Universitatea “Al.I.Cuza” Iasi **Nume:**
Facultatea de Informatică **Facultatea** **Sectia:** **An:** **Grupa(1-10):**

Examen la Programare II

8.07.03

Observații:

1. *Nu este permisă consultarea bibliografiei.*
2. *Toate întrebările sunt obligatorii.*
3. *Fiecare întrebare este notată cu 3 puncte exceptand exercitiul 9-10 care se puncteaza cu 6 puncte. Pentru intrebarile grila: 1 punct alegerea corectă a variantei, 2 puncte justificarea.*
4. *Nu este permisă utilizarea de foi suplimentare.*

1)

```
#include<iostream>
using namespace std;
inline void f(int& a, int b, int c
= 0)
   {a=b+c;}
inline void f(double& a, int b = 0,
              double c = 0.0)
   {a=b+c;}
int main(){
   int x; double y;
   f(x,3); f(y,3);
   cout << x << ' ' << y;
   return 0;
}
```

Precizati care sunt conceptele C++ reprezentate in codul alaturat si indicati rezultatul executiei.

Raspuns.

<iostream> este un fișier antet ce conține suportul pentru sistemul de intrare/ieşire

namespace std este spațiul de nume sub care sunt grupate toate funcțiile bibliotecii standard

functii inline sunt funcții al căror cod este înlocuit în codul sursă al programului apelant(se evită operațiile costisitoare legate de apelul funcțiilor clasice)

referința este un alias pentru un obiect; un parametru transmis prin referință indică implicit spre argumentul folosit pentru a apela funcția

cout este fluxul standard de ieşire

operatorul << supraîncârcat ca operator de ieşire introduce caractere într-un flux de ieşire

2)

```
class Test{
public:
      int x;
      Test():y(0), z(0){}
      Test(int a, int b, int c)
         :x(a),y(b),z(c){}
      int GetVal() const {return z;}
protected:
      int y;
      int SetVal(int Val) {y = Val;}
private:
      int z;
      int PutVal(int Val){z = Val;}
};
```

Care din afirmatiile ce urmeaza, referitoare la codul alaturat, este falsa:

a) Clasa Test are 2 constructori
b) Programul principal poate accesa membrul x din Test
c) Programul principal poate accesa membrul y din Test
d) Clasele derivate din Test pot accesa membrii x si y
e) Clasele derivate din Test nu pot accesa membrul z

Justificare.

Afirmația este falsă deoarece membrii protected pot fi accesați doar din clasele derivate din clasa Test. Din programul principal nu se poate accesa decât membrul public x.

3)

```
#include <iostream.h>
class C{
public:
      C(){n++;}
      static int index(){return n;}
private:
      static int n;
};
int C :: n = 0;
```

Care este rezultatul executiei codului urmator?

a) 1 3 5 7
b) 1 2 3 4
c) 0 0 0 0
d) 0 1 2 3

Justificare. La instanțierea obiectului a1 de tip A se apelează constructorul implicit la lui C pentru variabila membru a1.c, ceea ce duce la incrementarea variabilei membru statice n a clasei C. Apelul funcției statice index va

`class A{`
`private:`
`  C c;int a;`
`};`
`class B{`
`public:`
`  B(int i = 0) :b(i)`
`  {cout << C :: index() << ' ';}`
`private:`
`  A a;C c; int b;`
`};`
`int main(){`
`  A a1;`
`  cout << C :: index() << ' ';`
`  B b1[3];`
`  return 0;`
`}`
	returna **1**, valoare care este afişată.
	În continuare se instanțiază un vector de 3 obiecte de tip B, pentru fiecare dintre acestea se apeleaza în această ordine:
	• Pentru membrul a:
	o constructorul implicit al clasei C(are loc n++)
	o constructorul implicit al clasei A
	• Pentru data membru c:
	o Constructorul implicit al clasei C(n++)
	În concluzie, pentru fiecare obiect din vector, constructorul lui C este apelat de 2 ori, iar variabila statica n este incrementată de 2 ori, ducând la următoarea secvență de numere: **3 5 7**.

