```
/**
* @author Ionescu Popescu
* @group 30221
*
* Specificațiile problemei, ex: Comparați metodele de sortare X, Y
*
* Interpretarea personală despre complexitate (timp și spațiu), despre cazurile de testare (favorabil,
* mediu-statistic si nefavorabil) ex: Metoda X are complexitatea Y in cazul Z pentru ca ..
*/
```

1.2.4 Evaluarea complexității algoritmilor

- Pentru cazul mediu-statistic, repetați măsurătorile de cel puțin 5 ori
- Măsurați numărul de operații efectuate de algoritm (atribuiri și comparații pe datele de intrare sau pe variabile auxiliare ce conțin date de intrare)
- Variați dimensiunea datelor de intrare în concordanță cu specificația fiecărei teme

- Aplicați aceleași date de intrare pe fiecare algoritm în cazul evaluărilor comparative (cazul mediu statistic)
- Generați grafice pentru evaluare (fie în **Excel** sau folosind **Profiler**-ul)
- Analizați graficele și adaugați observațiile personale în secțiunea de început

1.2.5 Termene de predare

Temele pot fi predate:

- În cadrul laboratorului în care sunt discutate. La finalul laboratorului trebuie sa încărcați pe **moodle** o versiune a temei curente (cu cat ați apucat să lucrați la ea). Lipsa unei submisii la finalul orei (sau a unei submisii cu prea puțin cod relevant) se penalizează cu pana la 2 puncte din nota pe acea tema.
- **Extensia_1** (E1): la începutul următorului laborator
- **Extensia_2** (E2): anumite teme se pot preda la începutul celui de-al doilea laborator de după cel în care a fost prezentata tema (cu o **penalizare** de -2)
- Din cel de-al treilea laborator, tema nu mai poate fi predata (valorează 0)

Găsiți o planificare a temelor și a extensiilor corespunzătoare pe Moodle.

Temele trebuie încărcate pe Moodle înainte de începutul laboratorului în care sunt predate.

1.2.6 Tentativa de fraudare

Pentru prima tentativă de fraudare descoperită (copiatul codului altei persoane sau folosirea de cod generat prin unelte AI), tema respectivă se punctează cu 0 puncte. ***O tentativă ulterioară de fraudare duce la recontractarea materiei anul următor***.

1.3 Transferul între cadre didactice

Dacă doriți să participați la orele de laborator cu un alt cadru didactic trebuie să respectați următoarea regulă:

- Studentul S1 din grupa G1 poate să se mute in grupa G2 doar dacă găsește un student S2 din grupa G2 ce e dispus să participe la laborator cu grupa G1.

Pentru a formaliza "transferul" trebuie să trimiteți un email în care să precizați cu cine faceți transferul:

- Un email de la S1 către cele două cadre didactice implicate
- Un email de la S2 către cele două cadre didactice implicate

Termenul limita pentru "transfer" e sfârșitul **săptămânii 2** de scoala.

1.4 Bibliografie sugerată

- Cormen, T. H. et al (2009). Introduction to algorithms. MIT press
- J. Kleinberg, E. Tardos (2005). Algorithm Design. Addison Wesley
- Tutoriale C/C++
  - http://www.cprogramming.com/begin.html
  - http://www.learn-c.org
  - Accelerated C++: Practical Programming by Example
- Ghiduri de stil
  - http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html
  - http://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html
  - http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

1.5 Profiler

Biblioteca care se va utiliza pentru generarea graficelor, fiecare student va trebui sa parcurgă exemplul și tutorialul dat.

Cea mai recentă versiune se găseste aici:
https://github.com/cypryoprisa/utcn-fa-profiler

Profiler Tutorial:
- Part 1
- Part 2
- Part 3

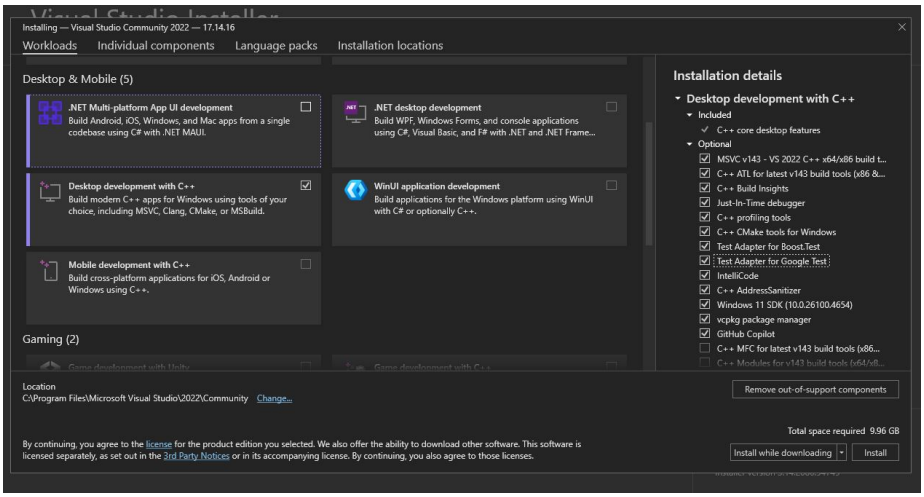
2 Sesiune introductiva

Pentru început asigură-te că ai citit Ghidul de laborator de pe Moodle. În acest laborator vei învăța cum să scrii un program C/C++ în Microsoft Visual Studio sau JetBrains CLion. De asemenea vei învăța cum să-ți generezi datele pentru evaluarea algoritmilor și cum să generezi grafice în Microsoft Office Excel.

2.1 Microsoft Visual Studio

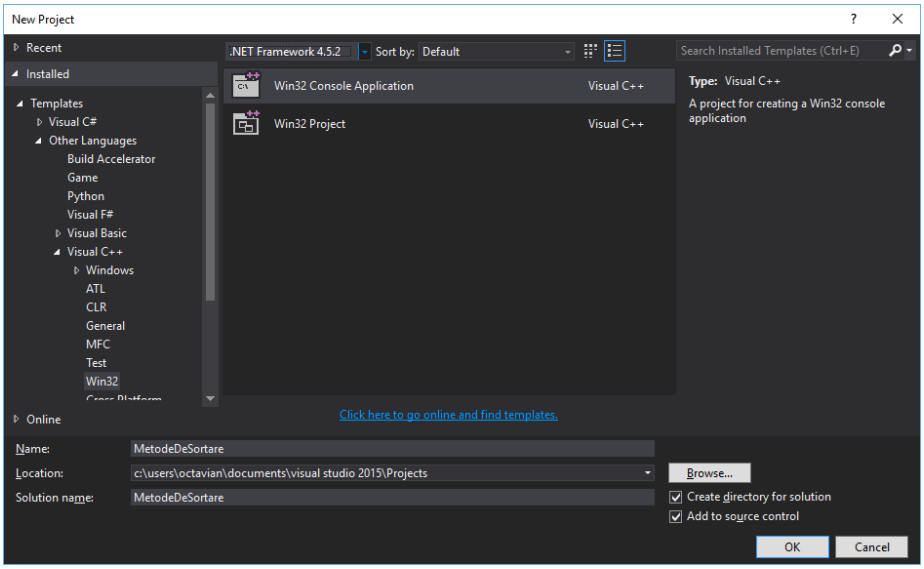
2.1.1 Instalare

Ediție gratuită: `https://visualstudio.microsoft.com/vs/community/`
În timpul instalării, se vor alege următoarele pachete:

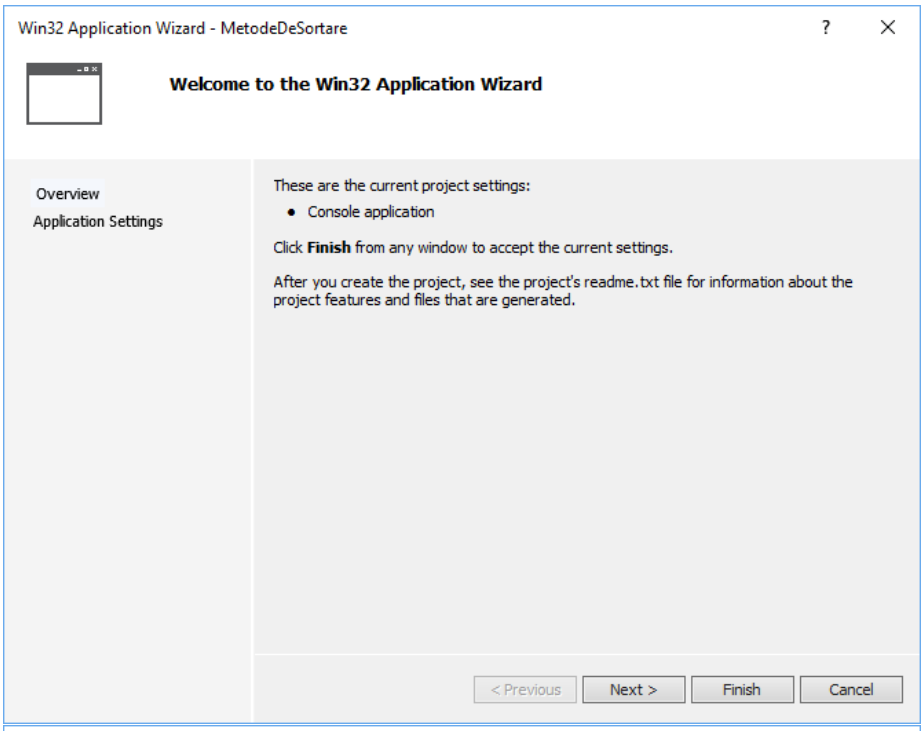


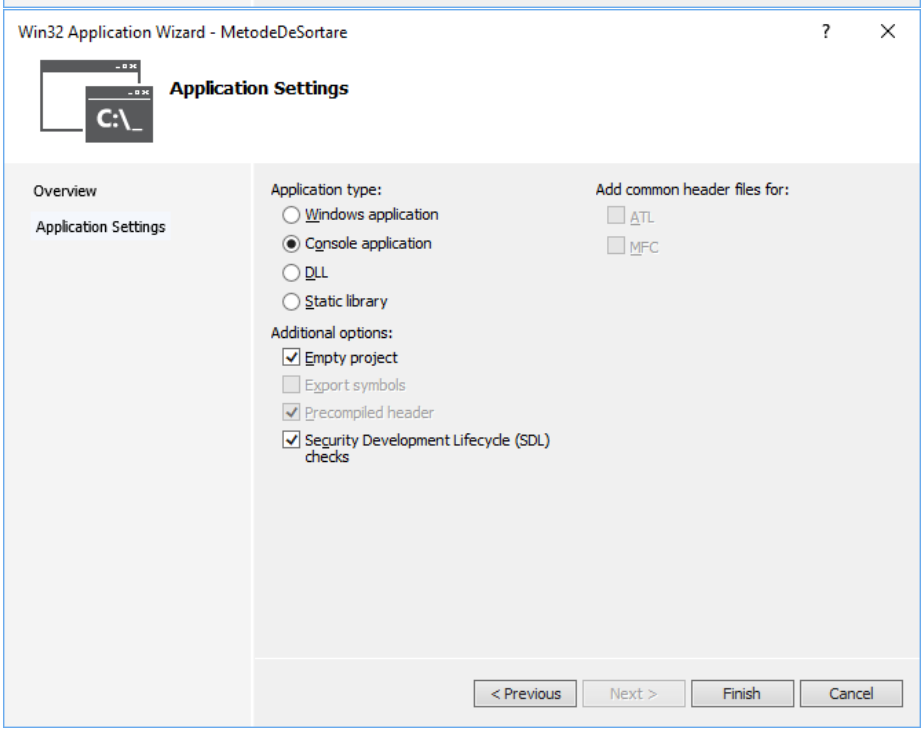
2.1.2 Utilizare

1. Creare proiect: *File -> New -> Project...*
2. Tip proiect: *Templates -> Other Languages -> Visual C++ -> Win32 -> Win32 Console App*

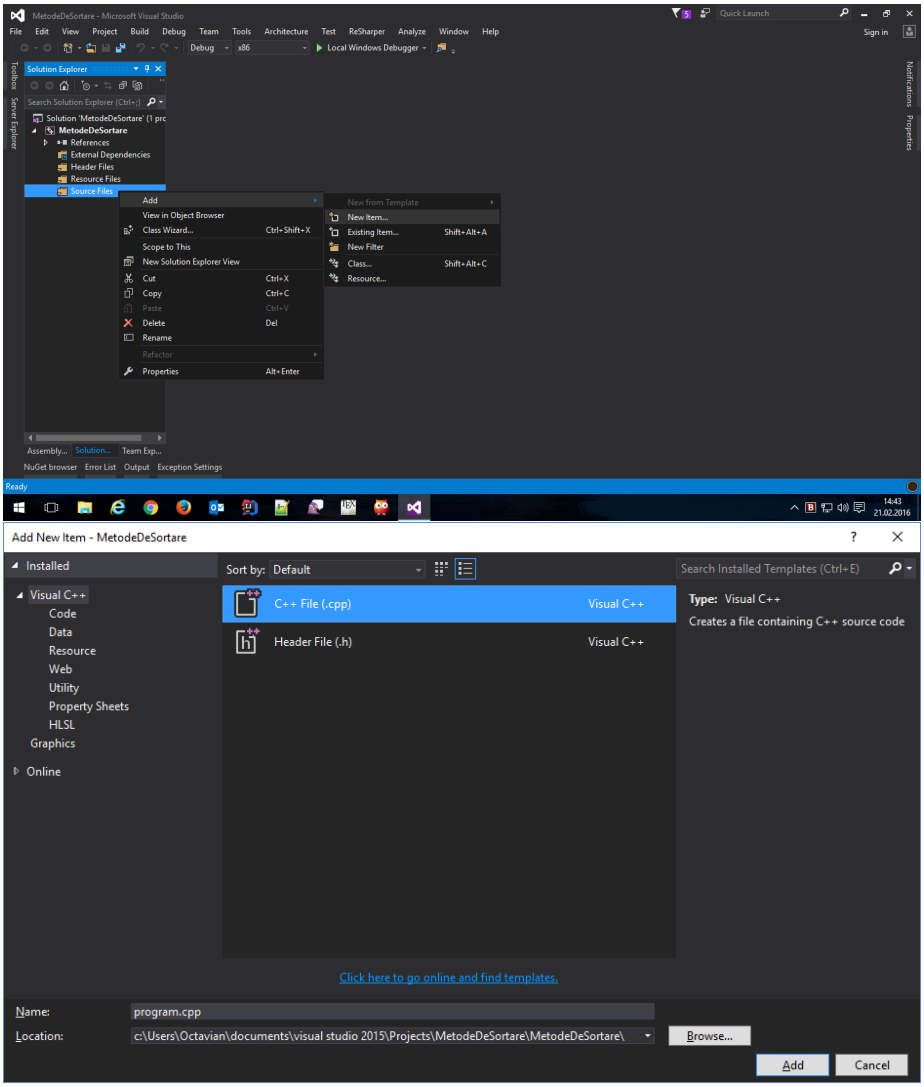


3. Proprietăți proiect: alege "*Console application*" și bifează "*Empty project*"

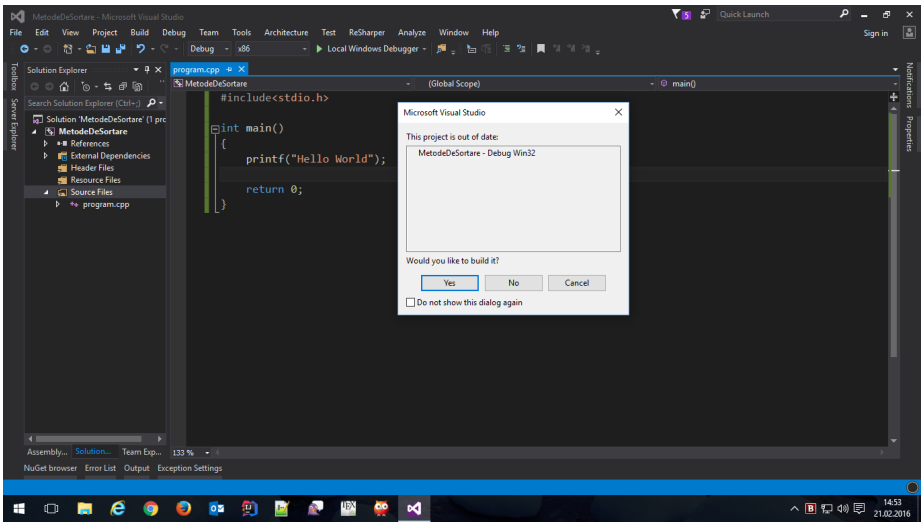

Win32 Application Wizard - MetodeDeSortare
Welcome to the Win32 Application Wizard
Overview
Application Settings
These are the current project settings:
Console application
Click Finish from any window to accept the current settings.
After you create the project, see the project's readme.txt file for information about the project features and files that are generated.
< Previous
Next >
Finish
Cancel


Win32 Application Wizard - MetodeDeSortare
Application Settings
Overview
Application Settings
Application type:
Windows application
Console application
DLL
Static library
Additional options:
Empty project
Export symbols
Precompiled header
Security Development Lifecycle (SDL) checks
Add common header files for:
ATL
MFC
< Previous
Next >
Finish
Cancel

