Clase

Observații:

- Documentul este gândit pentru studenții care au participat la seminar. Este posibil sa întâmpinați dificultăți in parcurgerea / înțelegerea lui daca nu ați fost prezenți.
- Eventualele dezacorduri sau greșeli de scriere au fost făcute intenționat, cu rolul de a vă binedispune în timpul citirii acestui document. ☺

Declarare

1. Declarați clasa **Student** cu următoarele atribute: **nume** (char*), **varsta** (int), **note** (int*), **nrNote** (int). Câmpurile vor fi declarate în zona publică

```
class Student
{
public:
	char* nume;
	int varsta;
	int nrNote;
	int *note;
};
```

2. În metoda **main** se va declara o variabilă de tipul **Student**

```
void main()
{
	Student s;
}
```

3. Declarați metoda citesteStudent (Student st) in exteriorul clasei. Metoda asigură într-o primă etapă doar preluarea vârstei de la tastatură.

```
//implementare gresita
void citesteStudent(Student st)
{
	cout << "Varsta: ";
	cin >> st.varsta;
}
```

4. Apelați metoda citesteStudent din metoda main. Adăugați codul necesar pentru afișarea varstei studentului.

```
void main()
```

```
{
	Student s;
	s.varsta = 0;
	citesteStudent(s);
	cout << "Varsta: " << s.varsta;
}
```

5. Rulați programul. Observați valoarea afișată pentru vârstă. Analizați din ce motiv vârsta afișată este incorectă.
6. Rulând programul in mod Debug adăugați în fereastra Watch adresele variabilelor **s** (metoda main) și **st** (metoda citeste student). Observați și conținutul celor două variabile.
7. Vârsta citită în metoda citesteStudent diferă de vârsta afișată datorită transmiterii prin valoare a parametrului metodei.
8. Modificați metoda citesteStudent astfel încât transmiterea parametrului să se facă prin pointer.

```
void citesteStudent(Student *st)
{
	cout << "Varsta: ";
	cin >> (*st).varsta; //echivalent cu st->varsta
}
```

9. Actualizați apelul metodei citesteStudent in cadrul metodei main.

```
void main()
{
	Student s;
	s.varsta = 0;
	citesteStudent(&s);
	cout << "Varsta: " << s.varsta;
}
```

10. Verificați citirea și afișarea corectă a vârstei studentului.
11. Modificați metoda citesteStudent pentru a prelua de la tastură și restul de date pentru student

```
//implementare gresita
void citesteStudent(Student *st)
{
	//1. Citire varsta
	cout << "Varsta: ";
	cin >> (*st).varsta; //echivalent cu st->varsta

	//2. Citire nume
	cout << "Nume:";
	//v1. gresita
	//cin >> st->nume;
	//v2.
	char buf[200];
	cin >> buf;

	(*st).nume = new char[strlen(buf) + 1];
	strcpy((*st).nume, buf);

	//3. Citire nrNote
	cout << "NrNote: ";
	cin >> st->nrNote;

	//4. Citire note
	//v1. gresita
	//cin >> st->note;
	//v2.
	st->note = new int[st->nrNote];
```

```
	for (int i = 0; i < st->nrNote; i++)
	{
		cout << "nota[" << i << "]: ";
		cin >> st->note[i];
	}
}
```

12. Actualizati codul din metoda main pentru a asigura afisarea datelor studentului

```
void main()
{
	Student s;
	s.nume = NULL;
	s.varsta = 0;
	s.nrNote = 0;
	s.note = NULL;

	citesteStudent(&s);
	cout << "Nume: " << s.nume<<endl;
	cout << "Varsta: " << s.varsta<<endl;
	for (int i = 0; i < s.nrNote; i++)
		cout << s.note[i];
}
```

13. Codul scris anterior genereaza memoryleak-uri in cazul in care apelam de doua ori metoda citesteStudent . Analizati din ce cauza.
14. Modificati metoda citesteStudent astfel incat sa fie asigurata dezalocarea spatiului de memorie alocat in HEAP

```
void citesteStudent(Student *st)
{
	//1. Citire varsta
	cout << "Varsta: ";
	cin >> (*st).varsta; //echivalent cu st->varsta

	//2. Citire nume
	cout << "Nume:";
	//v1. gresita
	//cin >> st->nume;
	//v2.
	char buf[200];
	cin >> buf;
	//a) dezalocare spatiu (doar daca exista spatiu alocat)
	if (st->nume != NULL)
		delete[] st->nume;
	//b) alocare spatiu pentru noul nume
	(*st).nume = new char[strlen(buf) + 1];
	strcpy((*st).nume, buf);

	//3. Citire nrNote
	cout << "NrNote: ";
	cin >> st->nrNote;

	//4. Citire note
	//v1. gresita
	//cin >> st->note;
	//v2.
	//a) dezalocare spatiu (doar daca exista spatiu alocat)
	if (st->note != NULL)
		delete[] st->note;
	//b) alocare spatiu pentru noul vector de note
	st->note = new int[st->nrNote];
	for (int i = 0; i < st->nrNote; i++)
```

```
	{
		cout << "nota[" << i << "]: ";
		cin >> st->note[i];
	}
}
```

