7/04/2023

Run time polymorphism

Run time is when the execution of the program has started. Here we will be discussing about the function over-riding.

Base class

```
Animal

void speak() {
    cout << "speaking";
}
```

derived class

```
Dog

void speak() {
    cout<<"Bark";
}
```

Inheritance

The function of base class will be inherited in the derived class but in the derived class we defined our own speak function & hence that code would be executed & hence this is known as function over-riding.

The advantage here is again reusability of code if we don't explicitly define the function in derived class.

Important cases

1) `Animal* a = new Animal();`
   `a->speak();`
   Here speaking is printed.

2) `Dog* b = new Dog();`
   `b->speak();`
   Here Barking is printed

3) Animal * a = new Dog();
a→speak();
Here if virtual keyword is not used in function of base class, then speaking is printed else barking is printed.
This concept is known as upcasting.

4) Dog * b = (Dog*) new Animal();
b→Speak();

Virtual keyword not used → Barking
Virtual keyword is used → Speaking

Note→ The below concept is very important for MCQ questions.

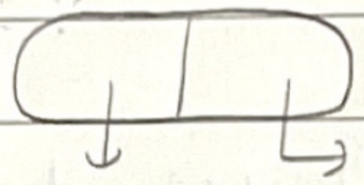
Virtual keyword used → new class() → function is called
Virtual keyword not used → Function of pointer will be called.

Constructor game
Base class → Animal
Child class → Dog

1) Animal * a = new Animal();
Here Animal's constructor will be called.
2) Dog * b = new Dog();
Animal constructor
Dog constructor
3) Animal * a = new Dog();
Animal constructor
Dog constructor

4) Dog * b = (Dog*) new Animal();
Animal constructor is called.

Abstraction ↱ This line is important
Encapsulation is subset of abstraction.
Abstraction is basically implementation hiding. Encapsulation is data hiding but leave the above concept.



Abstraction is basically showing essential information only.

Ex→ Group of many things → Here we can say we have achieved both encapsulation & abstraction.

Note→ Abstraction is basically when we are trying to achieve generalization.

Dyanamic memory allocation
Any program we create will be needing some memory such as int taking 4 bytes of memory.

(i) stack memory
Here local variables & function parameters are stored here & stack memory by default is smaller. However we can increase the size of stack memory with the help of some compiler and OS settings.

(ii) heap memory

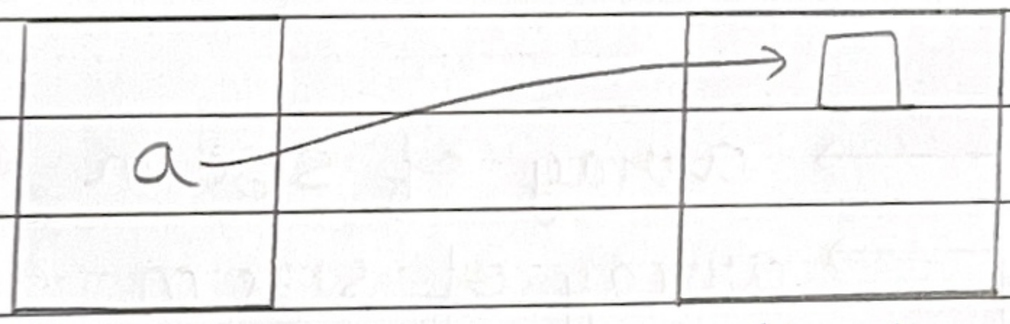
When new keyword is used, heap memory will be occupied. Also heap memory is bigger than stack.

```
int a = 5;      } Stack memory
int *a = new int;   } heap memory
```

returns address & stored in pointer.

Using delete keyword, or operator we can free the memory from heap & we have to do it.

```
int *a = new int;
stack   heap
```



Why heap is used?

int arr [n]; → Bad practice

This was a bad practice as it might happen that n was very large and there may be not enough space in the stack as it was smaller in size and heap is bigger than stack.

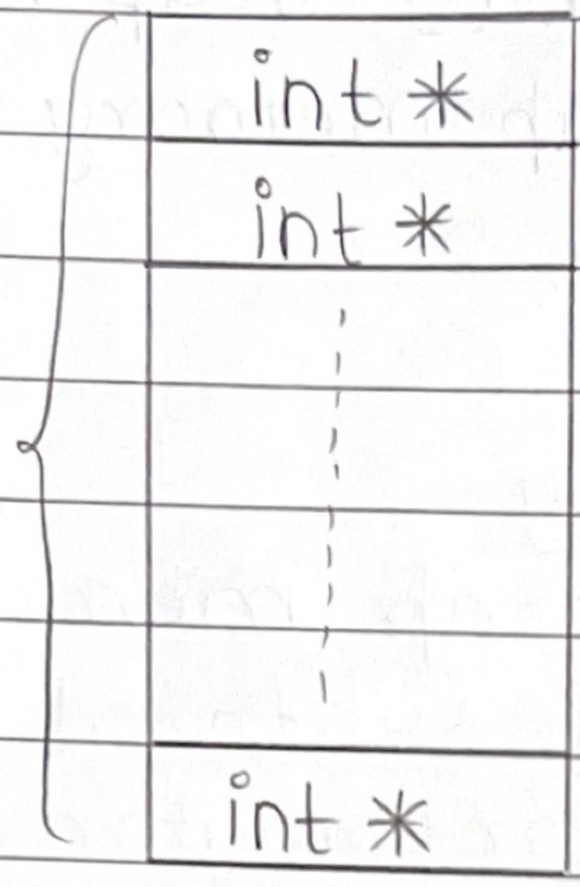
1D array creation

```
int * arr = new int [n];
```