4. Creează un fișier **.cpp*



5. Compilare și execuție program (în mod **DEBUG**)

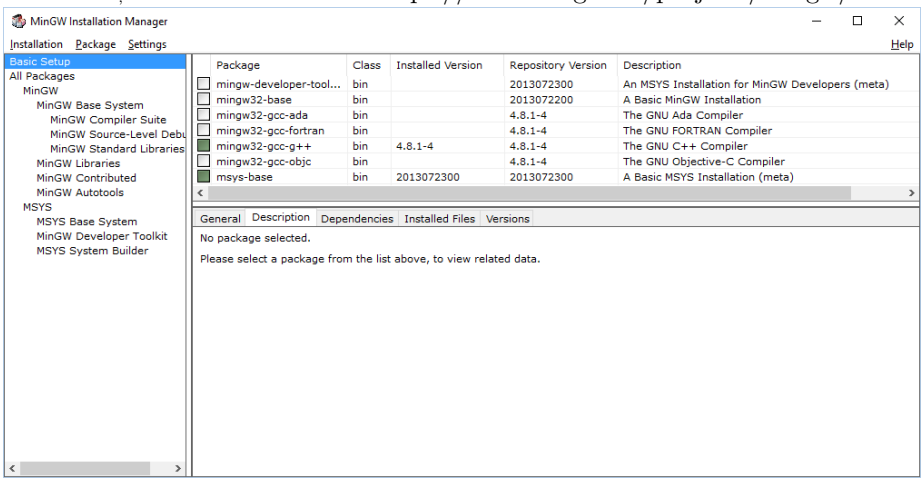


2.2 JetBrains CLion

2.2.1 Instalare

Înregistrează-te cu adresa *@student.utcluj.ro* pe https://www.jetbrains.com/student/

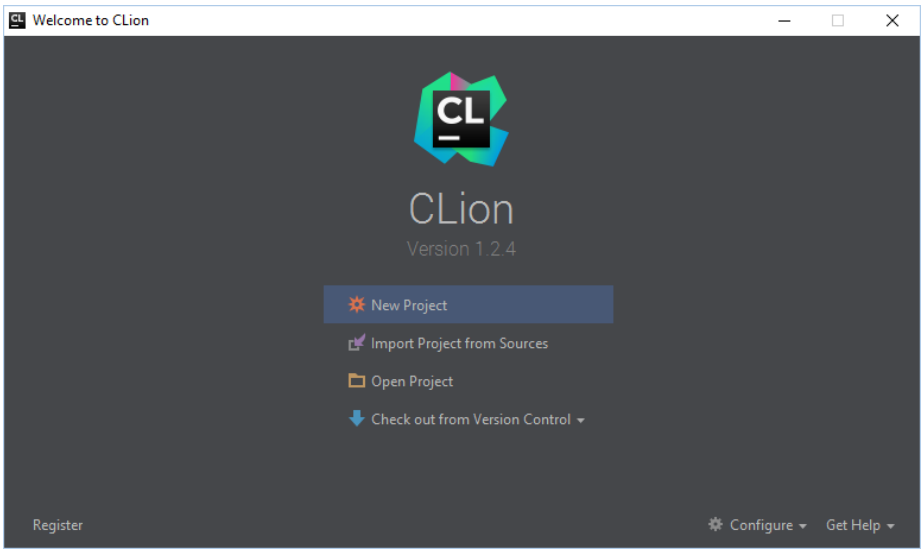
Descarcă și instalează MinGW: https://sourceforge.net/projects/mingw/



Descarcă și instalează CLion: `https://www.jetbrains.com/clion/download/#section=windows`

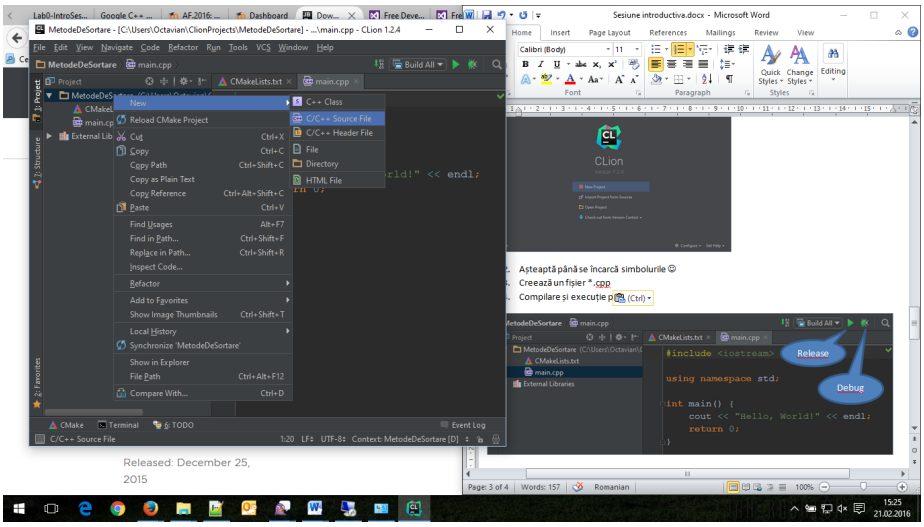
2.2.2 Utilizare

1. Creare proiect: *New Project*

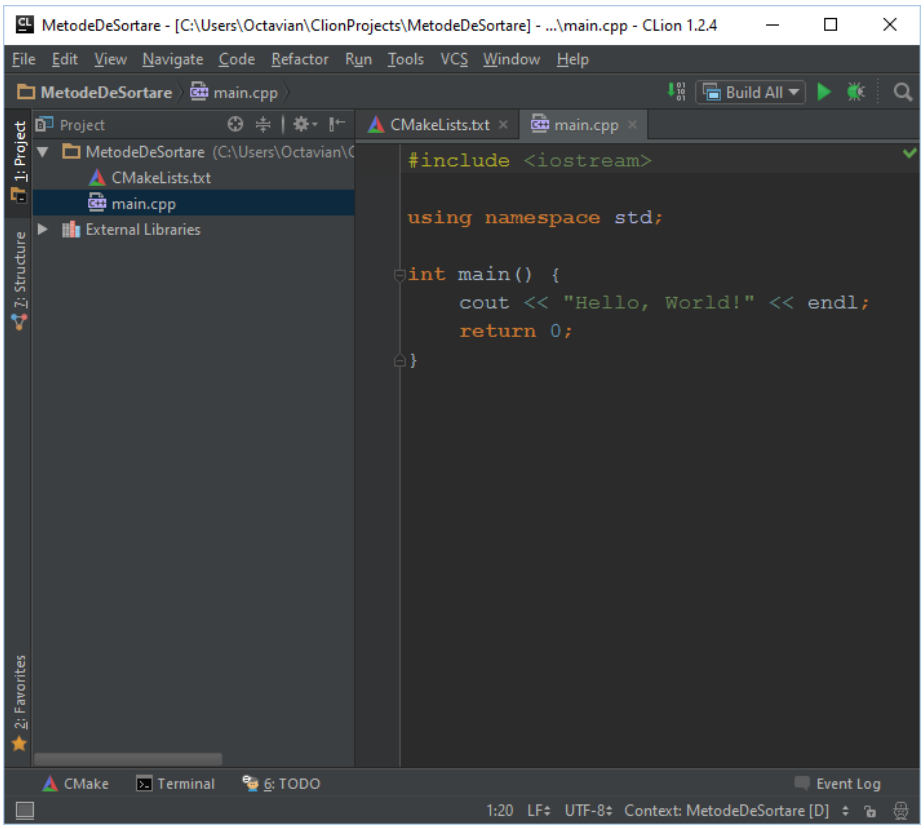


2. Așteaptă până se încarcă simbolurile

3. Creează un fișier **.cpp*



4. Compilare și execuție program



Debug
Release

2.3 C/C++

2.3.1 Citire/scriere fișiere

Exercițiu - pași:

- Declară un sir v de lungime *MAX_SIZE* (o constantă definită de tine)
- Citește n de la tastatură
- Deschide fișierul *input.txt*, citește n numere din el și salvează-le în v
- Salvează cele n numere în fișierul *output.txt* în ordine **inversă**

2.3.2 Generare cazuri de testare

Pentru a testa algoritmii care o să-i implementezi, va trebui să folosești o serie de date de intrare: șiruri ordonate crescător, șiruri ordonate descrescător, șiruri aleatoare etc. Generarea șirurilor crescătoare/descrescătoare ar trebui să fie simplă. Pentru generarea șirurilor aleatoare poți folosi următoarele:

- Biblioteca *Profiler* de pe Moodle (sau `https://github.com/cypryoprisa/utcn-fa-profiler`)
- metodele *rand*(), *srand*(), citește:
  - `http://www.cplusplus.com/reference/cstdlib/rand/`
  - `http://www.cplusplus.com/reference/cstdlib/srand/`
  - `http://www.cplusplus.com/reference/cstdlib/RAND_MAX/`

Exercițiu - pași:

- Citește n, *min* și *max* de la tastatură
- Generează un șir aleatoriu de n elemente cu valori cuprinse intre *min* și *max*
- Șirul trebuie să fie diferit la fiecare rulare a programului
- Adaugă șirul în fișierul *output.txt*

2.3.3 Generare grafice

Pentru generarea graficelor poți folosi:

- Biblioteca *Profiler* de pe Moodle (sau `https://github.com/cypryoprisa/utcn-fa-profiler`)
- Microsoft Office Excel

2.3.4 Microsoft Office Excel

Va trebui să creezi un fișier cu extensia *.csv* (comma-separated values). Fișierul ar trebui să aibă o structură de felul următor:

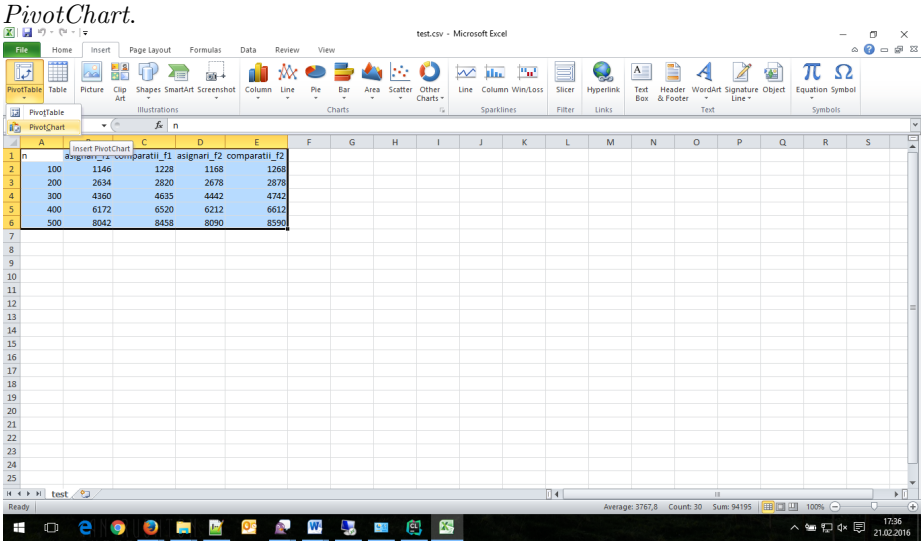
n,asignari_f1,comparatii_f1,asignari_m2,comparatii_m2
100,1146,1228,1168,1268
200,2634,2820,2678,2878
300,4360,4635,4442,4742
400,6172,6520,6212,6612
500,8042,8458,8090,8590

Legendă:

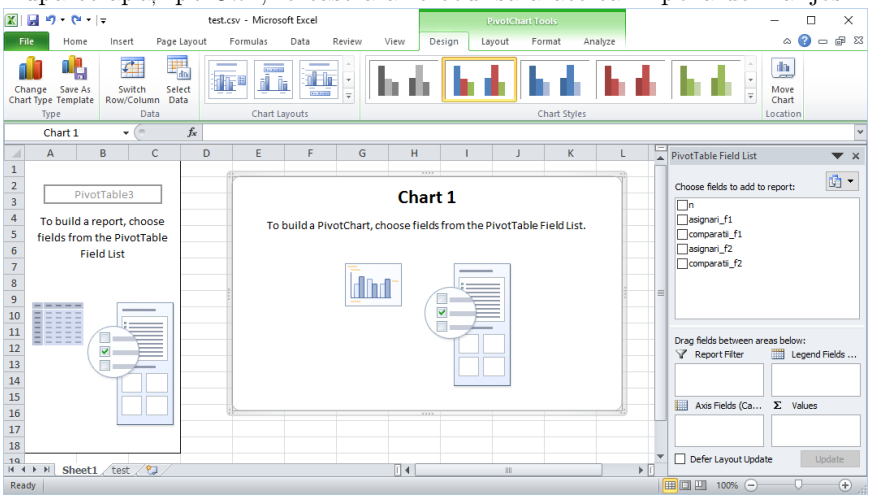
- n=dimensiunea problemei (ex: lungimea șirului de intrare)
- asignari_f1=numărul de asignări pentru cazul favorabil și metoda 1
- comparatii_f2=numărul de comparații pentru cazul favorabil și metoda 2

Atenție: Dacă deschizi fișierul CSV în Excel și valorile apar pe o singura coloană înseamnă că trebuie să folosești alt caracter de separare (ex: folosește punct și virgulă ";").

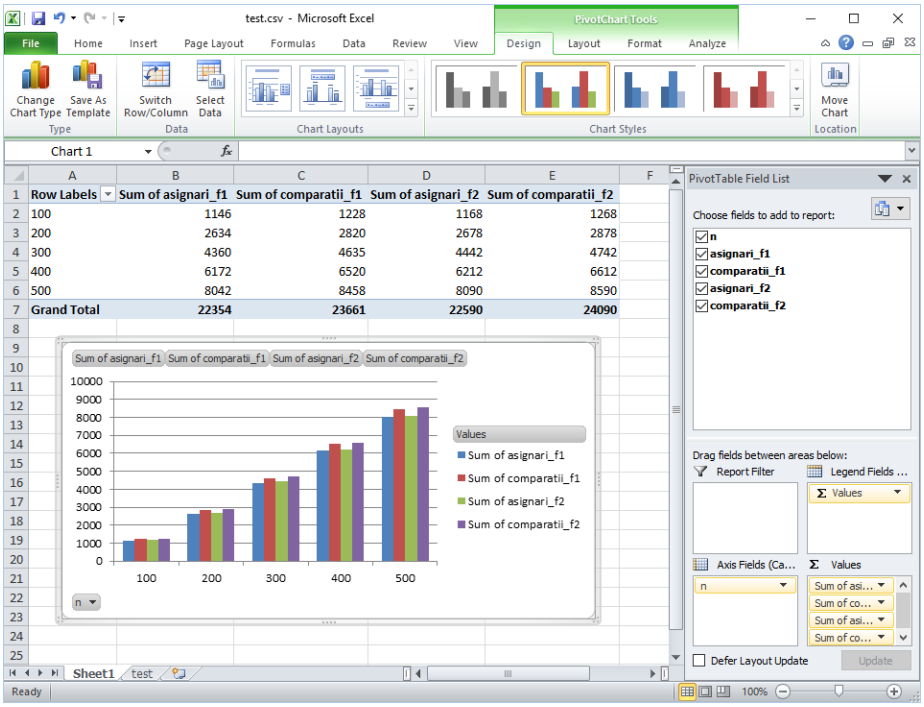
După ce ai deschis fișierul CSV în Excel, selectează toate valorile și creează un *PivotChart*.



După ce apeși pe *Ok*", fereastra ar trebui să arate ca în poza de mai jos.

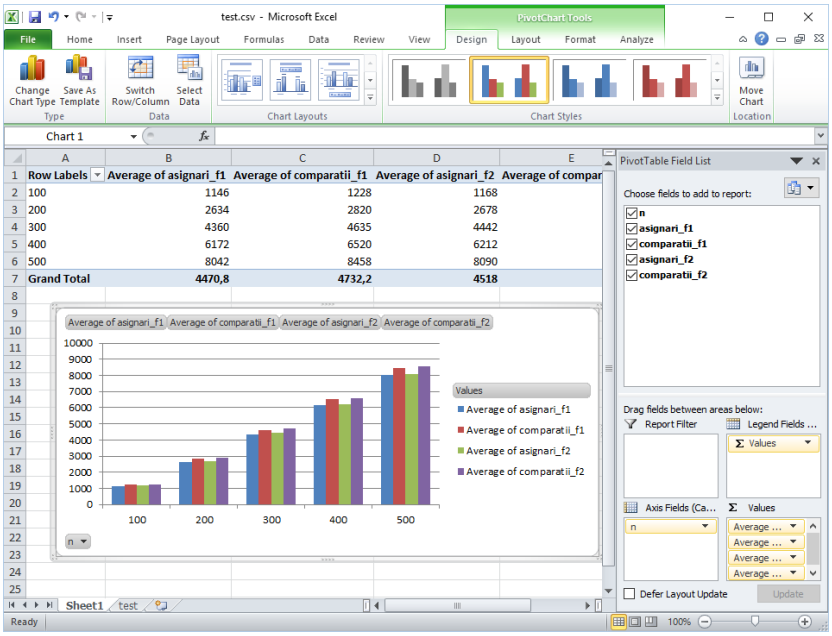


În panoul din stânga trage "*n*" în zona "*Axis Fields*" și celelalte coloane în zona "*Values*".



