Laborator #2
Clase & Obiecte

Constructori

Constructori de copiere (copy ctor)

Constructorii de copiere = funcții membre care inițializează obiecte folosind datele unui alt obiect din aceeași clasă, inițializat anterior.

```
class Rectangle{
private:
    float width;
    float height;

public:
    Rectangle(float width, float height){
        std::cout << "Constructor cu 2 parametri.\n";
        this->width = width;
        this->height = height;
    }

    Rectangle(const Rectangle &r){
        std::cout << "Constructor de copiere.\n";
        this->width = r.width;
        this->height = r.height;
    }
};
```

```
#include <iostream>

class Rectangle{...};

int main(){
    Rectangle r1(5, 3);
    Rectangle r2(r1); //r2.width=5; r2.height=3

    return 0;
}
```

```
Constructor cu 2 parametri.
Constructor de copiere.

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

Constructori de copiere (copy ctor)

```
Rectangle(const Rectangle &r){
    std::cout << "Constructor de copiere.\n";
    this->width = r.width;
    this->height = r.height;
}
```

- cuvântul cheie `const` – vrem să ne asigurăm că atunci când facem copierea nu modificăm obiectul pe care îl copiem (referința); prin urmare îl obligăm să rămână constant în domeniul constructorului.
- referința (`Rectangle &r`) – un nume alternativ pentru o variabilă existentă. Se obține prin prefixarea identificatorului (numelui) variabilei cu simbolul `&`.
  - nu forțează crearea unei copii suplimentare în funcția apelantă, ci folosește o indirectare către un obiect deja existent.
  - nu au nevoie de operator de dereferențiere pentru accesarea valorii.
  - membrii unei referințe pot fi accesați cu operatorul '.', fără să fie nevoie de '->'.

Ce s-ar întâmpla dacă nu s-ar folosi referință?

Constructori de copiere (copy ctor)

```
class Student{
private:
    int *note;
    int nr_note;
public:
    Student(int *note, int nr_note){
        this->nr_note = nr_note;
        this->note = new int[nr_note];
        for (int i=0; i<nr_note; ++i){
            *(this->note+i) = *(note+i);
        }
    }
    Student(Student &s){
        this->nr_note = s.nr_note;
        this->note = s.note;
    }
    void display(){
        for (int i=0; i<nr_note; ++i){
            std::cout << *(note+i) << " ";
        }
    }
    void increment(int incr){
        for(int i=0; i<nr_note; ++i){*(note+i) += incr;}
    }
};
```

Operator care alocă memorie și returnează pointer către începutul blocului alocat.

Copierea implicită se face element cu element

```
#include <iostream>

class Student{...};

int main(){
    int nr_note;
    std::cin >> nr_note;
    int *note = new int[nr_note];
    for(int i=0; i<nr_note; i++){
        std::cin >> *(note+i);
    }
    Student s1(note, nr_note);
    Student s2(s1);
    s1.display();
    std::cout<<std::endl;
    s2.increment(5);
    s1.display();

    return 0;
}
```

```
3
1 5 7
1 5 7
6 10 12
Process returned 0 (0x0)   execution time : 4.512 s
Press any key to continue.
```

Constructori de copiere (copy ctor)

```
class Student{
private:
    int *note;
    int nr_note;
public:
    Student(int *note, int nr_note){
        this->nr_note = nr_note;
        this->note = new int[nr_note];
        for (int i=0; i<nr_note; ++i){
            *(this->note+i) = *(note+i);
        }
    }
    Student(Student &s){
        this->nr_note = s.nr_note;
        this->note = new int[this->nr_note];
        for (int i=0; i<nr_note; ++i){
            *(this->note+i) = *(s.note+i);
        }
    }
    void display(){
        for (int i=0; i<nr_note; ++i){
            std::cout << *(note+i) << " ";
        }
    }
    void increment(int incr){
        for(int i=0; i<nr_note; ++i){*(note+i) += incr;}
    }
};
```

```
#include <iostream>

class Student{...};

int main(){
    int nr_note;
    std::cin >> nr_note;
    int *note = new int[nr_note];
    for(int i=0; i<nr_note; i++){
        std::cin >> *(note+i);
    }
    Student s1(note, nr_note);
    Student s2(s1);
    s1.display();
    std::cout<<std::endl;
    s2.increment(5);
    s1.display();

    return 0;
}
```

```
3
1 5 7
1 5 7
1 5 7
Process returned 0 (0x0)   execution time : 4.559 s
Press any key to continue.
```

Destructori

Destructori

Destructor = funcție membră specială a unei clase, cu același nume cu clasa, însă precedată de `~`, folosită pentru a elibera resursele ce au fost ocupate de obiect în timpul existenței sale.

- Destructorul este apelat implicit la finalul duratei de viață a unui obiect. De obicei, nu se apelează explicit.
- Ca regulă generală, pentru fiecare alocare dinamică de memorie (utilizarea cuvântului cheie `new`) avem nevoie de o eliberare dinamică de memorie (cuvântul cheie `delete`).

Destructori

