Structuri de Date și Algoritmi (SDA)

As.Drd.Ing. Bianca Gușiță
UPT-AC-TI, AN II

S1 – Laborator introductiv

Organizare

- Toate materialele se găsesc pe Campus Virtual.
- Mediul de lucru folosit va fi Visual Studio + gitlab.upt.ro. Aplicațiile se vor realiza în limbajul C.
- Fiecare lucrare de laborator va fi citită în avans, de pe site-ul laboratorului.
- Soluțiile la problemele de la temele de casa, vor fi realizate individual.
- **Copierea soluțiilor din alte surse (plagiatul) constituie o încălcare a regulamentului UPT și se va sancționa** conform Regulamentului UPT pentru examinare și notare.
- Dacă aveți întrebări, vă rog să îmi scrieți mesaj pe Campus Virtual.
- În timpul laboratorului NU este permisă utilizarea smartphone-ului.

Aplicațiile zilei

- Să se parcurgă tutorialul Visual Studio 2022;
- Să se creeze un proiect cu o aplicație de tip consolă urmărind pașii din tutorial;
- Să se ruleze o aplicație *”Care calculează suma primelor 5 numere naturale”* urmărind pașii pentru compilarea și rularea unei aplicații în Visual Studio 2022;
- Să se parcurgă tutorialul de GIT;
- Să se creeze un proiect test pe gilab.upt.ro in care sa se încarce programul realizat anterior.
- Să ne obișnuim cu Debugging
  - Breakpoint
  - Watch pe variabile
  - Step in/step out

Învățăm/Ne reamintim împreună

- **Debugging**
  - Procesul de identificare și corectare a erorilor (bugs) dintr-un program.
  - Te ajută să găsești și să repari problemele din cod printr-o serie de instrumente care îți permit să:
    - Rulezi codul pas cu pas (step by step) pentru a vedea exact cum se comportă.
    - Pui breakpoints – locuri în cod unde execuția se va opri automat, astfel încât să poți inspecta variabilele și starea programului în acel moment.
    - Verifici valorile variabilelor în timpul execuției pentru a vedea dacă primesc valorile corecte.
    - Monitorizezi fluxul execuției pentru a înțelege mai bine cum se desfășoară programul și unde apare problema.
  - ***Concluzii***: Acest proces te ajută să identifici erori de logică, probleme cu alocarea memoriei sau alte bug-uri care ar putea duce la comportamente neașteptate ale aplicației.

Învățăm/Ne reamintim împreună

Exemplu

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

- **Pasul 1: Rularea codului pas cu pas (Step by Step)**
- După ce codul a fost scris și ai dat build, poți să începi să faci debugging în Visual Studio.
- Pentru a rula codul pas cu pas:
  - Apasă **F10** pentru a executa fiecare linie de cod pe rând.
  - Pe măsură ce execuți codul linie cu linie, Visual Studio îți arată care linie urmează să fie rulată și îți permite să observi exact cum se modifică variabilele la fiecare pas.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

- **Pasul 2: Punerea de breakpoints**
- Un breakpoint este un punct din cod în care execuția se va opri pentru a inspecta starea programului.
  - Poți adăuga un breakpoint dând click în stânga liniei de cod, pe bara gri de lângă numărul liniei. De exemplu, poți pune un breakpoint la linia **sum += i;**.
  - Când rulezi codul (apăsând **F5**), execuția se va opri la acest punct. Astfel, poți verifica valorile variabilelor exact înainte de a se modifica.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

- **Pasul 3: Verificarea valorilor variabilelor**
- Când execuția programului se oprește la un breakpoint, poți verifica valorile variabilelor:
  - În fereastra "Locals" din Visual Studio, poți vedea valorile variabilelor locale, cum ar fi **sum** și **i**.
  - Alternativ, poți să dai hover cu mouse-ul peste o variabilă în cod și Visual Studio îți va afișa valoarea curentă.
- De exemplu, dacă ai pus un breakpoint la linia **sum += i;**, vei putea vedea valoarea lui **sum** și **i** înainte și după această linie.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