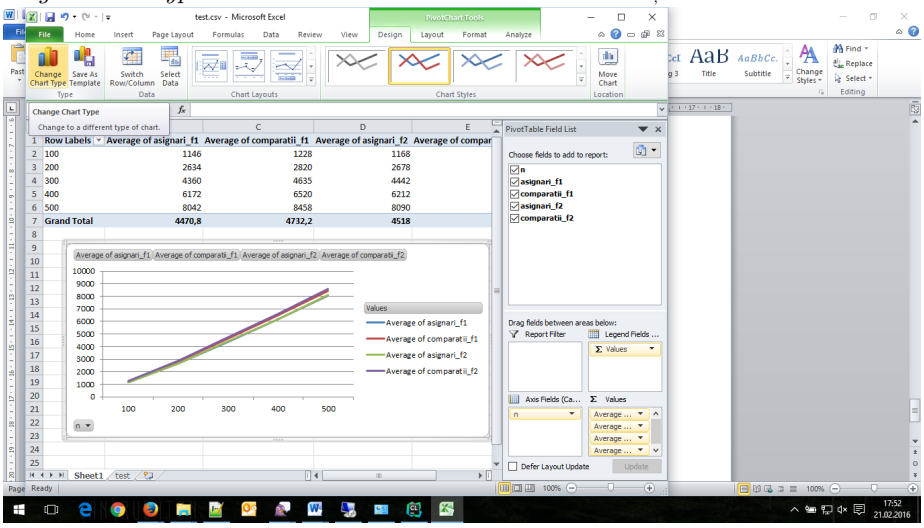
Row Labels	Sum of asignari_f1	Sum of comparatii_f1	Sum of asignari_f2	Sum of comparatii_f2
100	1146	1228	1168	1268
200	2634	2820	2678	2878
300	4360	4635	4442	4742
400	6172	6520	6212	6612
500	8042	8458	8090	8590
Grand Total	**22354**	**23661**	**22590**	**24090**

Schimbă funcția de agregare "*sum*" în "*average*": click pe săgeata neagra de la fiecare rând din zona "*Values*", apoi "*Value Field Settings*" și alege "*Average*". Dacă le-ai schimbat corect, fereastra ar trebui să arate ca în poza de mai jos.



Row Labels	Average of asignari_f1	Average of comparatii_f1	Average of asignari_f2	Average of compar
100	1146	1228	1168	
200	2634	2820	2678	
300	4360	4635	4442	
400	6172	6520	6212	
500	8042	8458	8090	
Grand Total	**4470,8**	**4732,2**	**4518**	

Ultimul pas este să schimbi tipul graficului intr-un grafic de tip linie de la "*Change Chart Type*". Rezultatul final ar trebui să arate așa:



2.3.5 Exercițiu

Scrie un program care pentru fiecare n din intervalul $\{100, 200, \ldots, 10.000\}$ calculează și adaugă într-un fișier următoarele valori:

n, 100*log(n), 10*n, n*log(n), 0.1*n^2, 0.01*n^3

Folosește valorile din fișier ca să generezi un grafic în funcție de n.

3 Tema nr. 1: Analiza și Compararea Metodelor Directe de Sortare

Timp alocat: 2 ore

3.1 Implementare

Se cere implementarea **corectă** și **eficientă** a 3 metode directe de sortare (*Sortarea Bulelor*, *Sortarea prin Inserție* – folosind inserție liniară sau binară, și *Sortarea prin Selecție*)

- Intrare: un șir de numere $< a_1, a_2, \ldots, a_n >$
- Ieșire: o permutare ordonată a șirului de la intrare $< a'_1 \leq a'_2 \leq \ldots \leq a'_n >$

Toate informațiile necesare și pseudo-codul se găsesc în notițele de la **Seminarul nr. 1** (Sortarea prin Inserție este prezentată și în **carte[1] – secțiunea 2.1**). Să verificați că ați implementat varianta eficientă pentru fiecare din algoritmii de sortare (dacă mai multe versiuni au fost prezentate)

3.2 Cerințe minimale pentru notare

- Interpretați graficul și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Pregatiti un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu nu preluam teme unde tot codul este pus in main)
- ***Punctajele din barem se dau pentru rezolvarea corectă și completă a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de voi la întrebările puse de către profesor.***

3.3 Cerințe

3.3.1 Implementarea metodelor de sortare (5.5p)

- Bubble sort (1.5p)
- Insertion sort (2p)
- Selection sort (2p)

Corectitudinea algoritmilor va trebui demonstrată pe un vector de dimensiuni mici (care poate să fie codat în funcția *main*).

3.3.2 Analiză algoritmilor pentru caz mediu statistic (1.5p – 0.5 per algoritm)

! Înainte de a începe să lucrați pe partea de evaluare a complexitatii algoritmilor, asigurati-va că aveti un algoritm corect!

Se cere compararea celor 3 algoritmi în cazurile: **favorabil (best)**, **mediu statistic (average)** și **defavorabil (worst)**. Pentru cazul **mediu** va trebui să repetați măsurătorile de **m** ori (m=5 este suficient) și să raportați media rezultatelor; de asemenea, pentru cazul **mediu,** asigurați-vă că folosiți **aceleași** date de intrare pentru cele 3 metode de sortare (astfel încât compararea lor să fie corectă); identificați și generați date de intrare pentru cazurile: **favorabil** și **defavorabil**, pentru toate cele 3 metode de sortare.

Pașii de analiză ai metodelor de sortare pentru fiecare din cele 3 cazuri (**favorabil, defavorabil, mediu)**:

- variați dimensiunea șirului de la intrare (n) între [100...10.000], cu un increment de maxim 500 (sugerăm 100).

- pentru fiecare dimensiune, generati datele de intrare adecvate pentru metoda de sortare; rulați metoda de sortare numărând operațiile (numărul de atribuiri, numărul de comparații și suma lor).

> **!** Doar atribuirile (=) și comparațiile (<, ==, >, !=) care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

3.3.3 Analiză in caz favorabil și defavorabil (3p - câte 0.5p pentru fiecare caz a fiecărui algoritm)

Pentru fiecare caz de analiză (**favorabil, defavorabil** si **mediu),** generati grafice care compara cele 3 metode de sortare; folosiți grafice diferite pentru numărul de atribuiri, comparații și suma lor. Dacă o curbă nu poate fi vizualizată corect din cauza că celelalte curbe au o rată mai mare de creștere (ex: o funcție liniară pare constantă atunci când este plasată în același grafic cu o funcție pătratică), atunci plasați noua curbă și pe un alt grafic. Denumiti adecvat graficele și curbele.

Corectitudinea algoritmilor va trebui demonstrată pe un vector de dimensiuni mici (care poate să fie codat în funcția "main").

3.3.4 Bonus: Insertion sort prin inserție binară (0.5p)

Corectitudinea algoritmilor va trebui demonstrată pe un vector de dimensiuni mici (care poate să fie codat în funcția "main").

Va trebui să comparați insertion sort prin inserție binară cu toți ceilalți algoritmi (bubble, insertion, selection) pentru toate cazurile (favorabil, mediu statistic, defavorabil).

4 Tema Nr. 2: Analiza și Compararea a două metode de construire a structurii de date Heap: "De jos în sus" (Bottom-up) vs. "De sus în jos" (Top-down)

Timp de lucru: 2 ore

4.1 Implementare

Se cere implementarea **corectă** și **eficientă** a două metode de construire a structurii de date Heap i.e., "de jos în sus" (*bottom-up*) și "de sus în jos" (*top-down*). De asemenea, se cere implementarea algoritmului *heapsort.*

Informații utile și pseudo-cod găsiți în notițele de curs sau în bibliografie[1]:

- "*De jos în sus*": secțiunea 6.3 (Building a heap)
- *Heapsort:* sectiunea 6.4 (The heapsort algorithm)
- "*De sus în jos*": secțiunea 6.5 (Priority queues) și problema 6-1 (Building a heap using insertion)

4.2 Cerințe minimale pentru notare

- Interpretați graficul și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.

- Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu nu preluam teme unde tot codul este pus in main)
- ***Punctajele din barem se dau pentru rezolvarea corectă și completă a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de voi la întrebările puse de către profesor.***

4.3 Cerințe

4.3.1 Analiza comparativă a *unuia* din algoritmii de sortare din L1 (la alegere) în versiune *iterativă* vs *recursivă*. Analiza se va efectua atât din *perspectiva numărului de operații, cât și a timpului de rulare* (2p)

Pentru analiza comparativă a versiunii iterative vs recursive, alegeți oricare din cei 3 algoritmi din laboratorul 1 (bubble sort, insertion sau selection). Folosiți varianta iterativă pe care ați implementat-o și predat-o în cadrul laboratorului (corectată, dacă este nevoie, în funcție de feedback-ul pe care l-ati primit) și implementați același algoritm de sortare în mod recursiv.

Trebuie să măsurați atât efortul total, cât și timpul de rulare al celor două versiuni (iterativ și recursiv) => două grafice, fiecare comparand cele două versiuni de algoritm.

Pentru a măsura timpul de execuție puteți folosi Profiler similar cu exemplul de mai jos.

```
profiler.startTimer("your_function", current_size);
for(int test=0; test<nr_tests; ++test) {
   your_function(array, current_size);
}
profiler.stopTimer("your_function", current_size);
```

Numărul de teste (*nr_tests* din exemplul de mai sus) trebuie ales în funcție de procesor și modul de compilare. Sugerăm valori mai mari, precum 100 sau 1000.

4.3.2 Implementarea metodei bottom-up de construire a unui heap (2p)

Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

4.3.3 Implementarea metodei top-down de construire a unui heap (2p)

Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

4.3.4 Analiza comparativă a celor două metode de construire în cazul mediu statistic (2p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un algoritm *corect*!

Se cere compararea celor două metode de constructive a structurii heap în cazul mediu statistic. Pentru cazul mediu statistic va trebui să repetați măsurătorile de m ori (m=5) și să raportați valoarea lor medie; de asemenea, pentru cazul mediu statistic, asigurați-vă că folosiți aceleași date de intrare pentru cele două metode.

Pașii de analiză:

- variați dimensiunea șirului de intrare (n) între $[100 \dots 10000]$, cu un increment de maxim 500 (sugerăm 100).

- pentru fiecare dimensiune (n), generați date de intrare adecvate metodei de construcție; rulați metoda numărând operațiile elementare (atribuiri și comparații - pot fi numărate împreună pentru această temă).

! Doar atribuirile și comparațiile care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

Generați un grafic ce compară cele două metode de construcție în cazul mediu statistic pe baza numărului de operații obținut la pasul anterior. Dacă o curba nu poate fi vizualizată corect din cauza că celelalte curbe au o rată mai mare de creștere, atunci plasați noua curbă și pe un alt grafic. Denumiți adecvat graficele și curbele.

4.3.5 Analiza comparativă a metodelor de construcție în cazul defavorabil (1p)

4.3.6 Implementarea și exemplificarea corectitudinii algoritmului heapsort (1p)

Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

5 Tema Nr. 3: Analiza și compararea metodelor avansate de sortare – HeapSort și QuickSort / QuickSelect

Timp alocat: 2 ore

5.1 Implementare

Se cere implementarea **corectă** și **eficientă** a Sortării Rapide (*Quicksort*), Sortării Rapide Hibridizate (*Hybrid Quicksort*) și *Quick-Select* (*Randomized-Select*). Se cere și analizarea comparative a complexității Sortării folosind Heapuri *(Heapsort,* implementat în Tema Nr. 2*)* și Sortarea Rapidă *(Quicksort).*

Informații utile și pseudo-cod găsiți în notițele de curs sau în carte[1]:

- *Heapsort*: capitolul 6 (Heapsort)
- *Quicksort*: capitolul 7 (Quicksort)
- *Hibridizare quicksort utilizând insertion sort iterativ* - in quicksort, pentru dimensiuni de șir $<$ prag, se utilizeaza insertion sort (folosiți implementarea insertion sort din Tema Nr. 1).
- *QuickSelect/Randomized Select:* capitolul 9

5.2 Cerințe minimale pentru notare

- Interpretați graficul și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu nu preluam teme unde tot codul este pus in main)
- ***Punctajele din barem se dau pentru rezolvarea corectă și completă a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de voi la întrebările puse de către profesor.***

5.3 Cerințe

5.3.1 QuickSort: implementare (2p)

Corectitudinea algoritmului va trebui exemplificată pe date de intrare de dimensiuni mici.

5.3.2 QuickSort: evaluare în caz mediu statistic, favorabil si defavorabil (3p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

Pașii de analiză:

- variați dimensiunea șirului de intrare (n) între $[100 \ldots 10000]$, cu un increment de maxim 500 (sugerăm 100).

- pentru fiecare dimensiune (n), generați date de intrare adecvate metodei de construcție; rulați metoda numărând operațiile elementare (atribuiri și comparații - pot fi numărate împreună pentru această temă).

! Doar atribuirile și comparație care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

5.3.3 QuickSort și HeapSort: analiză comparativă a cazului mediu statistic (2p)

Se cere compararea celor două metode de sortare în cazul **mediu statistic**. Pentru cazul **mediu statistic** va trebui să repetați măsurătorile de m ori (m=5) și să raportați valoarea lor medie; de asemenea, pentru cazul **mediu statistic**, asigurați-vă că folosiți **aceleași** date de intrare pentru cele două metode.

Generați un grafic ce compară cele două metode de construcție în cazul **mediu statistic** pe baza numărului de operații obținut la pasul anterior.

Dacă o curba nu poate fi vizualizată corect din cauza că celelalte curbe au o rată mai mare de creștere, atunci plasați noua curbă pe un alt grafic. Denumiți adecvat graficele și curbele.

5.3.4 Implementarea hibridizării quicksort-ului (1p)

Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

5.3.5 Determinare a unui prag optim în hibridizare + motivație (grafice/măsuratori) (1p)

Determinarea optimului din perspectiva pragului utilizat se realizează prin varierea valorii de prag pentru care se aplică insertion sort.

Comparați rezultatele obținute din perspectiva performanței (timpului de execuție și a numărului de operații) pentru a determina o valoare optima a pragului. Puteți folosi 10,000 ca dimensiune fixă a vectorului ce urmează sa fie sortat și variați pragul între [5,50] cu un increment de 1 până la 5.

Numărul de teste care trebuie sa fie repetate (nr_tests din exemplul de mai sus) trebuie ales în funcție de procesor și modul de compilare. Sugerăm valori mai mari, precum 100 sau 1000.

5.3.6 Analiză comparativă (între *quicksort și quicksort hibridizat*) din punct de vedere a numărului de operații și a timpului efectiv de execuție (1p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

Pentru hibridizare quicksort, trebuie să utilizați versiunea iterativă de insertion sort din prima temă în cazul în care dimensiunea vectorului este mică

(sugerăm utilizarea insertion sort dacă vectorul are sub 30 de elemente). Comparați *timpul de rulare și numărul de operații* (asignări + comparații) pentru quicksort implementat în tema 3 cu cel hibridizat.

Pentru a măsura timpul de execuție puteți folosi Profiler similar cu exemplul de mai jos.

```
profiler.startTimer("your_function", current_size);
for(int test=0; test<nr_tests; ++test) {
   your_function(array, current_size);
}
profiler.stopTimer("your_function", current_size);
```

În momentul în care măsurați timpul de execuție, asigurați-vă că opriți orice alte procese care nu sunt necesare.

5.3.7 Bonus: QuickSelect - Randomized-Select (0.5p)

Corectitudinea algoritmului va trebui exemplificată pe date de intrare de dimensiuni mici.

Pentru QuickSelect (Randomized-Select) nu trebuie facuta analiza complexității, doar corectitudinea trebuie exemplificată.

6 Tema Nr. 4: Interclasarea eficientă a k șiruri ordonate

Timp alocat: 2 ore

6.1 Implementare

Se cere implementarea **corectă** și **eficientă** a unei metode de complexitate $O(nlogk)$ pentru **interclasarea a k șiruri sortate**. Unde n este numărul total de elemente (Sugestie: folosiți un heap, vezi notițele de la *Seminarul al 2-lea*).

Cerințe de implementare:

- Folosiți liste înlănțuite pentru a reprezenta cele k șiruri sortate și secvența de ieșire

Intrare: k șiruri de numere $< a_1^i, a_2^i, \ldots, a_{m_i}^i >$, $\sum_{i=1}^{k} m_i = n$
Ieșire: o permutare a reuniunii șirurilor de la intrare $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

6.2 Cerințe minimale pentru notare

- Interpretați graficul și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.

- Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu nu preluam teme unde tot codul este pus in main)
- ***Punctajele din barem se dau pentru rezolvarea corectă și completă a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de voi la întrebările puse de către profesor.***

6.3 Cerințe

6.3.1 Demo pentru generarea a *k* liste aleatoare sortate de dimensiuni diferite (având în total *n* elemente, unde *n* și *k* sunt date) și interclasarea a 2 liste (5p)

Corectitudinea algoritmului (*generare și interclasare*) va trebui demonstrată pe date de intrare de dimensiuni mici (ex: k=4, n=20).

6.3.2 Adaptare operațiilor de *min-heap* pe structura nouă și interclasarea a k liste (3p)

Corectitudinea algoritmului (*interclasare*) va trebui demonstrată pe date de intrare de dimensiuni mici (ex: k=4, n=20).

6.3.3 Evaluarea algoritmului în cazul mediu statistic (2p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

Se cere analiza algoritmului în cazul **mediu statistic**. Pentru cazul **mediu statistic** va trebui să repetați măsurătorile de câteva ori. Din moment ce **k** și **n** pot varia, se va face o analiză în felul următor:

- Se alege, pe rând, 3 valori constante pentru k (**k1=5, k2=10, k3=100**); generează k șiruri **aleatoare** sortate pentru fiecare valoare a lui k astfel încât numărul elementelor din toate șirurile să varieze între **100 și 10000** cu un increment maxim de 400 (sugerăm 100); rulați algoritmul pentru toate valorile lui n (pentru fiecare valoare a lui k); generați un grafic ce reprezintă **suma atribuirilor și a comparațiilor** făcute de acest algoritm pentru fiecare valoare a lui k (în total sunt 3 curbe).
- Se alege ***n=10.000***; valoarea lui k va varia între **10 și 500** cu un increment de 10; generați k șiruri **aleatoare** sortate pentru fiecare valoare a lui k astfel încât numărul elementelor din toate șirurile să fie 10000; testați algoritmul de interclasare pentru fiecare valoare a lui k și generați un grafic care reprezintă **suma atribuirilor și a comparațiilor**.