size of array

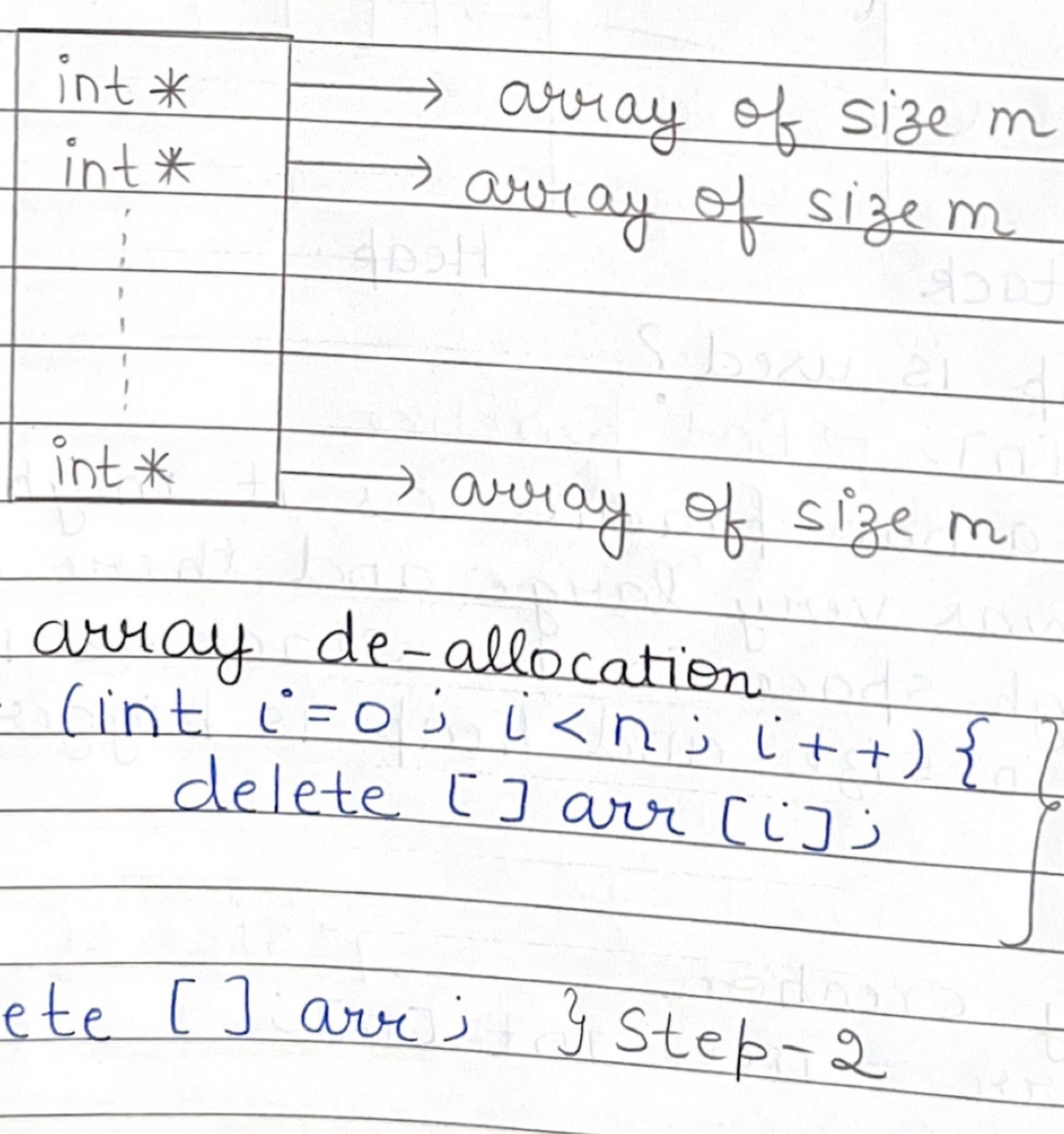
2D array creation

```
int **arr = new int*[n];
```



Run a for loop as,

```
for (int i=0 ; i<n ; i++) {
        arr[i] = new int[m];
}
```
↳ pointer



2D array de-allocation

```
for (int i=0 ; i<n ; i++) {
        delete [] arr[i];
}
delete [] arr ;
```
Step-1
Step-2

Here n → number of columns