- **Pasul 4: Monitorizarea fluxului execuției**
- În timpul debuggingului, poți să observi și să înțelegi mai bine fluxul execuției:
  - Folosind **F10** pentru a merge de la o linie la alta, poți vedea cum se repetă bucla **for** și cum se schimbă valorile lui **i** și **sum** la fiecare iterație.
  - Dacă fluxul execuției nu este cel așteptat, de exemplu dacă **sum** nu se acumulează corect, poți identifica unde apare problema și poți ajusta codul.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Acesta este un exemplu de bază, dar acești pași sunt esențiali în debugging. Ei te ajută să înțelegi și să controlezi execuția codului, pentru a găsi și rezolva problemele mai ușor.

Învățăm/Ne reamintim împreună

- În Visual Studio 2022, watch-ul pe variabile îți permite să monitorizezi valorile acestora în timp real, pe măsură ce rulezi codul, chiar și atunci când nu sunt vizibile în fereastra "Locals".
- Iată cum poți să pui un watch pe variabile în timpul debuggingului:

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

- **Pașii pentru a pune watch pe variabile:**

  - Pornește sesiunea de debugging
  - Accesează fereastra "Watch"
  - Adaugă variabile în Watch
  - Monitorizează variabilele

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

Pașii pentru a pune watch pe variabile:

- **1. Pornește sesiunea de debugging:**
  - Apasă F5 sau mergi la meniul Debug și selectează Start Debugging pentru a porni sesiunea de debugging.
  - Poți pune breakpoints pentru a opri execuția la anumite puncte.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

Pașii pentru a pune watch pe variabile:

- **2. Accesează fereastra "Watch":**
  - În timpul debuggingului, mergi la meniul Debug și apoi selectează Windows → Watch → Watch 1 (sau alte ferestre Watch, dacă ai nevoie de mai multe).
  - Va apărea fereastra "Watch", de obicei în partea de jos sau în dreapta interfeței.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

Pașii pentru a pune watch pe variabile:

- **3. Adaugă variabile în Watch:**
  - În momentul în care execuția este oprită (de exemplu, la un breakpoint), scrie numele variabilei pe care vrei să o urmărești în fereastra "Watch".
  - De exemplu, scrie **sum** sau **i** dacă vrei să monitorizezi valorile acestora din bucla **for**.
  - Apasă *Enter*, iar Visual Studio va afișa valoarea curentă a variabilei.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

Pașii pentru a pune watch pe variabile:

- **4. Monitorizează variabilele:**
  - Pe măsură ce rulezi codul pas cu pas (cu **F10**) sau continui execuția (cu **F5**), vei putea vedea cum se modifică valoarea variabilei în fereastra "Watch" în timp real.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

De ce e utilă fereastra "Watch“?

Monitorizare detaliată: Poți urmări variabile sau chiar expresii complexe (ex. **sum + i**).

Folositoare pentru variabile care nu sunt vizibile în fereastra "Locals", mai ales când variabilele sunt locale unei funcții care nu este în acel moment în execuție.

Expresii personalizate: Poți introduce orice expresie validă în C, nu doar variabile simple, iar Visual Studio îți va arăta rezultatul.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

Exemplu practic:

Dacă vrei să monitorizezi variabila **sum** din exemplul anterior, poți pune un breakpoint pe linia **printf** și apoi adaugi **sum** în fereastra "*Watch*".

Vei vedea cum valoarea variabilei **sum** crește după fiecare iterație a buclei **for**.