7 Tema Nr. 5: Căutarea în tabele de dispersie

Adresare deschisă, verificare pătratică

Timp alocat: 2 ore

7.1 Implementare

Se cere implementarea **corectă** și **eficientă** a operațiilor de *inserare* și *căutare* într-o tabelă de dispersie ce folosește *adresarea deschisă* cu *verificare pătratică*.

Informații utile și pseudo-cod găsiți în notițele de curs sau în carte (Cormen), in secțiunea *11.4 Open addressing*.

Noțiunile închis/deschis (closed/open) specifică dacă se obligă folosirea unei poziții sau structuri de date.

7.1.1 Hashing (se referă la tabela de dispersie (hash table))

- Open Hashing
  - Pe o anumită poziție se pot stoca mai multe elemente (ex: chaining)
- Closed Hashing
  - Se poate stoca doar un singur element pe o anumita pozitie (ex. linear/quadratic probing)

Addressing (se referă la poziția finală a unui element față de poziția inițială)

- Open Addressing (Adresare Deschisă)
  - Adresa finală (poziția finală) nu este complet determinată de către codul hash. Poziția depinde și de elementele care sunt deja în tabelă. (ex: linear/quadratic probing - verificare liniară/pătratică)
- Closed Addressing (Adresare Închisă)
  - Adresa finală este întotdeauna determinată the codul hash (poziția inițială calculată) și nu exista probing (ex: chaining)

Pentru această temă tabela de dispersie va avea o structură similară cu cea de mai jos:

typedef struct { int id; char name[30]; } Entry;

unde *adresa finală* în tabelă va fi calculată aplicând funcția de hashing pe câmpul *id* din structură. Câmpul *name* va fi folosit doar pentru demonstrarea corectitudinii operațiilor de căutare și ștergere și nu este necesară pentru evaluarea performanței (câmpul *name* va fi afișat în consolă dacă operația de căutare găsește *id*-ul respectiv, altfel afișați "negăsit").

7.2 Cerințe minimale pentru notare

- Interpretați graficul și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu nu preluam teme unde tot codul este pus in main)
- ***Punctajele din barem se dau pentru rezolvarea corectă și completă a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de voi la întrebările puse de către profesor.***

7.3 Cerințe

7.3.1 Implementarea operației de inserare și căutare folosind structura de date cerută + *demo (dimensiune 10)* (5p)

Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (ex. 10).

7.3.2 Evaluarea operației de căutare pentru un singur factor de umplere 95% (2p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

Se cere evaluarea operației de *căutare* în tabele de dispersie cu adresare deschisă și verificare pătratică, în cazul **mediu statistic** (nu uitați să repetați măsurătorile de 5 ori). Pentru a obține evaluarea, trebuie să:

1. Alegeți N, dimensiunea tabelei, un număr prim în jur de 10000 (e.g. 9973, sau 10007);
2. Pentru fiecare din următoarele valori pentru factorul de umplere $\alpha = \mathbf{0.95}$:
   a. Inserati în tabela n elemente aleator, astfel incat sa ajungeți la valoare lui α ($\alpha = n/N$)
   b. Căutați aleator, în fiecare caz, m elemente (m ~ 3000), astfel încât aproximativ jumătate din elemente sa fie *găsite*, iar restul sa *nu fie găsite* (în tabelă). *Asigurați-vă că elementele găsite sunt generate uniform, i.e. să căutați elemente care au fost introduse la moment diferite, cu probabilitate egală (există mai multe moduri în care se poate garanta acest lucru)*
   c. Numărați operațiile efectuate de procedura de căutare (i.e. numărul de celule accesate)

d. Atenție la valorile pe care le căutați, să fie într-o ordine aleatoare din cele introduse. *Dacă sunt primele 1500 adăugate, implicit efortul mediu găsite va fi 1.*

3. Generați un tabel de forma:

Table 1: Effort measurements at various filling factors

Filling factor	**Avg. Effort *(found)***	**Max Effort *(found)***	**Avg. Effort *(not-found)***	**Max Effort *(not-found)***
0.95	...	...	...	...

Efort mediu = efort_total / nr_elemente
Efort maxim = număr maxim de accese efectuat de o operație de căutare

7.3.3 Completarea evaluării pentru toți factorii de umplere (2p)

Respectând cerințele de la punctul 2 cu $\alpha \in \{0.8, 0.85, 0.9, 0.95, 0.99\}$, generați un tabel de forma:

Table 2: Effort measurements at various filling factors

Filling factor	**Avg. Effort *(found)***	**Max Effort *(found)***	**Avg. Effort *(not-found)***	**Max Effort *(not-found)***
0.80				
0.85				
...	...		...	

7.3.4 Implementare operației de ștergere într-o tabelă de dispersie, *demo (dimensiune 10)* și evaluarea operației de căutare după ștergerea unor elemente (1p)

Pentru evaluarea operației de de căutare după ștergere, umpleți tabela de dispersie pâna la factor de umplere 0.99. Stergeti elemente din tabela până ajungeți la factor umplere 0.8, după care căutați m elemente aleator (m ~ 3000), astfel incat aproximativ jumătate din elemente sa fie *găsite*, iar restul sa nu fie găsite (în tabela). Măsurați efortul necesar pentru *căutare* și adăugați în tabelul generat anterior.

8 Tema Nr. 6: Arbori Multicăi

Transformări între diferite reprezentări
Timp alocat: 2 ore

8.1 Implementare

1. Se cere implementarea **corectă** și **eficientă** a traversării *iterative* și *recursive* a unui arbore binar. Puteți găsi orice informații necesare și pseudocod în notele de curs și seminar.

2. În plus, se cere implementarea **corectă** și **eficientă** a unor algoritmi de complexitate *liniară* pentru transformarea arborilor multicăi între următoarele reprezentări:

    1. **R1**: *reprezentarea părinte*: pentru fiecare index, valoare din vector reprezintă indexul părintele, ex: $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$
    2. **R2**: *reprezentare arbore multicăi*: fiecare nod conține cheia si un vector de noduri copil
    3. **R3**: *reprezentare binara:* fiecare nod conține cheia si doi pointeri: unul către primul copil si al doilea către fratele din dreapta (ex: următorul frate).

Așadar, trebuie să definiți transformarea **T1** din *reprezentarea părinte* (**R1**) în *reprezentarea arbore multicăi* (**R2**), iar apoi transformarea **T2** din *reprezentarea arbore multicăi* (**R2**) în *reprezentarea binară* (**R3**). Pentru toate reprezentările (**R1**, **R2**, **R3**) trebuie să implementați afișarea prietenoasă (pretty print, **PP**) (vezi pagina 2).

Definiți structurile de date. Puteți folosi structuri intermediare (ex: memorie adițională).

8.2 Cerințe

8.2.1 Implementare a parcurgerii iterative și recursive a unui arbore binar în *O(n)* și cu *memorie aditională constantă* (3p)

Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

8.2.2 Implementarea transformărilor între diferite reprezentări

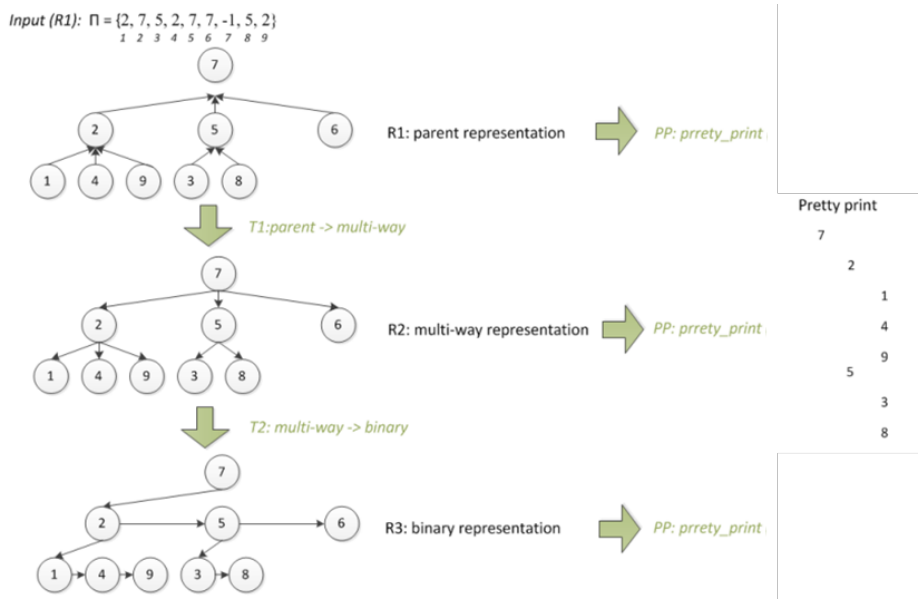
8.2.3 Implementarea corectă la pretty-print la *R1* (2p)

8.2.4 Implementarea corectă la *T1 (din R1 în R2)* și pretty-print la *R2* (1p) + *T1* în timp liniar (1p)

8.2.5 Implementarea corectă la *T2* (din R2 în R3) și pretty-print la *R3* (2p) + *T2* în timp liniar (1p)

Corectitudinea algoritmilor va trebui demonstrată pe exemplul $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$.

Folosiți afișarea prietenoasă pentru cele trei reprezentări. *Fiecare reprezentare (R1,R2,R3) necesită o afișare prietenoasă cu o implementare diferită dar aceeași afișare.*



Analizați eficienta în timp și spațiu a celor două transformări. Ați atins *O(n)*? Ați folosit memorie adițională?

9 Tema Nr. 7: Statistici dinamice de ordine

Timp alocat: 2 ore

9.1 Implementare

Se cere implementarea **corectă** și **eficientă** a operațiilor de management ale unui **arbore de statistică de ordine** (*capitolul 14 din carte*[1]).

[1]

Se cere să folosiți un *arbore binar* de *căutare* **perfect echilibrat**. Fiecare nod din arbore trebuie extins cu un câmp *size* (dimensiunea sub-arborelui ce are nodul ca rădăcină).

Operațiile de management ale unui **arbore de statistică de ordine**:

- BUILD_TREE(n)
  - *construiește* un arbore binar de căutare **echilibrat** cu cheile 1,2,...,n (*hint*: *divide et impera*)
  - nu uitați să inițializați câmpul *size*
- OS-SELECT(tree, i)
  - selectează elementul cu a i-a cea mai mică cheie
  - pseudocodul poate fi găsit la *Capitolul 14.1 din carte[1]*
- OS-DELETE(tree, i)
  - puteți folosi ștergerea dintr-un arbore binar de căutare, fără a crește înălțimea arborelui (De ce nu trebuie să re-balansați arborele?)
  - nu uitați să păstrați câmpul *size* consistent o dată cu ștergerile din arbore
  - există mai multe abordări prin care puteți modifica câmpul *size* fără a crește complexitatea algoritmului (găsiți cea mai bună soluție)

Seamănă OS-SELECT cu ceva ce ați studiat în acest semestru?

9.2 Cerințe

9.2.1 BUILD_TREE: implementare corectă și eficientă (5p)

Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

- afișați (cu pretty print) arborele construit inițial

9.2.2 OS_SELECT: implementare corectă și eficientă (1p)

Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

- executați OS-SELECT pentru câțiva (cel puțin 3) indecși selectați aleator.

9.2.3 OS_DELETE: implementare corectă și eficientă (2p)

Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

- executați secvența OS-SELECT urmat de OS-DELETE pentru câțiva (cel puțin 3) indecși selectați aleator (3) și *afișati arborele dupa fiecare execuție.*

9.2.4 Evaluarea operațiilor de management - BUILD, SELECT, DELETE (2p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

După ce sunteți siguri că algoritmul funcționează corect:

- variați n de la 100 la 10000 cu un pas de 100;
- pentru fiecare n (nu uitați să repetați de 5 ori)
  - construiți (BUILD) arborele cu elemente de la *1 la n*
  - repetați de n ori secvența OS-SELECT urmat OS-DELETE folosind un index selectat aleator dintre elementele rămase în arbore
  - Evaluați numărul de operații necesare pentru fiecare operație de management (BUILD, SELECT, DELETE – *reprezentați rezultatele sub forma unui grafic cu trei serii*). Evaluați complexitatea operațiilor de management ca și suma atribuirilor și a comparațiilor pentru fiecare valoare a lui n.

9.2.5 Bonus: Implementarea utilizând AVL / arbori roșu și negru (1p)

10 Tema Nr. 8: Mulțimi disjuncte

Timp alocat: 2 ore

10.1 Implementare

Se cere implementarea **corectă** și **eficientă** a operațiilor de bază pe **mulțimi disjuncte** (*capitolul 21.1*[2]) și a algoritmului lui **Kruskal** (găsirea arborelui de acoperire minimă, *capitolul 23.2*[1]) folosind mulțimi disjuncte.

Se cere să folosiți o pădure de arbori pentru reprezentarea mulțimilor disjuncte. Fiecare arbore trebuie extins cu un câmp *rank* (înălțimea arborelui).

Operațiile de bază pe **mulțimi disjuncte** sunt**:**

- MAKE_SET (x)
  - creează o mulțime nouă ce conține elementul x
- UNION (x, y)
  - realizează reuniunea dintre mulțimea care îl conține pe x și mulțimea care îl conține pe y

[2]Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*

  - euristica *union by rank* ține cont de înălțime celor doi arbori pentru a realiza reuniunea dintre mulțimi
  - pseudocodul poate fi găsit la *capitolul 21.3*[1]

- FIND_SET (x)
  - caută mulțime în care se află x
  - euristica *path compression* leagă toate elementele de pe ramura cu x la rădăcina arborelui

10.2 Cerințe

10.2.1 Implementare corectă a MAKE_SET, UNION și FIND_SET (5p)

Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici

- creați (MAKE) 10 mulțimi + afișare conținuturilor seturilor
- executați secvența UNION și FIND_SET pentru 5 elemente + afișare conținuturilor seturilor

10.2.2 Implementarea corectă și eficientă a algoritmului lui Kruskal (2p)

Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici

- creați un graf cu 5 noduri și 9 muchii + **afișare muchii**
- aplicarea algoritmului lui Kruskal + **afișarea muchiilor alese**

10.2.3 Evaluarea operațiilor pe mulțimi disjuncte (MAKE, UNION, FIND) folosind algoritmului lui Kruskal (3p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

O dată ce sunteți siguri că algoritmul funcționează corect:

- variați n de la 100 la 10000 cu un pas de 100
- pentru fiecare n
  - construiți un graf **conex**, **neorientat** și **aleatoriu** cu ponderi pe muchii (**n** noduri, **n*4** muchii)
  - determinați arborele de acoperire minima folosind algoritmul lui Kruskal

* evaluați efortul computațional al fiecărei operații de bază (MAKE, UNION, FIND – *reprezentați rezultatele sub forma unui grafic cu trei serii*) pe mulțimi disjuncte ca suma comparațiilor și atribuțiilor efectuate; astfel, ar trebui să existe **3 serii în grafic**, câte una pentru fiecare operație.

11 Tema Nr. 9: Căutarea în lățime (BFS)

11.1 Introducere

În acest laborator se va implementa algoritmul BFS (căutarea în lățime), conform secțiunii 22.2 din Cormen.

11.2 Structura proiectului

La acest laborator veți porni de la 3 fișiere:

- `main.cpp` - sursa principală, responsabilă cu interfața de vizualizare
- `bfs.h` - definiții pentru tipuri de structuri și funcții
- `bfs.cpp` - implementarea algoritmilor
- `grid.txt` - labirintul care va fi afișat și traversat
- `Profiler.h` - biblioteca pentru numărarea operațiilor și pentru grafice

!!! Pentru rezolvarea cerințelor trebuie să faceți modificări doar în `bfs.cpp`.

Pentru a vizualiza mai ușor funcționarea algoritmului, `main.cpp` va afișa o interfață de tip text, în care se va vedea o un labirint ce trebuie traversat (celulele negre sunt libere iar cele albe sunt pereți).

11.2.1 Inițializarea proiectului pe Windows, cu Visual Studio

Creați un proiect nou în Visual Studio, de tipul *"Empty Project"* și copiați fișierele de mai sus în folderul proiectului.

Adăugați cele doua fișiere `.h` la secțiunea *"Header Files"* și cele două fișiere `.cpp` la secțiunea *"Source Files"*, efectuând click dreapta → *"Add"* → *"Existing Item"*, ca în Figura 1.

11.2.2 Inițializarea proiectului pe Linux și Mac

Puteți edita fișierele proiectului cu orice editor doriți. Proiectul conține și un fișier `Makefile`, deci este suficient să rulați comanda `make` pentru compilarea acestuia. Executabilul `.exe` rezultat se va numi `main`, și se poate rula în terminal, executând `./main`.