15. Să presupunem că efectuăm citirea datelor asociate studenților într-o buclă for ca mai jos.

```
void main()
{
	for(int i=0; i<100; i++)
	{
		Student s;
		s.nume = NULL;
		s.varsta = 0;
		s.nrNote = 0;
		s.note = NULL;

		citesteStudent(&s);
		cout << "Nume: " << s.nume << endl;
		cout << "Varsta: " << s.varsta << endl;
		for (int i = 0; i < s.nrNote; i++)
			cout << s.note[i];
	}

	//alte instructiuni
}
```

16. Programul scris generaza memory leak-uri. Câte zone de memorie vor exista alocate în HEAP dar nereferite, în momentul în car execuția programului părăsește bucla for?
17. Adăugați codul necesar pentru dealocarea zonelor de memorie.

```
void main()
{
	for(int i=0; i<100; i++)
	{
		Student s;
		s.nume = NULL;
		s.varsta = 0;
		s.nrNote = 0;
		s.note = NULL;

		citesteStudent(&s);
		cout << "Nume: " << s.nume << endl;
		cout << "Varsta: " << s.varsta << endl;
		for (int i = 0; i < s.nrNote; i++)
			cout << s.note[i];
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (s.nume != NULL)
			delete[] s.nume;
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (s.note != NULL)
			delete[] s.note;
	}

	//alte instructiuni
}
```

Constructor fără parametrii, Destructor

18. O posibilă problemă în codul scris anterior este reprezentată de faptul că ori de câte ori declarăm o variabilă de tip Student trebuie să repetăm inițializările următoare (si evident sa riscam sa le uitam).

```
Student s;
s.nume = NULL;
s.varsta = 0;
s.nrNote = 0;
s.note = NULL;
Student s2;
s2.nume = NULL;
s2.varsta = 0;
s2.nrNote = 0;
s2.note = NULL;
```

19. C++ oferă un mecanism standard pentru inițializarea obiectelor, reprezentat de constructori. Declarați un constructor fără parametrii în clasa Student.

```
class Student
{
public:
	char* nume;
	int varsta;
	int nrNote;
	int *note;
	Student()
	{
		nume = NULL;
		varsta = 0;
		nrNote = 0;
		note = NULL;
	}
};
```

20. Actualizați codul din metoda main.

```
void main()
{
	Student s;
	citesteStudent(&s);
	cout << "Nume: " << s.nume << endl;
	cout << "Varsta: " << s.varsta << endl;
	for (int i = 0; i < s.nrNote; i++)
		cout << s.note[i];
	//dezalocare spatiu (doar daca exista spatiu alocat)
	if (s.nume != NULL)
		delete[] s.nume;
	//dezalocare spatiu (doar daca exista spatiu alocat)
	if (s.note != NULL)
		delete[] s.note;
}
```

21. Similar, codul pentru dealocarea spatiului alocat pentru nume si note, trebuie repetat pentru fiecare variabila declarata. C++ ofera un mecanism standard pentru realizarea de dealocari, reprezentat de destructori. Declarați un destructor în cadrul clasei Student.

```
class Student
{
public:
	char* nume;
	int varsta;
	int nrNote;
```

```
	int *note;
	Student()
	{
		nume = NULL;
		varsta = 0;
		nrNote = 0;
		note = NULL;
	}
	~Student()
	{
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (nume != NULL)
			delete[] nume;
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (note != NULL)
			delete[] note;
	}
};
```

22. Actualizați codul din metoda main.

```
void main()
{
	Student s;
	citesteStudent(&s);
	cout << "Nume: " << s.nume << endl;
	cout << "Varsta: " << s.varsta << endl;
	for (int i = 0; i < s.nrNote; i++)
		cout << s.note[i];
}
```

Codul sursa complet este disponibil pe http://online.ase.ro – “StudentClass - Constructor, Destructor”

Metode de acces

Execriţiul se bazează pe codul scris la pașii anterior, disponibil pe http://online.ase.ro – “StudentClass - Constructor, Destructor”