```
class Student{
private:
    int *note;
    int nr_note;
public:
    Student(int *note, int nr_note){
        this->note = new int[nr_note];
        for (int i=0; i<nr_note; i++){
            *(this->note+i) = *(note+i);
            this->nr_note = nr_note;
        }
    }

    ~Student(){
        delete [] note;
        std::cout << "Apel destructor!" << std::endl;
    }

    void display(){
        for (int i=0; i<this->nr_note; ++i){
            std::cout << *(note+i) << " ";
        }
        std::cout << std::endl;
    }
};
```

```
#include <iostream>

class Student{...};

int main(){
    int a[4] = {3, 7, 1, 5};
    Student s1(a, 4);
    s1.display();

    return 0;
}
```

```
3 7 1 5
Apel destructor!

Process returned 0 (0x0)   execution time : 0.082 s
Press any key to continue.
```

Alocare dinamică a unui vector de întregi

Eliberarea memoriei alocate dinamic pentru un vector

Liste de inițializare

Liste de inițializare

Liste de inițializare = un mod de a inițializa datele membre în definiția constructorilor.

Utilizare:

- Inițializarea datelor membre constante non-statice
- Inițializarea datelor membre de tip referință
- Inițializarea obiectelor membre care nu au constructor default

Liste de inițializare

```
#include <iostream>
class Example{
private:
    int m;
    int &r;
    const int c;
public:
    Example(int m, int &referinta, const int constanta):m(m), r(referinta), c(constanta){}

    void display(){
        std::cout << "m:" << m << std::endl;
        std::cout << "r:" << r << std::endl;
        std::cout << "c:" << c << std::endl;
    }
};

int main(){
    int a = 20;
    const int b = 30;
    Example e(10, a, b);
    e.display();
    return 0;
}
```

```
m:10
r:20
c:30

Process returned 0 (0x0)   execution time : 0.024 s
Press any key to continue.
```

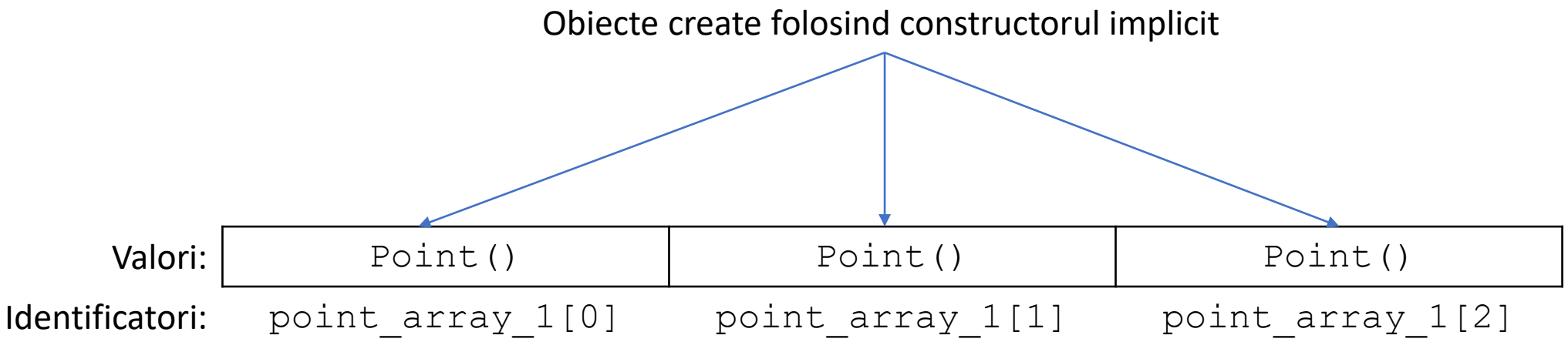
Vectori de obiecte

1. Vector static de obiecte fără inițializare explicită