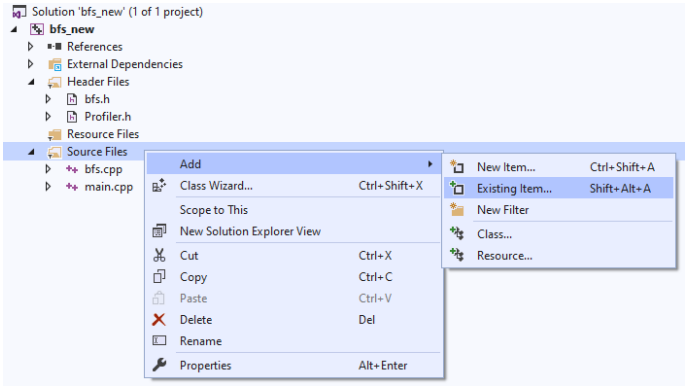


Figure 1: Fereastra *"Solution Explorer"* în Visual Studio

11.3 Funcționare

La pornirea programului se va afișa labirintul, în mod similar cu Figura 2.
În partea de jos, utilizatorul poate tasta una din comenzile de mai jos:

- `exit`
  terminarea programului
- `clear`
  curățarea informațiilor anterioare din grilă
- `neighb <row> <col>`
  se vor afișa vecinii celulei de pe linia `<row>` și coloana `<col>`.
- `bfs <row> <col>`
  se va efectua o parcurgere BFS, pornind de la celula de la linia `<row>` și coloana `<col>`.
- `bfs_step <row> <col>`
  la fel ca la `bfs`, dar rezultatul se va afișa pas cu pas, în funcție de distanța față de sursă
- `bfs_tree <row> <col>`
  la fel ca la `bfs`, dar se va afișa și arborele BFS sub grilă
- `path <row1> <col1> <row2> <col2>`
  se va afișa cel mai scurt drum între celulele `(<row1> <col1>)` și `(<row2> <col2>)`
- `perf`
  se vor genera graficele pentru evaluarea performanței algoritmului

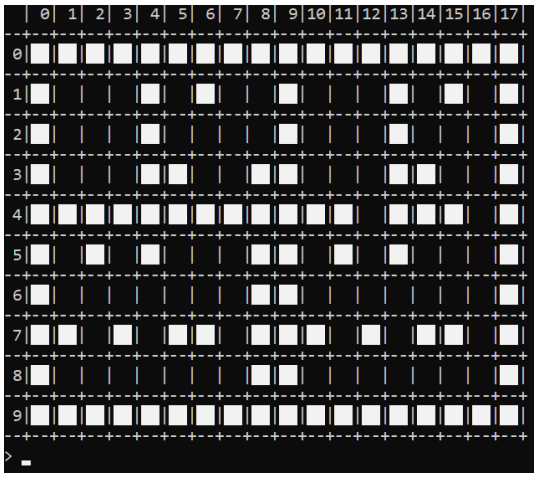


Figure 2: Interfața programului

11.3.1 Exemplu: comanda `neighb`

Dacă se rulează:
`neighb 2 3` ar trebui să apară imaginea din Figura 3.

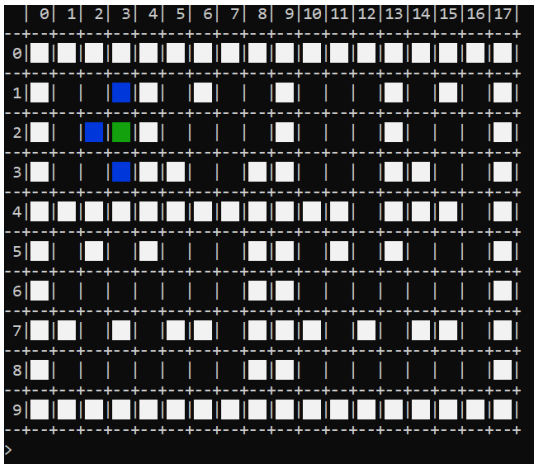


Figure 3: Rezultatul comenzii `neighb 2 3`

Celula de start se va colora cu verde, iar vecinii acesteia cu albastru.

În momentul de față funcția `get_neighbors()` nu este implementată, deci nu se va afișa rezultatul dorit. Puteți verifica dacă ați implementat corect această funcție rulând comanda pentru diverse celule. Fiecare celulă va avea maxim 4 vecini (sus, jos, stânga, dreapta), și nu trebuie afișate celule din afara grid-ului sau celule care conțin pereți.

11.3.2 Exemplu: comenzile `bfs` și `bfs_step`

Dacă se rulează:
`bfs 6 3` ar trebui să apară imaginea din Figura 4.

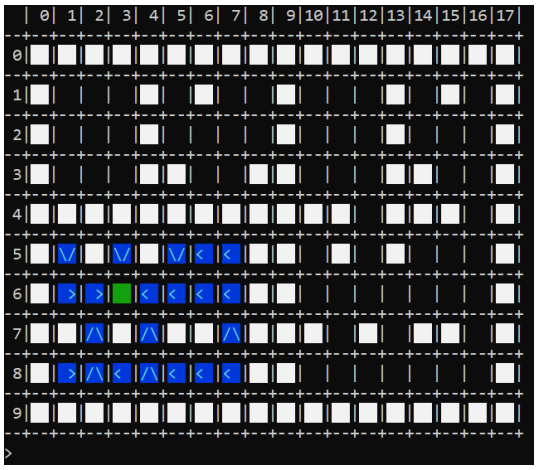


Figure 4: Rezultatul comenzii `bfs 6 3`

Celula de start se va colora cu verde, iar celulele parcurse se vor colora cu albastru. Pe fiecare celulă albastră va apărea o săgeată care va indica în ce direcție se află părintele din arborele BFS.

În momentul de față funcția `bfs()` nu este implementată, deci nu se va afișa rezultatul dorit. Puteți verifica dacă ați implementat corect această funcție rulând comanda pentru diverse celule.

11.3.3 Exemplu: comanda `bfs_tree`

Dacă se rulează:
`bfs 2 6` ar trebui să apară imaginea din Figura 5.

Rădăcina arborelui este nodul de start, respectiv `(2, 6)`. Copii acestui nod, sunt nodurile în care se poate ajunge direct din rădăcină: `(2, 5)`, `(2, 7)` și `(3, 6)` (ordinea acestora poate să difere în altă implementare).

11.3.4 Exemplu: comanda `path`

Dacă se rulează:
`path 5 10 3 15` ar trebui să apară imaginea din Figura 6.

Celula de start se va colora cu verde, cea de final cu roșu, iar celulele care fac parte din drumul cel mai scurt se vor colora cu albastru. Pe fiecare celulă albastră va apărea o săgeată care va indica direcția de mers.

În momentul de față funcțiile `shortest_path()` și `bfs()` nu sunt implementate, deci nu se va afișa rezultatul dorit. Puteți verifica dacă ați implementat corect aceste funcții rulând comanda pentru diverse perechi de celule.

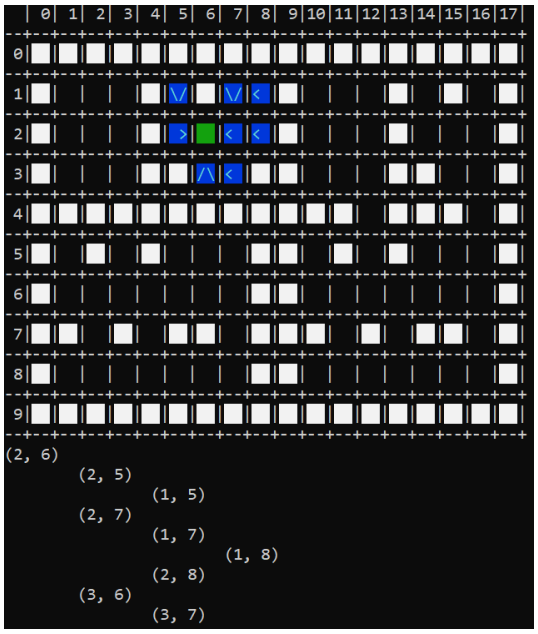


Figure 5: Rezultatul comenzii `bfs_tree 2 6`

11.3.5 Structuri de date folosite

În fișierul `bfs.h` sunt definite câteva structuri de date utile în cadrul framework-ului.

Structura `Grid` modelează o grilă, formată din `rows` linii și `cols` coloane, elementele acesteia fiind în matricea `mat`. O celulă liberă va avea valoarea 0, iar una ce conține un perete va avea valoarea 1.

Structura `Point` modelează un punct sau o celulă din grilă, câmpurile `row` și `col` reprezentând linia și coloana la care se află.

Structura `Node` modelează un nod din graf și conține următoarele câmpuri:

- `position` de tip `Point` reprezintă celula din grilă corespunzătoare nodului.
- `adjSize` - numărul de vecini ai nodului respectiv
- `adj` - vectorul de vecini, de dimensiune `adjSize`
- `color` - culoarea nodului; la început toate nodurile au culoarea `COLOR_WHITE`, adică valoarea 0
- `dist` - distanța față de nodul de start, în parcurgerea BFS
- `parent` - pointer la nodul părinte, în arborele BFS

Structura `Graph` modelează un graf și conține numărul de noduri `nrNodes` și vectorul `v` cu pointeri spre acestea.

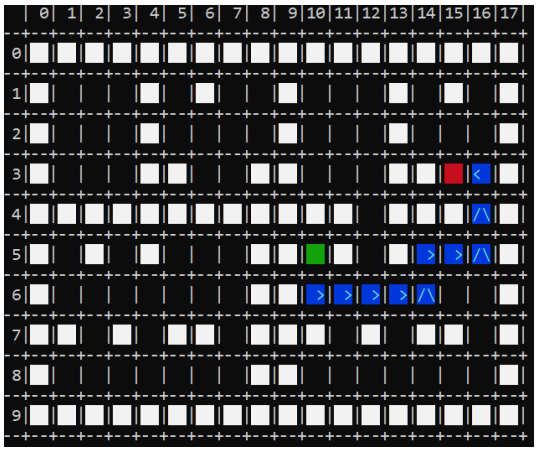


Figure 6: Rezultatul comenzii `path 5 10 3 15`

11.4 Cerințe

11.4.1 Determinarea vecinilor unei celule (2p)

În `bfs.cpp`, trebuie completată funcția `get_neighbors()` care primește ca parametri un pointer la structura de tip `Grid`, un punct `p` de tip `Point` și un vector de puncte `neighb` care se va completa cu vecinii punctului `p`. Funcția va returna numărul de vecini completați în vectorul `neighb`.

Un punct din grilă va avea maxim 4 vecini (sus, jos, stânga, dreapta). Nu toți vecinii sunt valizi: unii vecini pot ajunge în afara grilei (coordonate negative, sau peste dimensiuni) sau pot fi în interiorul unui zid. Din acest motiv, după ce calculați poziția unui vecin, ar trebui să verificați că aceasta cade în interiorul grilei, apoi că aceasta este liberă (valoarea din matrice la poziția respectivă e 0).

Vecinii valizi se vor completa în vectorul `neighb`. Se garantează că la apelul funcției din framework, acesta va avea cel puțin 4 elemente, deci nu se poate depăși capacitatea acestuia. Deoarece numărul de vecini completați poate fi mai mic de 4, trebuie să returnăm numărul acestora.

11.4.2 Implementarea algoritmului BFS (3p)

În `bfs.cpp`, trebuie completată funcția `bfs()` care primește ca parametri un pointer la structura de tip `Graph` și nodul de start `s` de tip `Node*`. Funcția va aplica algoritmul BFS conform secțiunii 22.2 din Cormen.

Nodurile din graf au la început culoarea `COLOR_WHITE`, iar câmpurile `dist` și `parent` sunt inițializate cu 0, respectiv `NULL`. După parcurgere, toate nodurile la care se poate ajunge din nodul de start trebuie să aibă culoarea `COLOR_BLACK`, distanța `dist` setată pe numărul de pași de la nodul de start până la nodul respectiv, iar pointerul `parent` trebuie să indice părintele în arborele BFS.

11.4.3 Afișarea arborelui BFS (2p)

În `bfs.cpp`, trebuie completată funcția `print_bfs_tree()` care primește ca parametru un pointer la structura de tip `Graph` pe care s-a rulat deja algoritmul BFS, deci culorile nodurilor și părinții sunt deja setate.

În funcția data, este deja implementată construcția unui vector de părinți `p`, în care nodurile colorate cu negru în parcurgerea BFS vor fi numerotate de la 0 la n. Deasemenea se construiește vectorul `repr` care conține coordonatele fiecărui nod.

Pentru afișarea acestui arbore se poate adapta codul din laboratorul de arbori multi-căi.

11.4.4 Evaluarea performanței algoritmului BFS (3p)

Funcția `performance()` realizează numărarea operațiilor, variind pe rând numărul de muchii, respectiv numărul de vârfuri al grafului. Pentru fiecare valoare, trebuie să implementați construcția unui graf aleator, conex, care să aibă numărul respectiv de vârfuri și de muchii.

În interiorul funcției `bfs()` va trebui să implementați numărarea propriu-zisă a operațiilor, folosind parametrul opțional `op`. Deoarece acest parametru este opțional, uneori funcția `bfs()` va fi apelată din framework cu valoarea acestuia setată pe `NULL`. Din acest motiv, atunci când numărați o operație, verificați tot timpul că `op` e un pointer valid, ca în exemplul de mai jos:

```
if(op != NULL) op->count();
```

11.4.5 Bonus: Determinarea celui mai scurt drum (0.5p)

În `bfs.cpp`, trebuie completată funcția `shortest_path()` care primește ca parametri un pointer la structura de tip `Graph`, nodurile de început și sfârșit `start` și `end` de tip `Node*`, respectiv vectorul `path`, ca parametru de ieșire în care se vor completa nodurile de pe traseu, în ordine. Funcția va returna numărul de noduri completate în vectorul `path`.

Pentru determinarea celui mai scurt drum între două noduri, se recomandă folosirea algoritmului BFS, implementat anterior, apoi reconstrucția drumului mergând din părinte în părinte în arborele BFS.

Vectorul `path`, în care se completează traseul, va avea o lungime de minim 1000 de elemente, la apelarea funcției. Returnați numărul de elemente care au fost completate în el, sau -1 în cazul în care nu se poate ajunge la nodul `end` pornind de la nodul `start`.

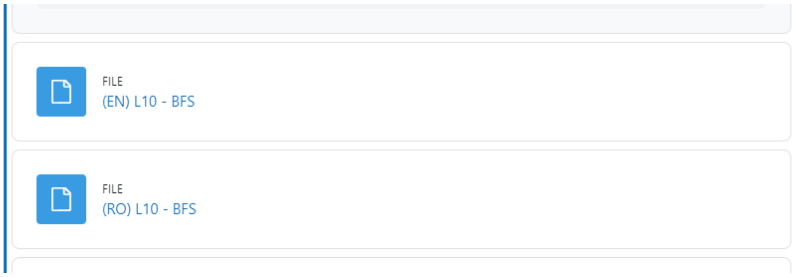
11.4.6 Bonus: Unde poate ajunge un cal pe tablă? (0.5p)

Folosind framework-ul dat, arătați că un cal poate ajunge pe orice poziție a unei table de șah goale, pornind din colțul din stânga-sus. Dați exemple de tablă care să conțină poziții libere la care nu se poate ajunge.

12 Tema Nr. 9: Tutorial

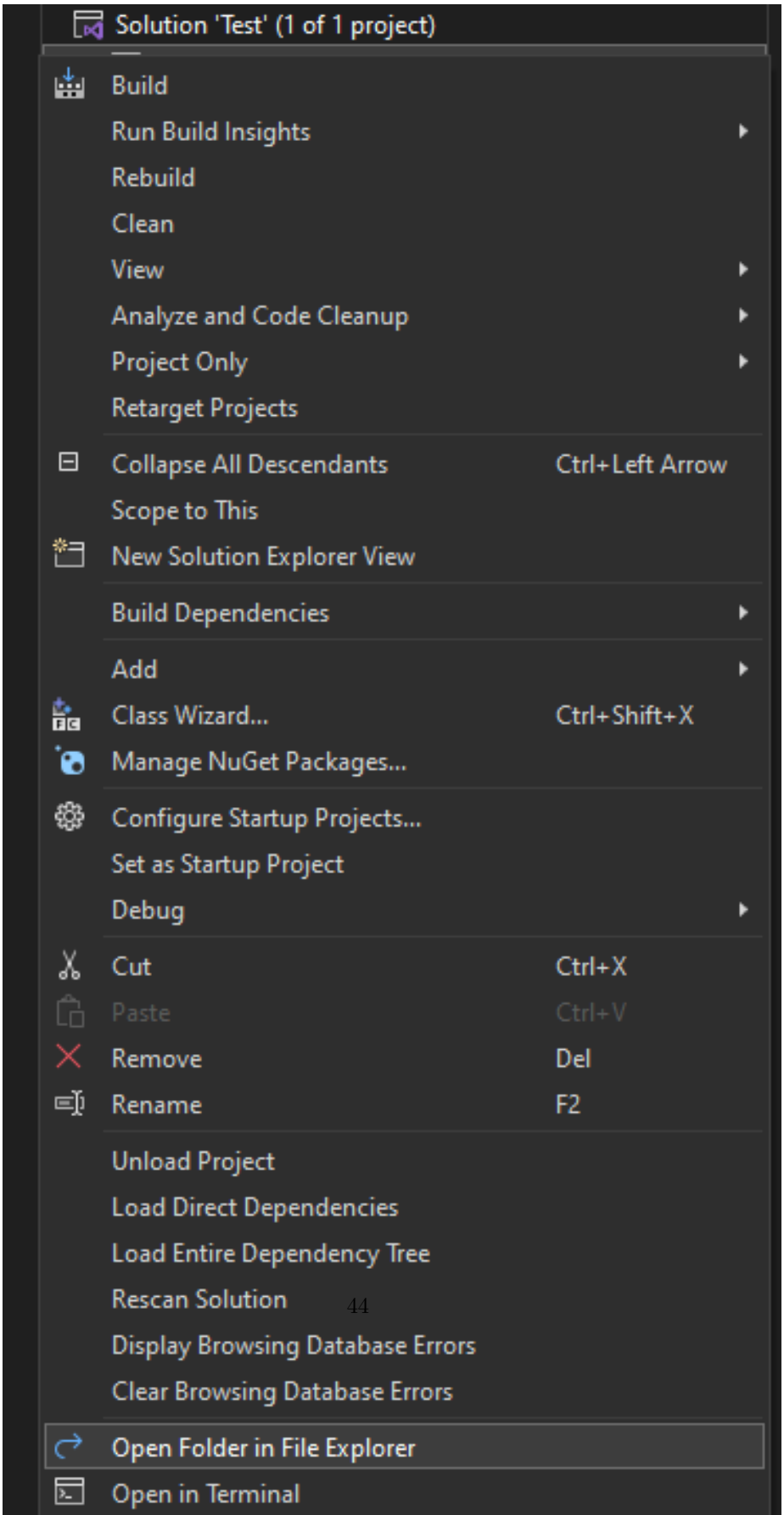
12.1 Configurare Proiect Visual Studio

1. Accesați Moodle și selectați una dintre următoarele:

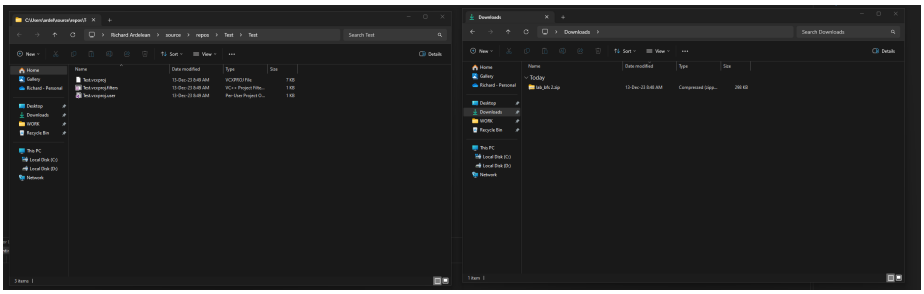


   Aceasta va rezulta în descărcarea unui fișier '.zip'.