4)
`#include <iostream>`
`using namespace std;`
`class A{public:virtual char label() = 0;};`
`class B:public A{`
`public:`
`  char label(){return 'b';}`
`protected:`
`  static int b;`
`};`
`class C:public B{`
`public:`
`  C(){b++;}`
`  char label(){ cout << b; return 'c';}`
`};`
`int B::b = 0;`
`int main() {`
`  A* a = new C;`
`  cout << a->label() << " ";`
`  B* b = new C;`
`  cout << b->label() << " ";`
`  delete a;  delete b;  return 0;`
`}`
	Ce este afisat dupa executia programului alaturat?
	a) 0c 1c
	b) 1c 2c
	c) 1b 2b
	d) 1b 2c
	
	Justificare.
	Clasa A este abstractă şi are ca membru o metodă virtuală `label`. Suprascrierea funcțiilor virtuale în clasele derivate B şi C permite polimorfismul la execuție. Astfel, pointerul a la clasa A este folosit spre a indica spre clasa C, derivată din A, iar apelul `a->label` va determina apelul metodei `label` din C.
	Inițial membrul `static b` al clasei B este 0. La crearea obiectului *a, b devine 1. La apelul `label`, se afişează valoarea lui b, adică 1, urmată de afişarea caracterului ‘c’, valoarea returnată de funcție.
	În mod similar, ponterul spre clasa B este utilizat pentru a se crea un obiect *b de tip C, n devine 2, iar apelul funcției label din C determină afişarea valorii 2, urmată de caracterul ‘c.’

5)

```
#include <algorithm>
#include <vector>
#include <string>
#include <iostream>
using namespace std;
int main() {
  ostream_iterator<string> out(cout, "
");
  vector<string> v, r;
  v.push_back("unu");
  v.push_back("doi");
  v.push_back("trei");
  v.push_back("patru");
  v.push_back("cinci");
  copy(v.begin(), v.end(), out);
  cout << endl;
  sort(v.begin(), v.end());
  copy(v.begin(), v.end(), out);
  cout << endl;
  v.pop_back();
  v.erase(v.begin());
  copy(v.begin(), v.end(), out);
  cout << endl;
  return 0;
}
```

Explicati codul alaturat si precizati ce se afiseaza dupa executie.

Raspuns.

Sunt utilizate clasele din STL(Standard Template Library)`<algorithm>`(pentru `sort, copy`) `<vector>`pentru utilizarea obiectelor `vector`
Se declară un obiect de tip `ostream_iterator`, care inserează un obiect(de tip string, în cazul nostru) într-un flux de ieşire(`cout`), utilizându-se delimitatorul " ". Se declară 2 obiecte de tip vector de string-uri. În v se adaugă la sfârşit pe rând valorile "unu", "doi", "trei", "patru", "cinci". Apoi vectorul este afişat pe ecran utilizându-se funcţia `copy`, care utilizează un iterator asupra vectorului v, copiind conţinutul acestuia în iteratorul `out` definit anterior. Se afişează `unu doi trei patru cinci.` Apoi vectorul este sortat lexicografic, apoi se afisează pe ecran vectorul sortat: `cinci doi patru trei unu. pop_back` extrage ultimul element din vector. Erase şterge elementul indicat de `v.begin(),` adică primul element din vector. A treia oară se afişează `doi trei patru`.

6)

```
#include <iostream.h>
#include <string.h>
class StringVar{
      char *str;
public:
  StringVar(char *msg) {
      str = new char[strlen(msg)+1];
      strcpy(str, msg);
  }
  ~StringVar(){delete str;}
  friend ostream& operator<<(ostream&
out,
                    const StringVar&
sir);
};
void afisare (StringVar& sir)
  { cout << "Sirul este:" << sir; }
ostream& operator<<(ostream& out,
                    const StringVar&
sir)
  {
      return out << sir.str;
  }
void main(){
  StringVar mesaj("Ce mai faci?");
  afisare(mesaj);
  cout << "Dupa apel:" << mesaj <<
endl;
}
```

Ce puteți spune despre programul alaturat:

a) este corect şi va afişa la execuţie: Sirul este: Ce mai faci?Dupa apel: Ce mai faci?
b) este corect şi va afişa la execuţie: Sirul este: Ce mai faci?Dupa apel:
c) eroare la compilare
d) eroare la execuţie
e) comportare nedeterminata

Justificare.

Se instanțiază obiectul mesaj de tip `StringVar`, apelându-se constructorul cu parametru al clasei. Urmează apelul funcției globale `afisare`, care practic apelează funcția `operator<<`, care a fost supraîncărcată pentru clasa `StringVar`. Urmarea: se afişează "Sirul este: Ce mai faci?" În continuare, din programul principal se afişează şirul "Dupa apel:", după care se apelează din nou funcția `operator<<` pentru obiectul mesaj, adică "Sirul este:Ce mai faci?"

