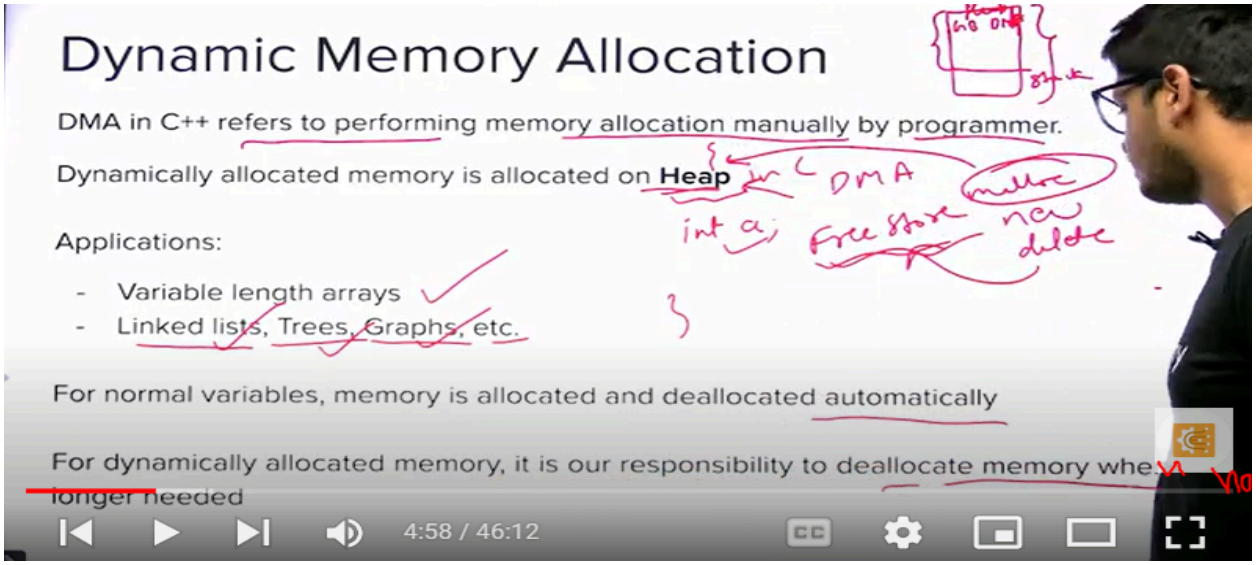
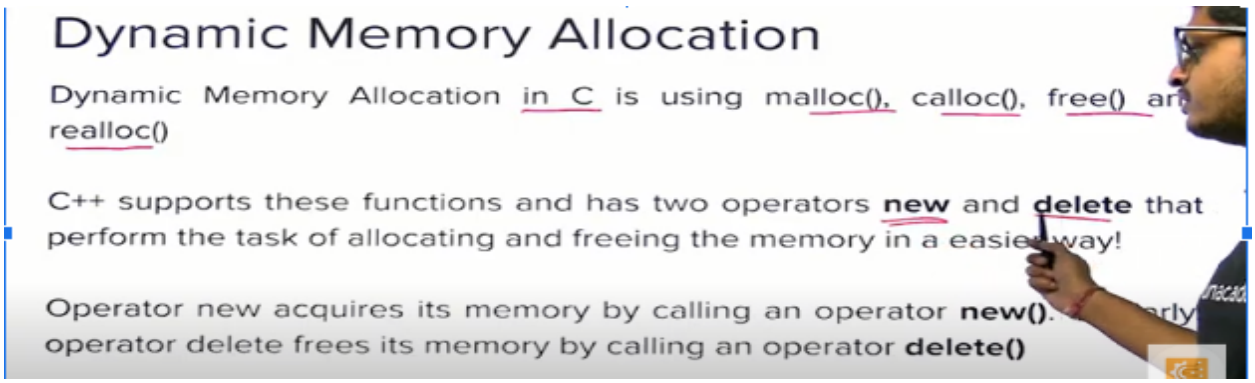
**Dynamic Memory Allocation & Typecasting | C++...