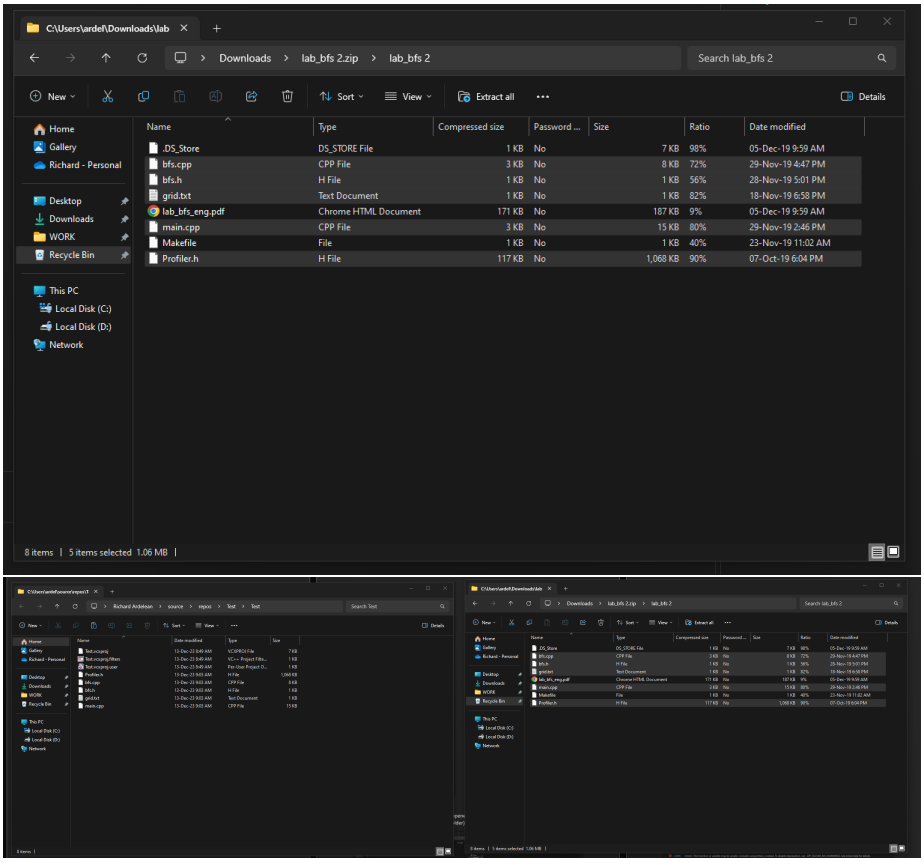
2. Creați un Proiect C++ Gol și făcând clic dreapta pe proiect (nu pe soluție) veți putea selecta 'Deschideți folderul în File Explorer'.


Solution 'Test' (1 of 1 project)
Build
Run Build Insights
Rebuild
Clean
View
Analyze and Code Cleanup
Project Only
Retarget Projects
Collapse All Descendants Ctrl+Left Arrow
Scope to This
New Solution Explorer View
Build Dependencies
Add
Class Wizard... Ctrl+Shift+X
Manage NuGet Packages...
Configure Startup Projects...
Set as Startup Project
Debug
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename F2
Unload Project
Load Direct Dependencies
Load Entire Dependency Tree
Rescan Solution
Display Browsing Database Errors
Clear Browsing Database Errors
Open Folder in File Explorer
Open in Terminal

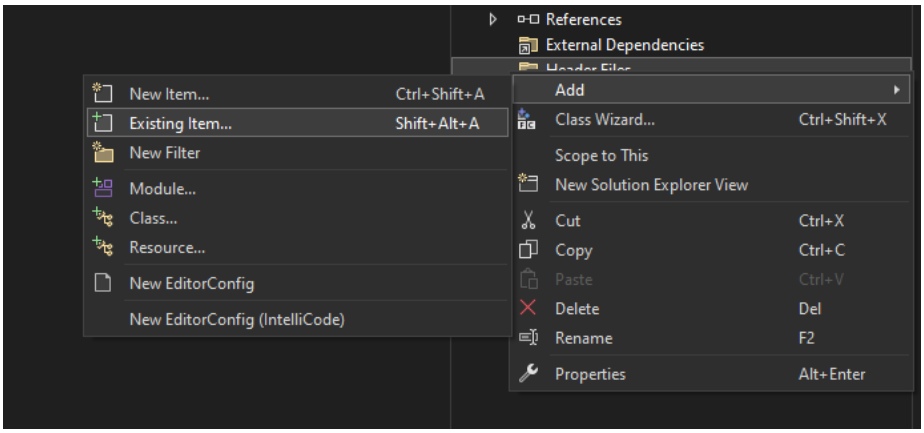
3. În acest moment, se va deschide o fereastră ‘File Explorer’. Deschideți o altă fereastră ‘File Explorer’ la locația descărcărilor dvs. (cel mai probabil folderul Downloads).



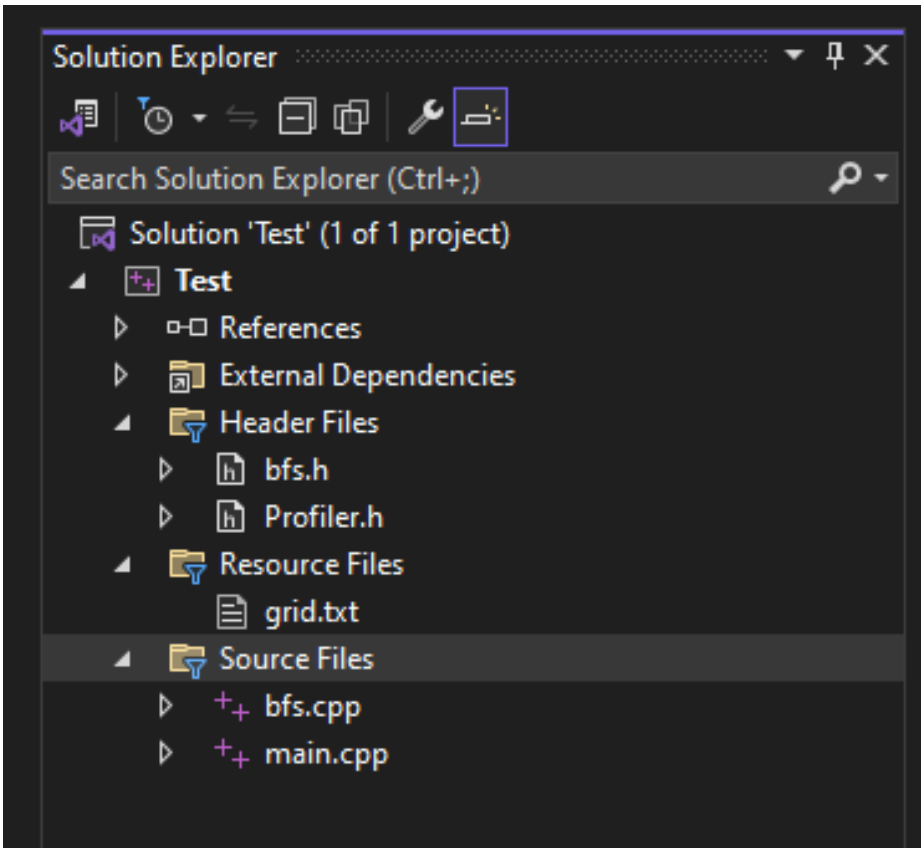
4. Deschideți fișierul ‘.zip’ și copiați, așa cum se arată mai jos, fișierele din arhivă în folderul proiectului.



5. Selectând folderele din Solution Explorer al Visual Studio ‘Fișiere Header / Resurse / Fișiere Sursă‘ utilizați următoarele opțiuni:

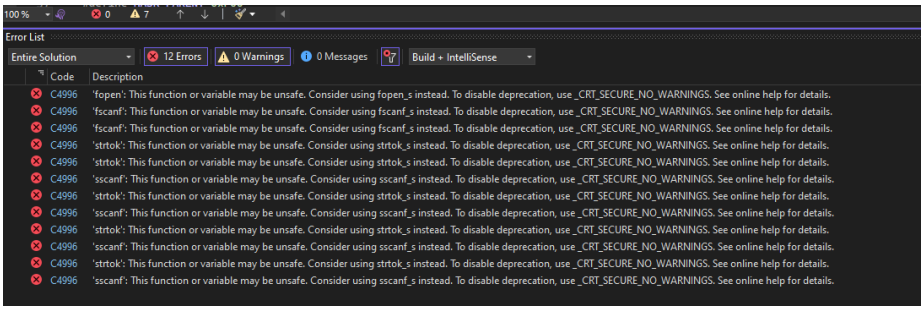


6. Astfel, se realizează următoarea structură:



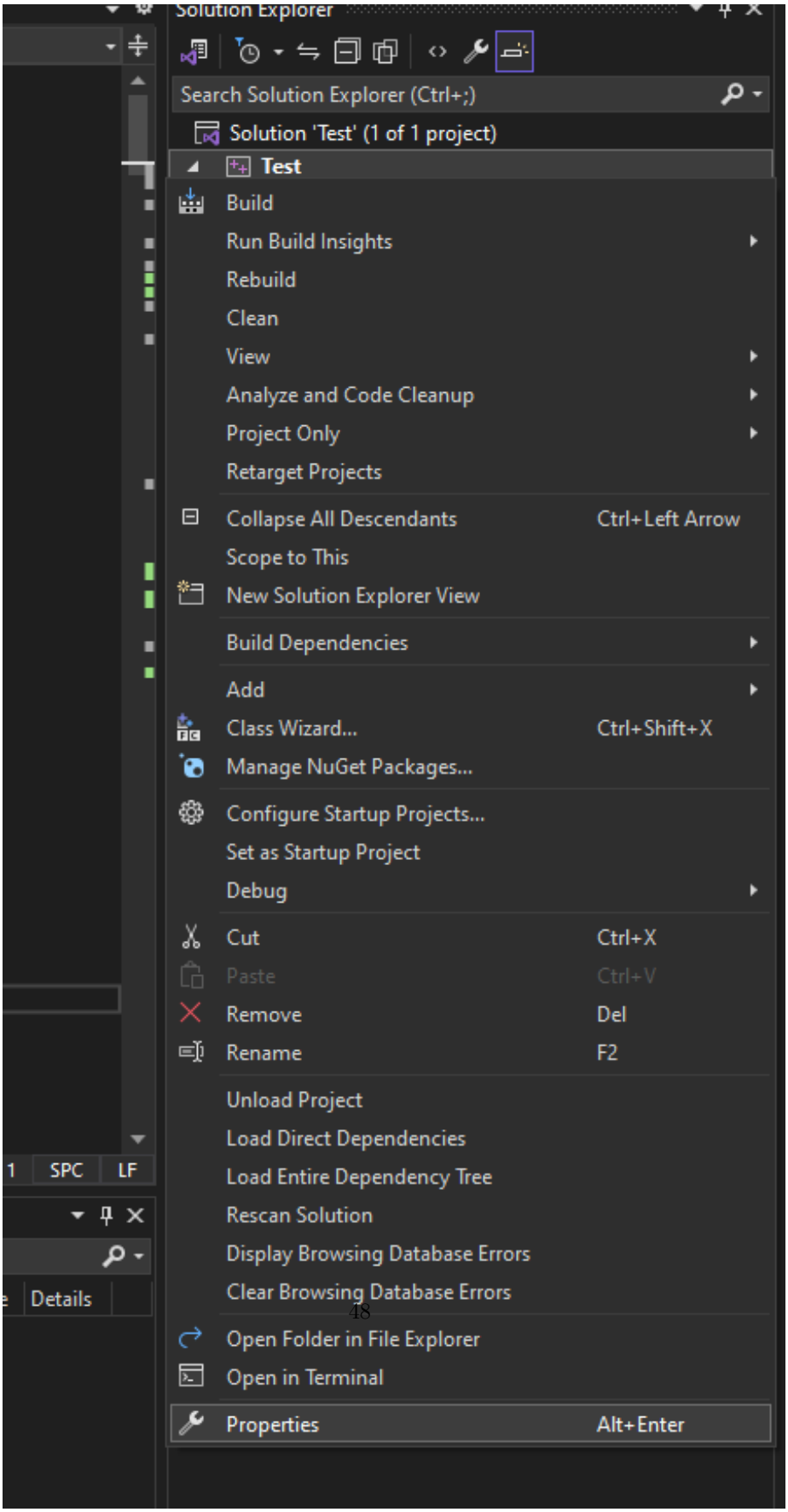
12.2 Eroare 'unsafe' în Visual Studio

Exemplu:



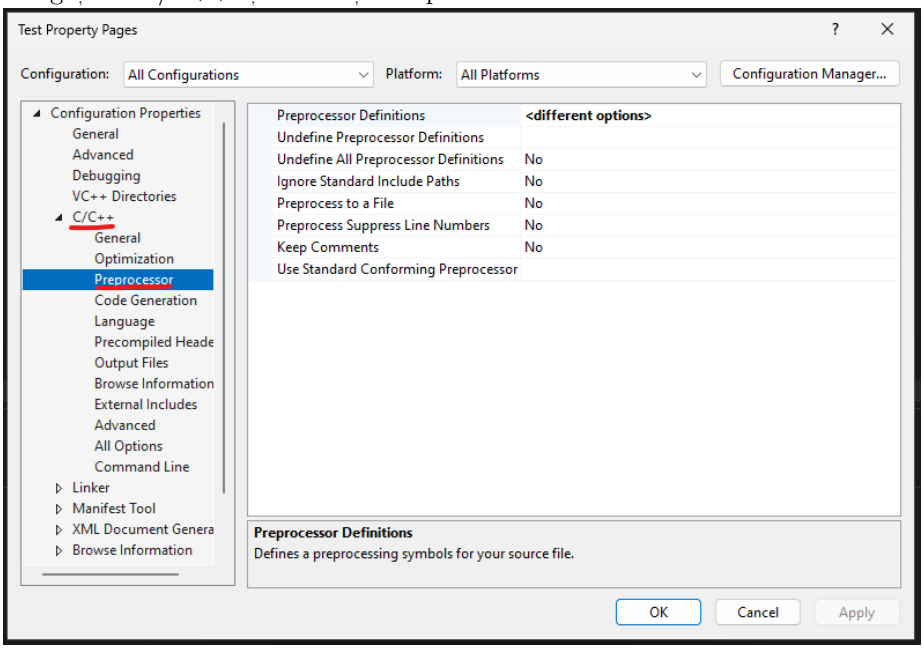
Soluție:

Faceți clic dreapta pe proiect (nu pe Soluție, care cel mai probabil are același nume) și selectați ‘Proprietăți‘.

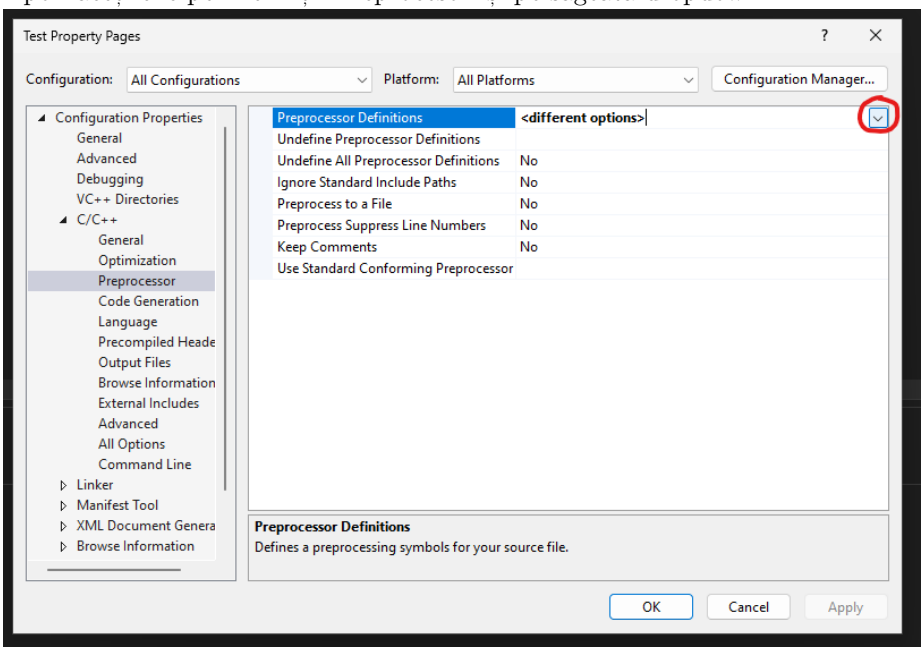

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Test' (1 of 1 project)
Test
Build
Run Build Insights
Rebuild
Clean
View
Analyze and Code Cleanup
Project Only
Retarget Projects
Collapse All Descendants Ctrl+Left Arrow
Scope to This
New Solution Explorer View
Build Dependencies
Add
Class Wizard... Ctrl+Shift+X
Manage NuGet Packages...
Configure Startup Projects...
Set as Startup Project
Debug
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename F2
Unload Project
Load Direct Dependencies
Load Entire Dependency Tree
Rescan Solution
Display Browsing Database Errors
Clear Browsing Database Errors
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter
SPC
LF
Details

Actualizați ‘Configurație‘ și ‘Platformă‘ în partea superioară a ferestrei la ‘Toate configurațiile‘ și ‘Toate platformele‘, respectiv.