7)

```
class A {
public:
	virtual void Afisare() { cout<<"A ";
}
};

class B: public A {
public:
	B(){ Afisare(); }
	void Afisare() { cout<<"B "; }
};
void main() {
	B *b = new B;
	b->A::Afisare();
	((A*)b)->Afisare();
}
```

Ce va afişa următorul program?

a) B B B
b) A A A
c) B A A
d) B A B
e) nimic deaorece va da eroare la compilare

Justificare.

Clasa A defineşte o metodă virtuală `Afisare.` Crearea obiectului b de tip B* determină apelul metodei `Afisare` din clasa B(apelată din constructor), care afişează "B ". Utilizarea operatorului de rezoluție forțează apelul funcției afişare din clasa A , urmarea fiind afişarea şirului "A ". Datorită polimorfismului la execuție, conversia explicită de tip nu influențează comportamentul obiectului, apelându-se tot funcția `Afişare()` din clasa B şi se afişează "B ".

8)

```
#include <iostream.h>
template <class T>
class A { protected: T x; };
template <class T>
class C;
template <class W>
class B : public A<W> {
protected:
	int y;
public:
	B(int a=0) : y(a) {}
	friend class C<W>;
	W getX() {return x;}
};

template <class T>
class C {
private:
	B<T> b;
public:
	C() : b(5) {}
	int f() {return b.y;}
};
void main() {
	C<double> c; cout << c.f();
}
```

Programul alaturat:

a) nu se poate executa pentru ca are erori sintactice;
b) va da eroare in timpul executiei;
c) va afisa 5 dupa executie;
d) va afisa 0 dupa executie.

Justificare.

La crearea obiectului c al clasei se apelează constructorul implicit al clasei C, care(prin lista de inițializare) va apela constructorul cu parametru al clasei B(cu argumentul 5), constructor care inițializează variabila membru `y` cu valoarea 5.
Urmează apelul funcției `f` din clasa C. Funcția retrunează valoarea variabilei membru `y` a obiectului `b` de tip B (din clasa C). Clasa C este friend a clasei B, deci poate accesa membrul protected al acesteia y.
Valoarea returnată de `c.f()` este 5, valoarea care este afişată.

9-10)

a) Să se proiecteze clase pentru reprezentarea listelor (secvențelor) de dreptunghiuri în plan. Se consideră numai dreptunghiuri cu laturile paralele cu axele. Un dreptunghi este precizat prin coordonatele a două vârfuri opuse: stânga-sus şi dreapta-jos.
b) Să se scrie un subprogram care determină un dreptunghi cu diagonala cea mai mare dintr-o listă dată.
Rezolvare:

```
class Dreptunghi{
private:
	double x1,y1,x2,y2;
public:
	Dreptunghi(){x1=y1=x2=y2=0;}
	double diagonala(){
		return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
	}
	friend ostream& operator <<(ostream& os,const Dreptunghi& dr){
		os<<"("<<dr.x1<<","<<dr.y1<<")"<<" ("<<dr.x2<<","<<dr.y2<<")";
	return os;
	}
```

```
	friend istream& operator >>(istream& is,Dreptunghi& dr){
		is>>dr.x1>>dr.y1>>dr.x2>>dr.y2;
	return is;
	}

};

class ListaDr{
private:
	Dreptunghi *secv;
	int dim;
public:
	ListaDr();
	ListaDr(Dreptunghi*, int dim);
	~ListaDr();
	Dreptunghi detMax();
};
ListaDr::ListaDr(){
	dim=0;
	secv=NULL;
}

ListaDr::ListaDr(Dreptunghi *secv, int dim){
	this->dim=dim;
	this->secv=new Dreptunghi[dim];
	for(int i=0;i<dim;i++){
		*(this->secv+i)=*(secv+i);
	}
}

ListaDr::~ListaDr(){
	delete [] secv;
}

Dreptunghi ListaDr::detMax(){
	int i,pozMax=0;
	double diag,maxDiag=-1;
	for(i=0;i<dim;++i){
		diag=(secv+i)->diagonala();
		if(diag>maxDiag){
			maxDiag=diag;
			pozMax=i;
		}
	}
	return *(secv+pozMax);
}
void main(){
	int dim;
	Dreptunghi ld[10];
	cout<<"Numarul de dreptunghiuri";
	cin>>dim;
	for(int i=0;i<dim;i++){
		cout<<"coordonatele stanga sus si dreapta jos pentru dreptunghiul "<<i+1;
		cin>>ld[i];
	}
	ListaDr lst(ld,dim);
	Dreptunghi d=lst.detMax();
	cout<<d;

}
```