23. Atributele din cadrul unei clase se declară în zona privată a acesteia. Accesul din exteriorul clasei la aceste atribute se va face prin intermediul unor metode de tip get/set. Mutați câmpul vârstă in zona privata si adăugați metodele necesare.

```
class Student
{
private:
	int varsta;
public:
	char* nume;
	int nrNote;
	int *note;
	Student()
	{
		nume = NULL;
		varsta = 0;
		nrNote = 0;
		note = NULL;
```

```
	}
	~Student()
	{
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (nume != NULL)
			delete[] nume;
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (note != NULL)
			delete[] note;
	}
	int getVarsta()
	{
		return varsta;
	}
	void setVarsta(int varstaNoua)
	{
		varsta = varstaNoua;
	}
};
```

24. Compilați programul și depanați erorile apărute în urma mutării atributului vârsta în zona privată.
25. Utilitatea metodelor de acces constă în posibilitatea de a valida valorile atributelor înainte de a face atribuirea. Validați vârsta primită ca parametru după cum urmează.

```
void setVarsta(int varstaNoua)
{
	if (varstaNoua >= 0 && varstaNoua < 150)
		varsta = varstaNoua;
	else
		throw new exception("Varsta intre 0 si 150!");
}
```

26. Actualizati codul din metoda main ca mai jos.

```
void main()
{
	Student s;
	s.setVarsta(21);
	cout << "Varsta: " << s.getVarsta()<<endl;
}
```

27. Incercati sa setati varsta cu valoarea 2000 (in loc de 21). Programul va arunca o eroare. Puteti citi mai multe despre exceptii si tratarea lor la adresa: http://www.cplusplus.com/doc/tutorial/exceptions/ .

```
void main()
{
	Student s;
	try {
		s.setVarsta(2000);
		cout << "Varsta: " << s.getVarsta() << endl;
	}
	catch (exception* e)
	{
		cout << e->what();
	}
	//catch(...)
}
```

28. Mutați și celelalte atribute din clasă în zona privată. Compilați programul și depanați erorile apărute în urma mutării atributelor în zona privată

```
class Student
{
private:
	int varsta;
	char* nume;
	int nrNote;
	int *note;
public:
	Student()
	{
		nume = NULL;
		varsta = 0;
		nrNote = 0;
		note = NULL;
	}
	~Student()
	{
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (nume != NULL)
			delete[] nume;
		//dezalocare spatiu (doar daca exista spatiu alocat)
		if (note != NULL)
			delete[] note;
	}

	//Get/Set Varsta
	int getVarsta()
	{
		return varsta;
	}
	void setVarsta(int varstaNoua)
	{
		if (varstaNoua >= 0 && varstaNoua < 150)
			varsta = varstaNoua;
		else
			throw new exception("Varsta intre 0 si 150!");
	}

	//Get/Set Nume
	char* getNume() {
		return nume;
	}
	void setNume(char* numeNou) {
		if (numeNou == NULL)
			throw new exception("Nume nu poate fi NULL!");
		else
		{
			if (this->nume != NULL)
				delete[] this->nume;
			this->nume = new char[strlen(numeNou) + 1];
			strcpy(this->nume, numeNou);
		}
	}

	//Get/Set Note & NrNote
	int GetNrNote()
	{
		return nrNote;
	}
	int* GetNote()
	{
		return note;
	}
```

```
	void SetNote(int nrNote, int*note)
	{
		if (this->note != NULL)
			delete[] this->note;
		this->note = new int[nrNote];
		for (int i = 0; i < nrNote; i++)
			this->note[i] = note[i];
	}
};
```

C++ Codul sursa complet este disponibil pe http://online.ase.ro – "StudentClass - Metode de acces"

Constructor cu parametrii, Pointer this

C++ Execriţiul se bazează pe codul scris la paşii anterior, disponibil pe http://online.ase.ro – "StudentClass - Metode de acces"

29. In forma actuala a programului pentru declararea si iniţializarea cu valori a unui obiect de tip Student este necesar un cod similar cu cel de mai jos.

```
void main()
{
	Student s;
	s.setVarsta(2000);
	s.setNume(...)
	s.SetSetNote(...)
}
```

C++ ofera un mecanism standard pentru initializarea obiectelor la declarare, numit constructor cu parametrii. Adăugaţi un constructor cu parametrii dupa modelul de mai jos. Remarcaţi utilizarea pointerului **this**.

```
Student(char* nume, int varsta, int nrNote, int*note)
{
	this->varsta = varsta;
	this->nrNote = nrNote;
	//nume
	//this->nume = nume; //gresit
	this->nume = new char[strlen(nume) + 1];
	strcpy(this->nume, nume);
	//note
	//this->note = note; //gresit
	this->note = new int[nrNote];
	for (int i = 0; i < nrNote; i++)
	this->note[i] = note[i];
}
```