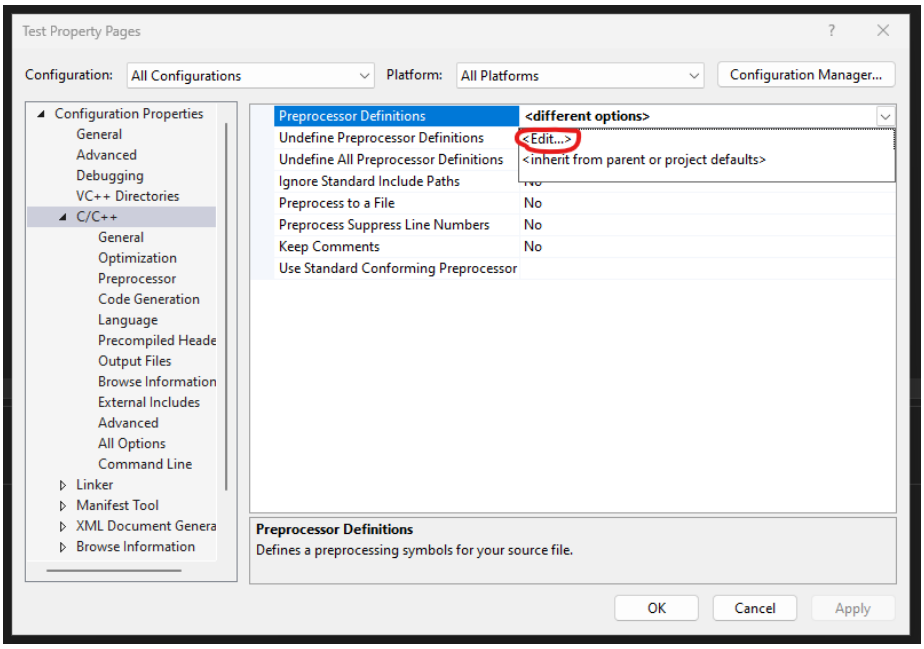
Mergeți la ‘C/C++‘ și selectați ‘Preprocesor‘.



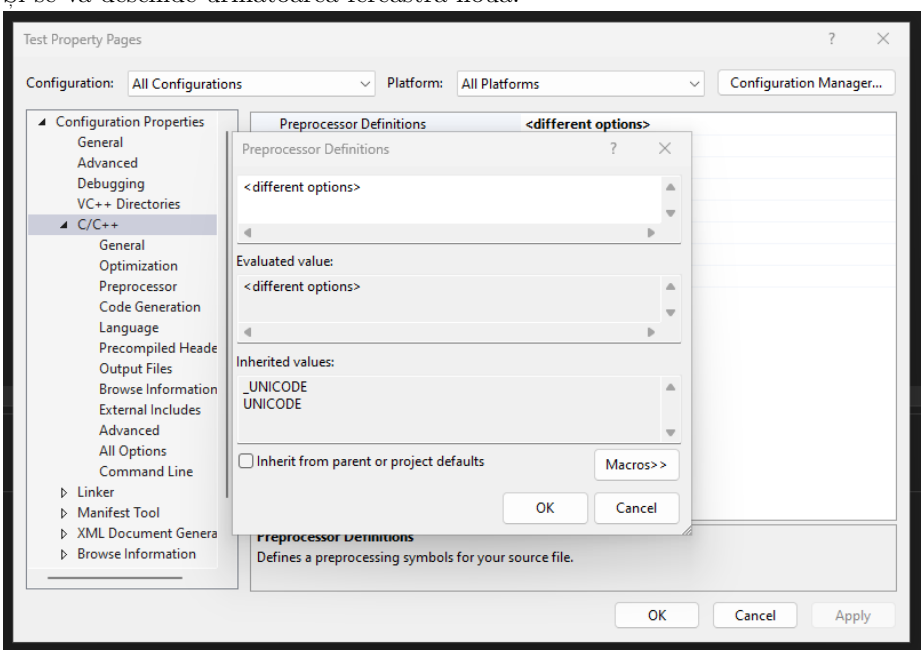
Apoi faceți clic pe ‘Definiții Preprocesor‘ și pe săgeata dropdown.



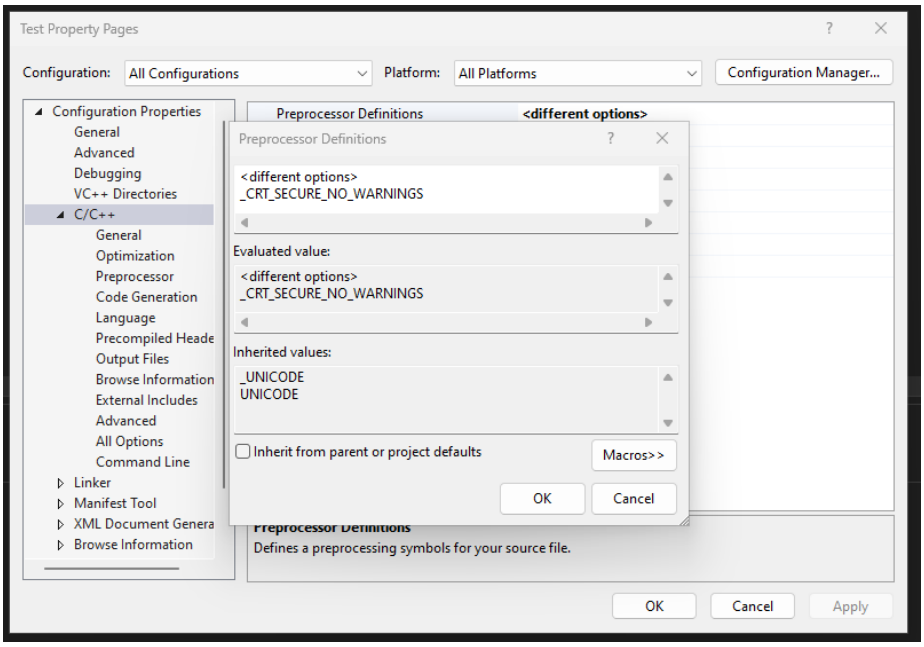
Selectați ‘Editare‘



Și se va deschide următoarea fereastră nouă:



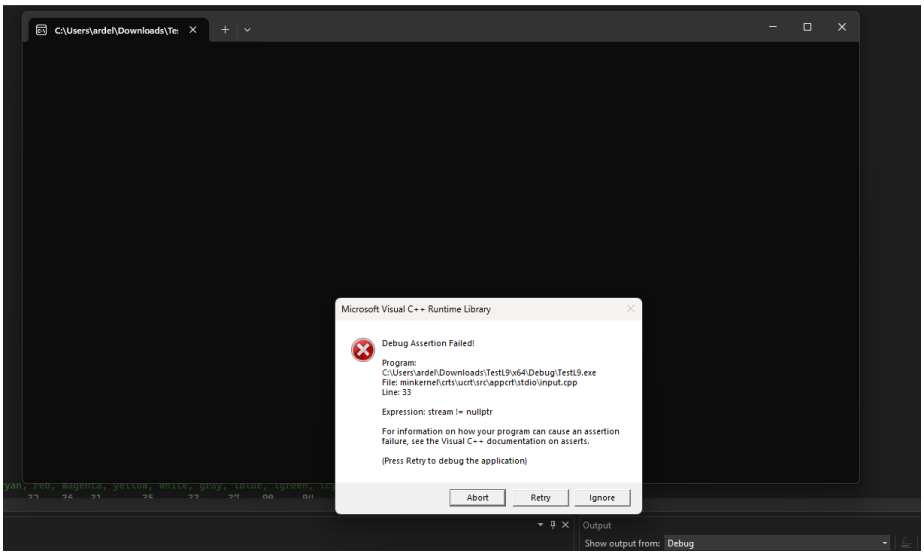
Introduceți '_CRT_SECURE_NO_WARNINGS' sub '<opțiuni diferite>', așa cum se arată mai jos.



Faceți clic pe ‘OK‘ la toate ferestrele deschise până când reveniți la fereastra principală Visual Studio a proiectului și veți putea rula proiectul.

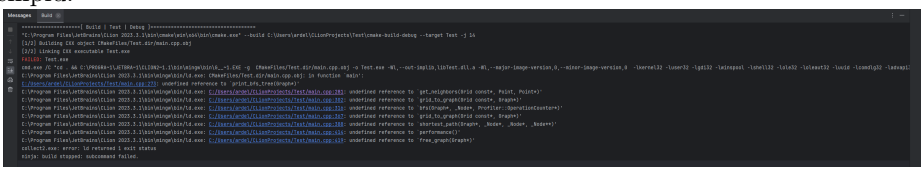
12.3 Eroare ‘Assertion‘ în Visual Studio

Aceasta indică faptul că nu ați urmat tutorialul. Întoarceți-vă la prima pagină și asigurați-vă că ați mutat fișierele din folderul ‘Downloads‘ în folderul ‘Project‘. De asemenea, este posibil să fie nevoie să *ștergeți* toate fișierele din IDE-ul Visual Studio și să le *adăugați din nou* manual.

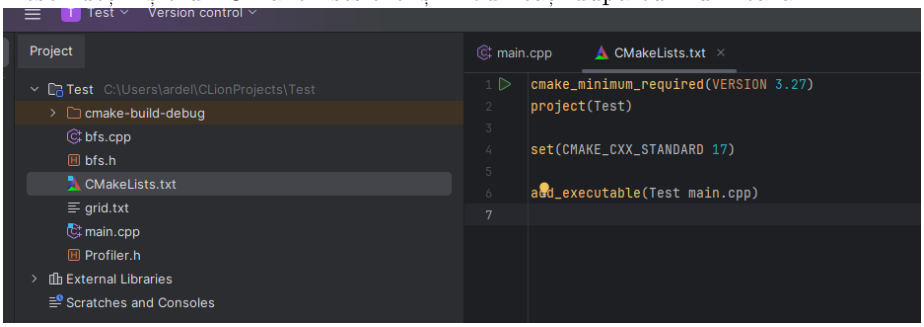


12.4 Eroare undefined în CLion

Exemplu:



Soluție:
Deschideți fișierul ‘CMakeLists.txt‘ și modificați după cum urmează:

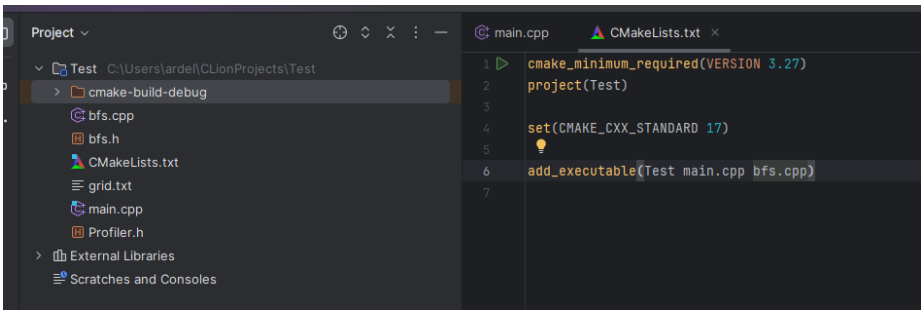


adăugați

```
add_executable(Test main.cpp)
```

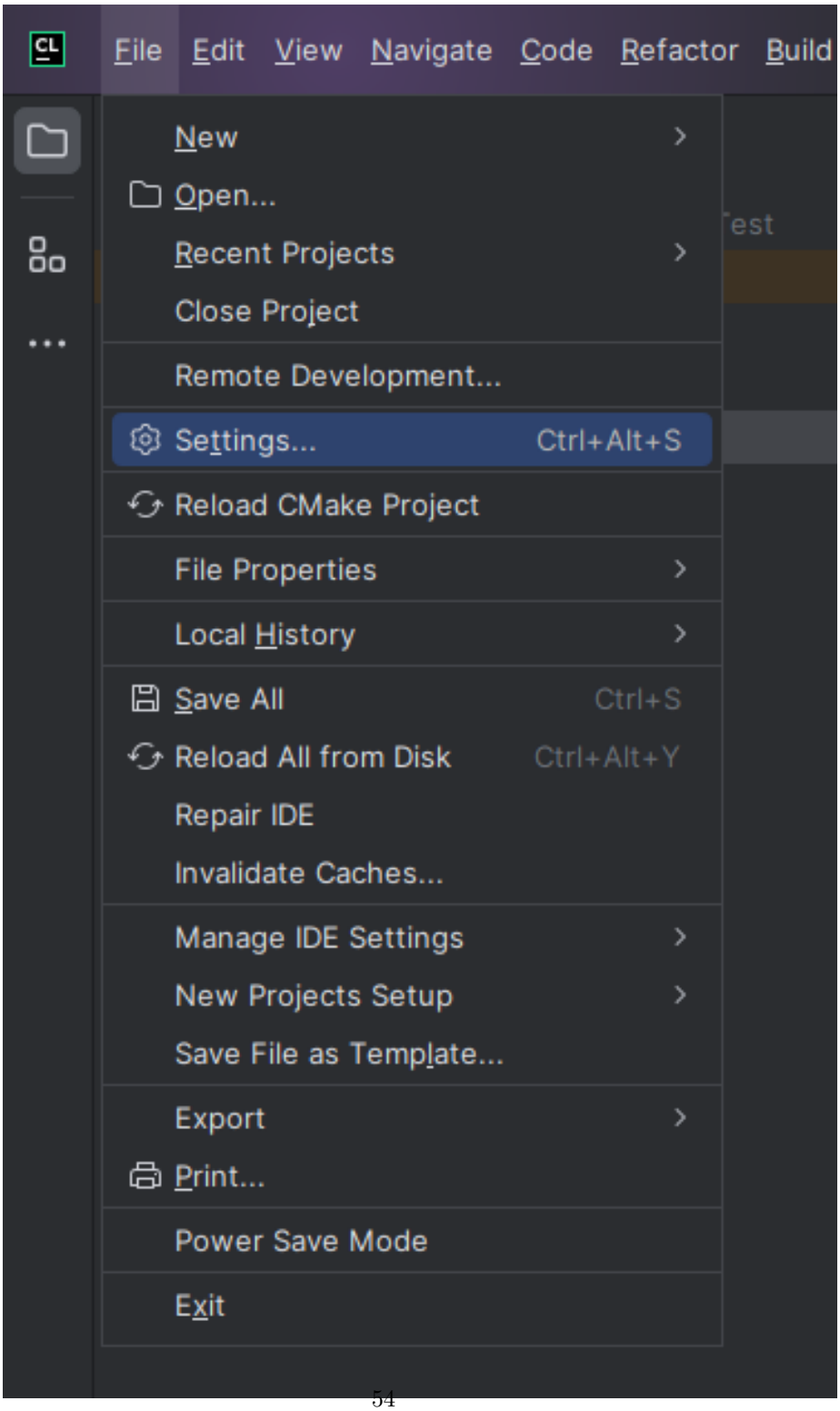
în

```
add_executable(Test main.cpp bfs.cpp)
```