Acest proces te ajută să urmărești în detaliu evoluția valorilor și să identifici rapid eventualele probleme.

```c
#include <stdio.h>

int main() {
    int sum = 0;
    for (int i = 1; i <= 5; i++) {
        sum += i;
    }
    printf("Suma primelor 5 numere este: %d\n", sum);
    return 0;
}
```

Învățăm/Ne reamintim împreună

În Visual Studio 2022, **Step In (F11)** și **Step Out (Shift + F11)** sunt funcții esențiale pentru a controla execuția codului în timpul debuggingului. Iată cum le poți folosi:

- 1. **Step In (F11)**
  - **Ce face?**
    - Execută următoarea linie de cod și, dacă acea linie conține un apel de funcție, va intra în acea funcție și va începe să ruleze codul linie cu linie din interiorul funcției.
  - **Când să folosești?**
    - Utilizezi **Step In** atunci când vrei să vezi în detaliu ce se întâmplă într-o funcție. Este util mai ales când suspectezi că o funcție nu funcționează corect sau vrei să vezi cum se modifică variabilele interne.
  - **Cum să folosești?**
    - În timpul debuggingului, dacă execuția se află pe o linie de cod care apelează o funcție, apasă **F11**.
    - Visual Studio va intra în funcția respectivă și va începe să execute codul din funcție linie cu linie.

Învățăm/Ne reamintim împreună

- **Step In (F11)**
- Exemplu
  - Dacă te afli pe linia

  int result = suma(3, 5); și apeși **F11**, execuția va intra în funcția **suma**, astfel încât să vezi cum se calculează suma lui **a** și **b**.

```c
int suma(int a, int b) {
    return a + b;
}

int main() {
    int result = suma(3, 5);
    printf("Rezultatul este: %d\n", result);
    return 0;
}
```

Învățăm/Ne reamintim împreună

În Visual Studio 2022, **Step In (F11)** și **Step Out (Shift + F11)** sunt funcții esențiale pentru a controla execuția codului în timpul debuggingului. Iată cum le poți folosi:

- 2. **Step Out (Shift + F11)**
  - **Ce face?**
    - Continuă execuția unei funcții până la final și revine în funcția apelantă, adică la locul de unde a fost apelată funcția. Este util dacă ai intrat într-o funcție, dar nu mai vrei să urmărești fiecare linie din acea funcție și vrei să revii la codul care a apelat-o.
  - **Când să folosești?**
    - Folosește **Step Out** când ești în interiorul unei funcții, dar ai terminat analiza și vrei să ieși din funcție rapid, pentru a continua execuția codului apelant.
  - **Cum să folosești?**
    - Dacă te afli în interiorul unei funcții și apeși **Shift + F11**, Visual Studio va continua execuția până la finalul funcției curente și va reveni la codul care a apelat funcția.

Învățăm/Ne reamintim împreună

- **Step In (F11)**
- Exemplu
  - Dacă ai intrat în funcția **suma** folosind **Step In** și apoi apeși **Shift + F11**, execuția va termina funcția **suma** și te va readuce la **linia int result = suma(3, 5);** din **main**.

```c
int suma(int a, int b) {
    return a + b;
}

int main() {
    int result = suma(3, 5);
    printf("Rezultatul este: %d\n", result);
    return 0;
}
```

Învățăm/Ne reamintim împreună

Recapitulare:

- **Step In (F11):**
  - Intră în funcția apelată și execută linie cu linie din interiorul ei.
- **Step Out (Shift + F11):**
  - Iese din funcția curentă și revine la funcția care a apelat-o.

Aceste funcții sunt foarte utile pentru a înțelege mai bine fluxul programului și pentru a localiza erori în funcții complexe.

Aplicațiile zilei

- Să se parcurgă tutorialul Visual Studio 2022 de pe Campus Virtual;
- Să se creeze un proiect cu o aplicație de tip consolă urmărind pașii din tutorial;
- Să se ruleze o aplicație *”Care calculează suma primelor 5 numere naturale”* urmărind pașii pentru compilarea și rularea unei aplicații în Visual Studio 2022;
- Să se parcurgă tutorialul de GIT de pe Campus Virtual;
- Să se creeze un proiect test pe gilab.upt.ro în care să se încarce aplicația realizată anterior.