30. Analizaţi de ce o atribuire de tipul this->nume=numes este gresita.
31. Analizaţi de ce nu este necesar sa stergem spatiul cu delete [] this->nume.
32. O scriere declarare echivalenta a constructorului cu parametrii este cea de mai jos. Remarcati modul de iniţializare a atributelor varsta si nrNote

```
Student(char* nume, int varsta, int nrNote, int*note)
		:varsta(varsta),nrNote(nrNote)
{
```

```
	//this->varsta = varsta;
	//this->nrNote = nrNote;
	//nume
	//this->nume = nume; //gresit
	this->nume = new char[strlen(nume) + 1];
	strcpy(this->nume, nume);
	//note
	//this->note = note; //gresit
	this->note = new int[nrNote];
	for (int i = 0; i < nrNote; i++)
		this->note[i] = note[i];
}
```

33. Declarați un obiect de tip student in metoda main

```
void main()
{
	int marks[] = { 10,9 };
	Student s("Nume", 21, 2, marks);
}
```

34. Adăugați o metoda de afișare afiseazaStudent in clasa Student
35. Mutați metoda citesteStudent in cadrul clasei

C++ Codul sursa complet este disponibil pe http://online.ase.ro – "StudentClass - Constructor cu parametrii"

Atribute si metode statice

C++ Execriținl se bazează pe codul scris la pașii anterior, disponibil pe http://online.ase.ro – "StudentClass - Constructor cu parametrii"

36. Declarați in clasa Student un câmp static nrStudenti.

```
class Student
{
private:
	int varsta;
	char* nume;
	int nrNote;
	int *note;
public:
	static int nrStudenti;
	//constructor fara parametrii
	Student()
	{
		.....

}
```

37. Inițializați valoarea variabilei cu 0.

```
int Student::nrStudenti = 0;
void main()
{
	int marks[] = { 10,9 };
	Student s("Nume", 21, 2, marks);
}
```

38. Se va asigura incrementarea numarului de studenti in constructorul fara parametrii al clasei.

```
//constructor fara parametrii
Student()
{
	nume = NULL;
	varsta = 0;
	nrNote = 0;
	note = NULL;
	//incrementare valoare variabila statica
	nrStudenti++;
}
```

39. Se va asigura incrementarea numarului de studenti in constructorul cu parametrii al clasei.
40. Mutați atributul static in zona privata a clasei
41. Declarați o metoda statica de tip get in zona publica a clasei prin care sa se poata accesa valoarea atributului. Testati functionarea metodei prin apelarea ei din metoda main.

C++ Codul sursa complet este disponibil pe http://online.ase.ro – "StudentClass - Atribute si metode statice"

Atribute constante

C++ Execriţiul se bazează pe codul scris la pașii anterior, disponibil pe http://online.ase.ro – "StudentClass - Atribute si metode statice"

- Valoarea atributelor constante nu poate fi modificată după inițializarea atributului prin intermediul listei de inițializatori asociate constructorului clasei;
- Constructorul de clasă va avea obligatoriu în lista de inițializatori și im inițializator pentru atributul constant.

Activitate

42. Similar cu abordarea utilizata la bazele de date, dorim ca fiecare student sa aiba un identificator unic Id. Identificatorul nu va putea fi modificat.

```
class Student
{
private:
	const int id;

	int varsta;
	char* nume;
	int nrNote;
	int *note;
	static int nrStudenti;
public:
	//constructor fara parametrii
	Student():id(nrStudenti)
	{
		nume = NULL;
		varsta = 0;
		nrNote = 0;
		note = NULL;
		//incrementare valoare variabila statica
		nrStudenti++;
	}
	Student(char* nume, int varsta, int nrNote, int*note)
		:varsta(varsta), nrNote(nrNote), id(nrStudenti)
	{
		//this->varsta = varsta;
```

```
            //this->nrNote = nrNote;
            //nume
            //this->nume = nume; //gresit
            this->nume = new char[strlen(nume) + 1];
            strcpy(this->nume, nume);
            //note
            //this->note = note; //gresit
            this->note = new int[nrNote];
            for (int i = 0; i < nrNote; i++)
                  this->note[i] = note[i];
            //incrementare valoare variabila statica
            nrStudenti += 1;
      }
      .......

      //Get nrStudenti
      static int GetNrStudenti()
      {
            return nrStudenti;
      }
};
```

43. Verificati initializarea corecta a campului id in cadrul main.

Codul sursa complet este disponibil pe http://online.ase.ro – “StudentClass - Atribute si metode statice”