```
#include <iostream>

class Point{
    int x, y;
public:
    Point(int x=0, int y=0):x(x), y(y){}
    void display(){
        std::cout << "x=" << x << std::endl;
        std::cout << "y=" << y << "\n\n";
    }
};

int main(){
    Point point_array_1[3];

    for (int i=0; i<3; ++i){
        point_array_1[i].display();
    }

    return 0;
}
```

Apel implicit constructor fără parametri

1. Vector static de obiecte fără inițializare explicită

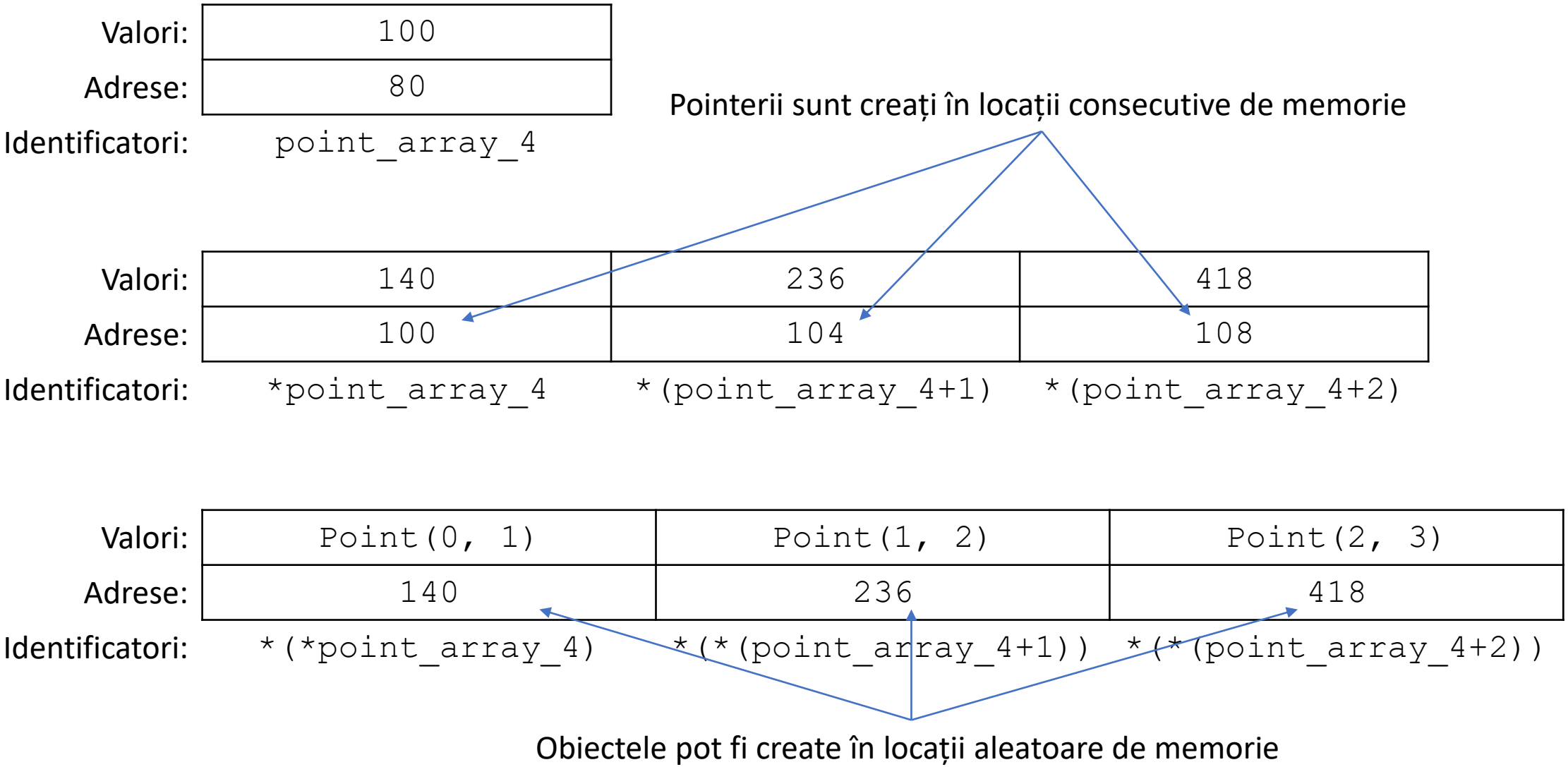


2. Vector dinamic de pointeri cu inițializare explicită

```
#include <iostream>
class Point{
    int x, y;
public:
    Point(int x=0, int y=0):x(x), y(y){}
    void display(){
        std::cout << "x=" << x << std::endl;
        std::cout << "y=" << y << "\n\n";
    }
};
int main(){
    int nr_points;
    Point **point_array_4 = nullptr;
    std::cin >> nr_points;
    point_array_4 = new Point*[nr_points];

    for(int i=0; i<nr_points; ++i){
        *(point_array_4 + i) = new Point(i, i+1);
    }
    for(int i=0; i<nr_points; ++i){
        (*(point_array_4 + i)) -> display();
        delete *(point_array_4 + i);
    }
    delete point_array_4;
    return 0;
}
```

```
3
x=0
y=1

x=1
y=2

x=2
y=3


Process returned 0 (0x0)   execution time : 1.615 s
Press any key to continue.
```

2. Vector dinamic de pointeri cu inițializare explicită



Sfârșit laborator #2