Varianta 1

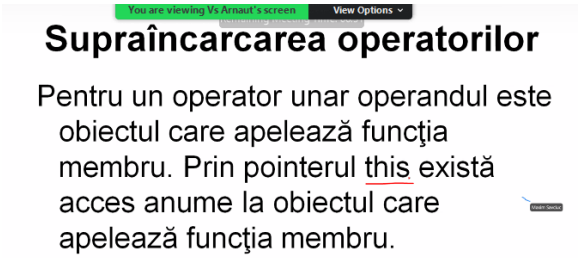
1)

```
#include <iostream>
using namespace std;

template <class T>
class Stiva {
private:
    T* data;
    int maxSize;
    int currentIndex;

public:

    Stiva();

    void push(T element);

    T operator-();

    void print();
};


template <typename T>
Stiva<T>::Stiva() {
    data = new T[100];
    maxSize = 100;
    currentIndex = 0;
}

template <typename T>
void Stiva<T>::push(T element) {
    if (currentIndex >= maxSize)
        cout << "ERROR\n";
    data[currentIndex++] = element;
}

template <typename T>
T Stiva<T>::operator-() {
    T ret = data[--currentIndex];
    return ret;
}

template <typename T>
void Stiva<T>::print() {
    cout << "Continutul stivei:\n";
    for (int i = currentIndex - 1; i >= 0; i--)
    {
        cout << data[i] << endl;
    }
}

int main() {
    Stiva<int> stiva1;

    stiva1.push(2);
    stiva1.push(2);
    stiva1.push(5);
    stiva1.push(5);
    stiva1.push(8);
    stiva1.push(8);

    stiva1.print();

    -stiva1;
    -stiva1;
    stiva1.print();

}
```

2) int

3)
```
class Patrat {
    double a;
public:
    Patrat() {
        a = 0;
    }
    Patrat(double x) {
        a = x;
    }
    double SQR() {
        return a * a;
    }
};
```

4)



5)
```
class Persoana {
protected:
    string nume;
public:
    Persoana() {
        nume = "";
    }
    Persoana(string s) {
        nume = s;
    }
};

class Student:public Persoana {
protected:
    string facultate;
public:
    Student() {
        nume = "";
        facultate = "";
    }
    Student(string s,string f) {
        nume = s;
        facultate = f;
    }
};

class Student_Chimist :public Student {
public:
    Student_Chimist() {
        nume = "";
        facultate = "";
    }
    Student_Chimist(string s) {
        nume = s;
        facultate = "chimie";
    }
};

class Student_Fizician :public Student {
public:
    Student_Fizician() {
        nume = "";
        facultate = "";
    }
    Student_Fizician(string s) {
        nume = s;
        facultate = "fizica";
    }
};
```

6)
```
class Sir {
private:
    const char* str;
public:
    Sir() {
        str = new char[333];
        str = "";
    }

    Sir(const char* x) {
        str = new char[333];
        str = x;
    }

    Sir(const char* x) {
        str = new char[333];
        str = x;
    }

    Sir(int x) {
        str = new char[x];
    }

    ~Sir() {
        delete[] str;
    }
};
```

7) “nu”
0
512.4

8)
Virtual este utilizat la declarareafunctiilor virtuale. O funcție virtuală este o funcție membru care este declarată într-o clasă de bază și este redefinită(înlocuită) de o clasă derivată. Când vă referiți la un obiect de clasă derivat folosind un pointer sau o referință la clasa de bază, puteți apela o funcție virtuală pentru acel obiect și executa versiunea clasei derivate a funcției.

9)

```
#include <iostream>

using namespace std;

template <typename T>
class plusVector {
private:
    T* arr;
    int size;
public:
    plusVector() {
        arr = new T[0];
        size = 0;
    }

    plusVector(int n,T tab[]) {
        arr = new T[n];
        size = n;
        for (size_t i = 0; i <size; i++)
            arr[i] = tab[i];

    }

    void operator+(plusVector x ) {
        for (size_t i = 0; i < size; i++)
            this->arr[i] += x.arr[i];
    }

    void afisare() {
        for (size_t i = 0; i < size; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};

int main()
{
    int a[] = { 1,2,3,4,5 };
    plusVector<int> arr1(5, a);

    int b[]= { 2,2,2,2,2 };
    plusVector<int> arr2(5, b);

    arr1+arr2;

    arr1.afisare();

    double c[] = { 1.1,2.2};
    plusVector<double> arr3(2,c);

    double d[] = { 2.0,2.0 };
    plusVector<double> arr4(2, d);

    arr3 + arr4;

    arr3.afisare();
}
```

10) *Constructorul de copiere* crează obiectul și îl inițializează cu un obiect deja existent dat ca parametru.