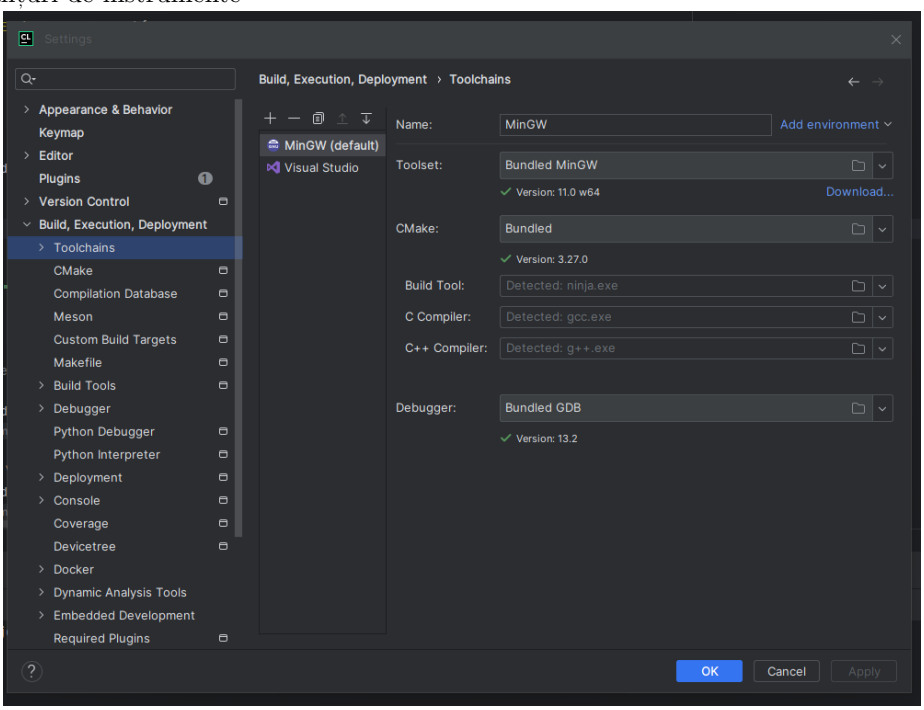


12.5 CLion Visual Studio – Opțiunea 1 (mai lentă)

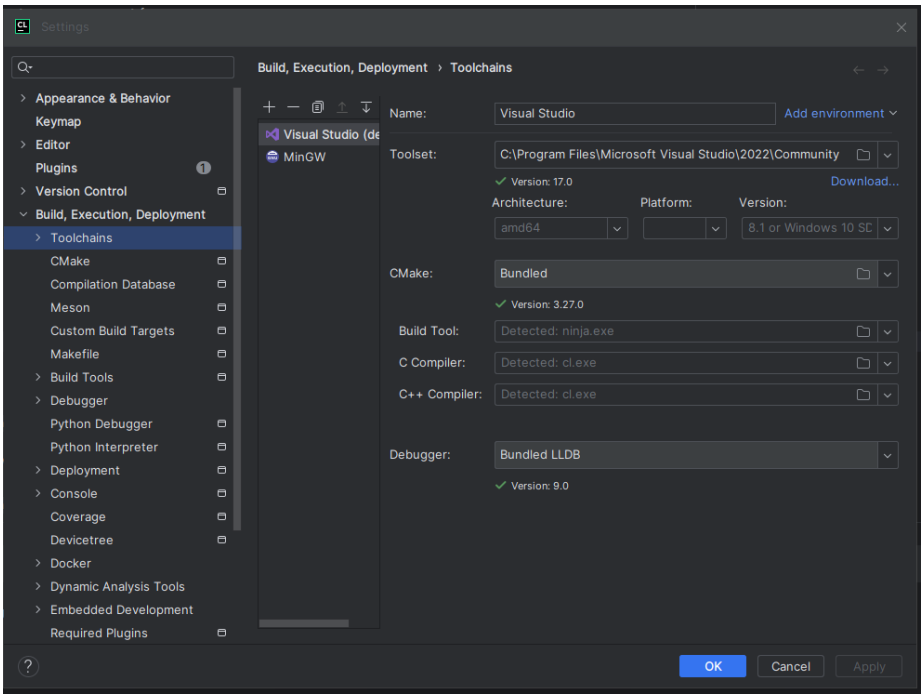
Mergeți la 'Fișier' -¿ 'Setări':


File
Edit
View
Navigate
Code
Refactor
Build
New
Open...
Recent Projects
Close Project
Remote Development...
Settings... Ctrl+Alt+S
Reload CMake Project
File Properties
Local History
Save All Ctrl+S
Reload All from Disk Ctrl+Alt+Y
Repair IDE
Invalidate Caches...
Manage IDE Settings
New Projects Setup
Save File as Template...
Export
Print...
Power Save Mode
Exit

În fereastra nou deschisă, mergeți la ‘Compilare, Execuție, Implementare‘ -¿ ‘Lanțuri de instrumente‘



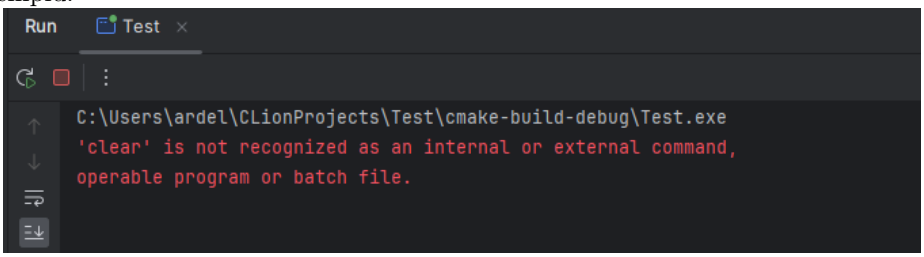
Folosind săgețile, setați Visual Studio ca implicit:



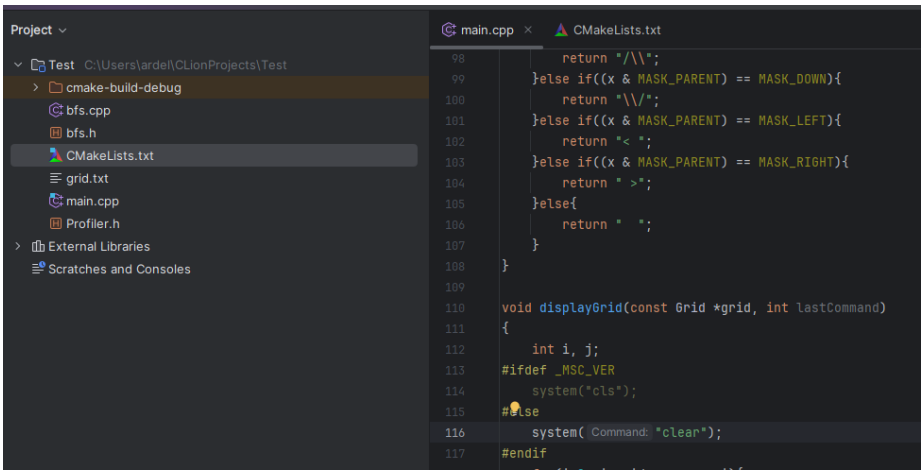
12.6 CLion MinGW – Opțiunea 2 (mai rapidă, necesită consolă externă)

12.6.1 Eroare Clear în CLion

Exemplu:



Soluție:
Accesați fișierul main.cpp și derulați la liniile 113-117 în funcția displayGrid.

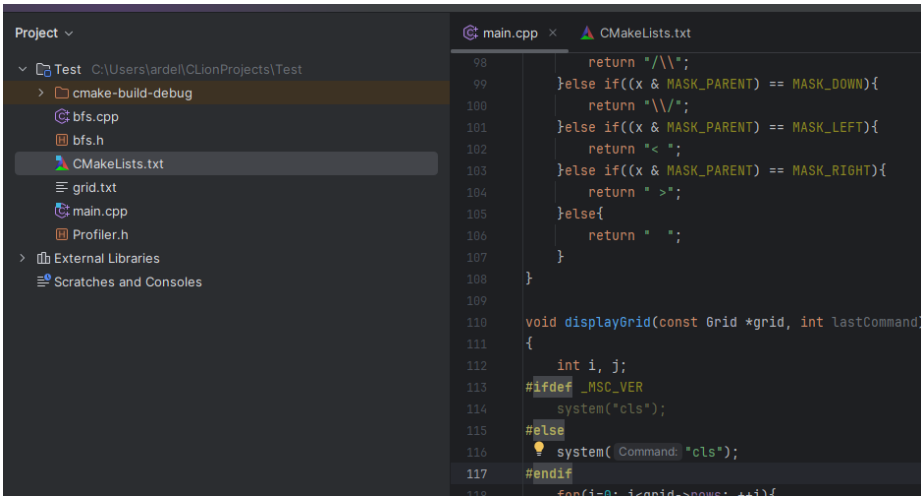


Modificați după cum urmează:
În ramura else de la linia 116

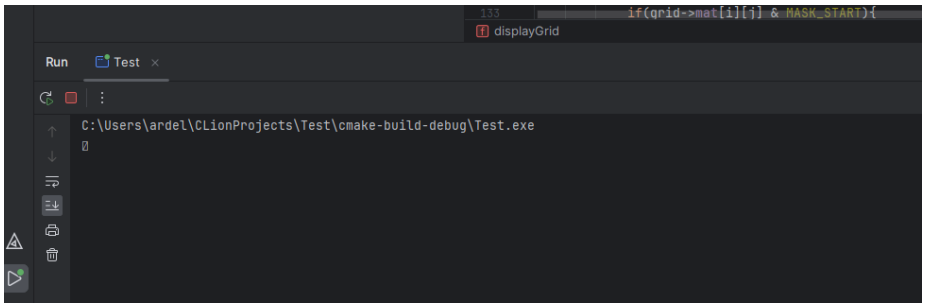
```
system("clear");
```

înlocuiți cu

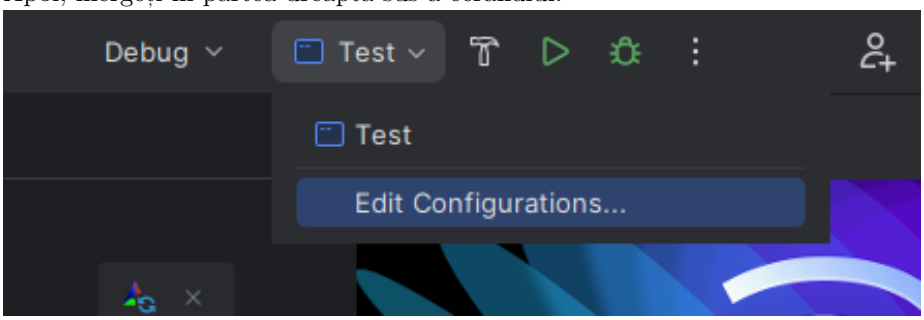
```
system("cls");
```



12.6.2 CLion nu afișează grila

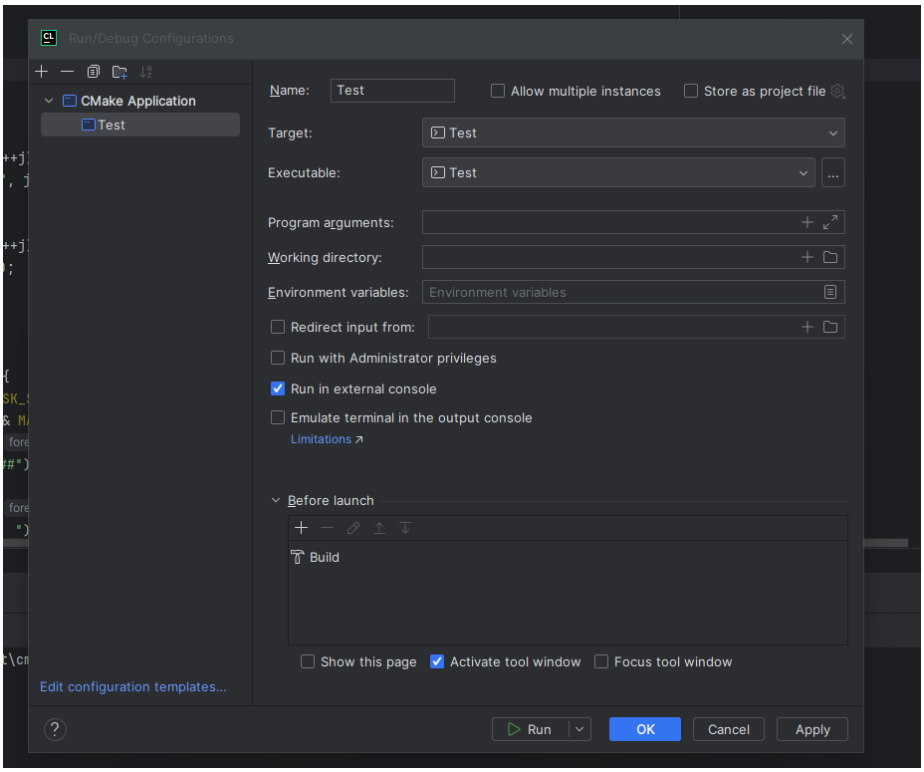


Copiați fișierul ‘grid.txt‘ în folderul ‘cmake-build-debug‘.
Apoi, mergeți în partea dreaptă sus a ecranului:



Selectați ‘Edit Configurations‘ și bifați următoarele opțiuni:

- Rulați în consolă externă



12.7 Comanda de rulare pe Mac

```
g++ main.cpp bfs.cpp -std=c++11 && ./a.out
```

13 Tema Nr. 10: Căutare în adâncime (DFS)

Tema Nr. 10: Căutare în adâncime (DFS)
Timp Alocat: 2 ore

13.1 Implementare

Se cere implementarea corectă și eficientă a algoritmului de căutare în adâncime (Depth-First Search - DFS) (*Capitolul 22.3[1]*). Pentru reprezentarea grafurilor, va trebui să folosești liste de adiacență. De asemenea, va trebui să:

- Implementarea algoritmului Tarjan pentru componente tare conexe
- Implementezi sortarea topologică (*Capitolul 22.4 [1]*)

13.2 Cerințe

13.2.1 DFS (5p)

Demonstrați corectitudinea algoritmului pe un graf de dimensiune mică:

- afișați graful inițial (liste de adiacență)
- afișați arborele rezultat în urma DFS

13.2.2 Sortare topologică (1p)

Demonstrați corectitudinea algoritmului pe un graf de dimensiune mică:

- afișați graful inițial (liste de adiacență)
- afișați listă de noduri sortate topologic (dacă are / dacă nu are de ce nu are?)

13.2.3 Tarjan (2p)

Demonstrați corectitudinea algoritmului pe un graf de dimensiune mică:

- afișați graful inițial (liste de adiacență)
- afișați componentele puternic conexe ale grafului

13.2.4 Analiza performanței pentru DFS (2p)

! Înainte de a începe să lucrați la partea de evaluare, asigurați-vă că aveți un algoritm corect!

Cum timpul de execuție al algoritmului DFS variază în funcție de numărul de vârfuri (|V|) și de numărul de muchii (|E|) aveți de făcut următoarele analize:

1. Fixați |V|=100 și variați |E| între 1000 și 4500 cu un pas de 100. Generați pentru fiecare caz un graf aleator și asigurați-vă că nu generați aceeași muchie de 2 ori. Execută DFS pentru fiecare graf generat și numără operațiile efectuate. Apoi construiește graficul cu variația numărului de operații în funcție de |E|;
2. Fixați |E|=4500 și variați |V| între 100 și 200 cu un pas de 10. Repetă procedura de mai sus și construiește graficul cu variația numărului de operații în funcție de |V|.

14 Rezolvarea erorilor

14.1 Profiler + VS Code error

În fișierul `Profiler.h` ai următoarele linii începând cu linia 11:

```
// OS detection
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
    #define PROFILER_WINDOWS
#elif __APPLE__
    #define PROFILER_OSX
#elif __linux__
    #define PROFILER_LINUX
#endif

#ifdef PROFILER_WINDOWS
    #include <Windows.h>
    #include <Shellapi.h>
#else
    #include <unistd.h>
#endif
```

Trebuie să le modifici în felul următor:

```
// OS detection
#if defined(__MINGW32__) || defined(__CYGWIN__)
    #define PROFILER_VSCODE
#elif defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
    #define PROFILER_WINDOWS
#elif defined(__APPLE__)
    #define PROFILER_OSX
#elif defined(__linux__)
    #define PROFILER_LINUX
#endif

#if defined(PROFILER_WINDOWS) || defined(PROFILER_VSCODE)
    #include <Windows.h>
    #include <Shellapi.h>
#else
    #include <unistd.h>
#endif
```

Mai jos sunt capturile de ecran (în jur de 80% din lățimea textului), încadrate ca figure:

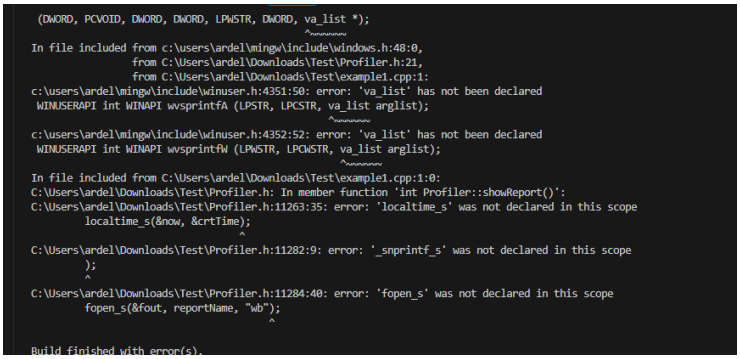


Figure 7: Eroarea inițială în VS Code Profiler

Figure 8: Captura de ecran originală (before fix)

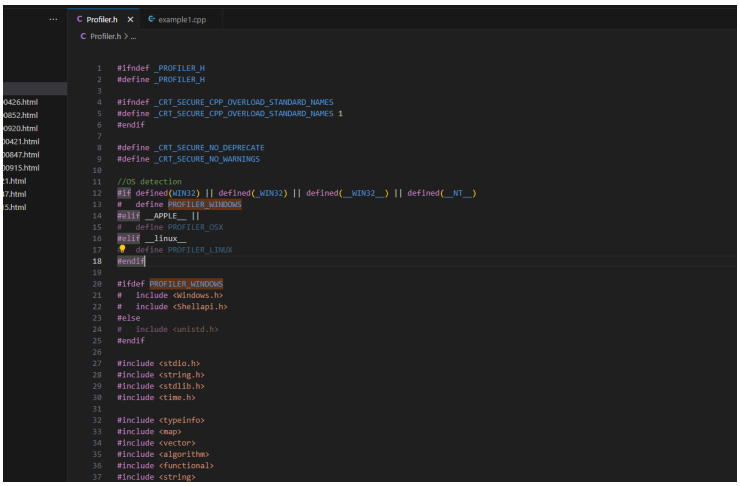


Figure 9: Captura de ecran după actualizare (after fix)

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.