Exemplu:

```
Name(Name &obj) {
      Var1 = obj.Var1;
      Var2 = obj.Var2;
```

Varianta 2

1)
```
class Fundament {

protected:
    string nume;
public:
    Fundament() {
        nume = "";
    }
    Fundament(string s) {
        nume = s;
    }
};

class Casa :public Fundament {
protected:
    string tipCasa;
public:
    Casa() {
        nume = "";
        tipCasa = "";
    }
    Casa(string s, string t) {
        nume = s;
        tipCasa = t;
    }
};

class CasaCuEtaje :public Casa {
public:
    CasaCuEtaje() {
        nume = "";
        tipCasa = "CasaCuEtaje";
    }
    CasaCuEtaje(string s) {
        nume = s;
        tipCasa = "CasaCuEtaje";
    }
};

class CasaCeva :public Casa {
public:
    CasaCeva() {
        nume = "";
        tipCasa = "CasaCeva";
    }
    CasaCeva(string s) {
        nume = s;
        tipCasa = "CasaCeva";
    }
};
```

2) Void

~Timp(int);

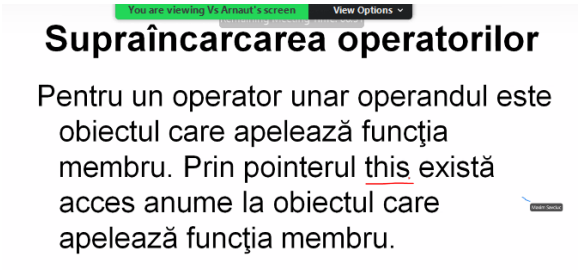
3)
```
class Rand {

private:
    const char* linie;
public:
    Rand() {
        linie = new char[90];
    }
    Rand(const char* z) {
        linie = new char[90];
        linie = z;
    }
    Rand(int n) {
        linie = new char[n];
    }
    ~Rand(){
        delete[] linie;
    }
};
```

4)
```
class Dreptunghi {

    double a,b;
public:
    Dreptunghi() {
        a = 0;
        b = 0;
    }
    Dreptunghi(double x,double y) {
        a = x;
        b = y;
    }
    double Ar() {
        return a * b;
    }
};
```

5)

Cuvantul rezervat **friend** are drept scop definire unei clase/functii prietene a altei clase predefinite. Declararea functiei prietene apare într-un corp de clasă și acordă unei funcții sau unei alte clase acces membrilor privați și protejați ai clasei respective în care apare declarația **friend**.

8)



7)

Initiere:
Prin intermediului unui constructor

Definiţie 1. Constructorul este funcţia membră specială a clasei, destinată pentru iniţializarea fiecărui obiect al acesteia. Iniţializarea se face imediat după crearea obiectului (imediat după declararea variabilei de tip clasă), adică după ce el a fost declarat şi lui i s-a alocat memorie.

Distrugere:

Definiţie 2. Destructorul este funcţia membră specială a clasei, destinată pentru efectuarea unor operaţii adăugătoare la distrugerea fiecărui obiect al clasei, când expiră termenul lui de „viaţă". (De exemplu, eliberarea memoriei alocate obiectului, închiderea sau ştergerea unor fişiere etc. Deci, destructorul are funcţia de a elimina toate „urmele" obiectului la distrugerea lui)

3

6)

```
#include <iostream>

using namespace std;

template<typename T>
class printVector {
private:
    T* arr;
    int size;
    int current;
public:
    printVector() {
        arr = new T[100];
        size = 100;
        current = 0;
    }
    printVector(int x) {
        arr = new T[x];
        size = x;
        current = 0;
    }

    void add(T x) {
        arr[current++] = x;
    }
    ~printVector() {
        delete[] arr;
    }
    void afisare() {
        for (size_t i = 0; i < size; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};
int main()
{
    printVector<int> a(2);
    a.add(2);
    a.add(3);
    a.afisare();

    printVector<float> b(2);
    b.add(2.2);
    b.add(3.2);
    b.afisare();

}
```

9)

```
#include <iostream>

using namespace std;

class Masa {
    int valoare;
public:
  /*  Masa() { valoare = 0; };*/
    Masa(int m = 20) { valoare = m; };
    void afisare() { cout << valoare; };
};
int main()
{
    Masa mz;
    Masa m(37);
    mz.afisare();
    m.afisare();
}
```

"da"

Explicatii: stergem primul constructor implicit fara parametri.

10)
```
#include <iostream>

using namespace std;

template <typename T>
class Vector {
private:
    T* arr;
    int size;
public:
    Vector() {
        arr = new T[0];
        size = 0;
    }

    Vector(int n, T tab[]) {
        arr = new T[n];
        size = n;
        for (size_t i = 0; i < size; i++)
            arr[i] = tab[i];

    }

    void operator+(Vector x) {
        for (size_t i = 0; i < size; i++)
            this->arr[i] += x.arr[i];
    }

    void afisare() {
        for (size_t i = 0; i < size; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};

int main()
{
    int a[] = { 1,2,3,4,5 };
    Vector<int> arr1(5, a);

    int b[] = { 2,2,2,2,2 };
    Vector<int> arr2(5, b);

    arr1 + arr2;

    arr1.afisare();

    double c[] = { 1.1,2.2 };
    Vector<double> arr3(2, c);

    double d[] = { 2.0,2.0 };
    Vector<double> arr4(2, d);

    arr3 + arr4;

    arr3.afisare();
}
```