**

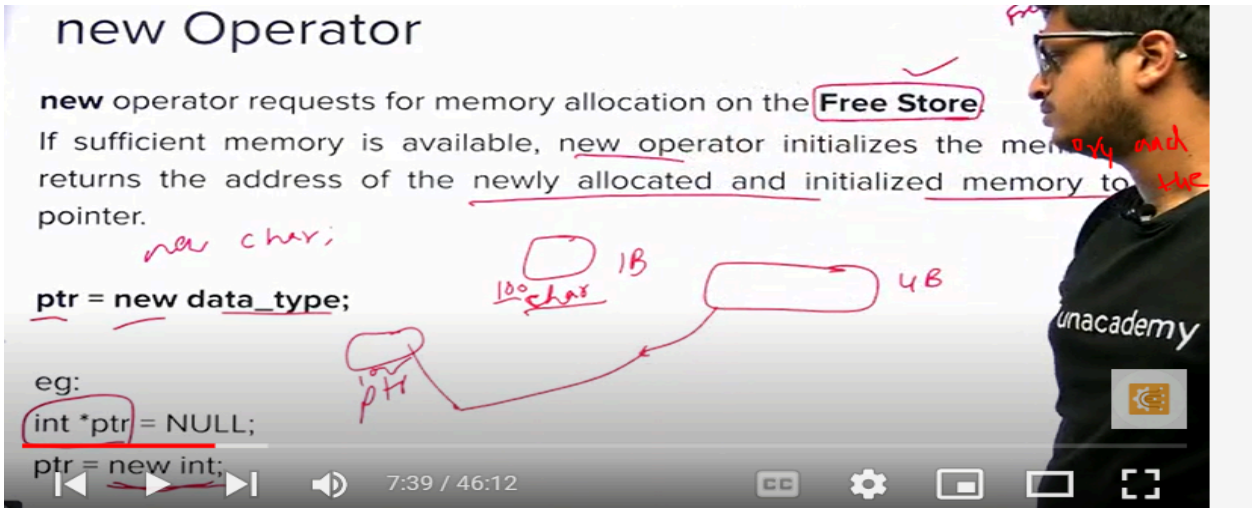
And **Structures, Unions, and Enumerations | C++**

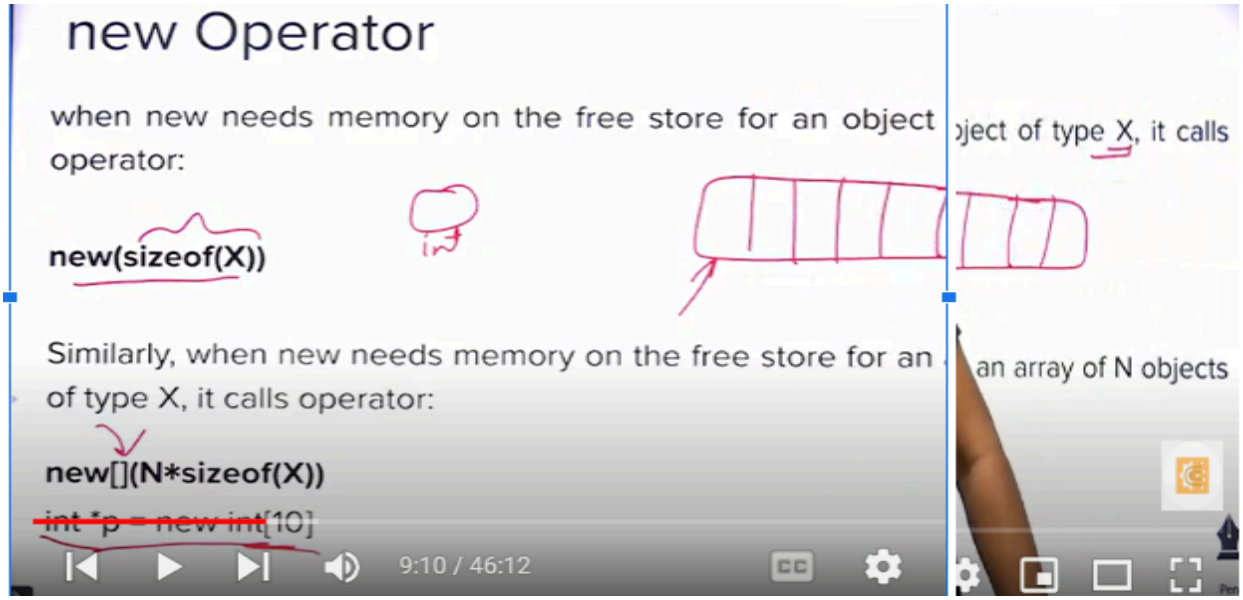


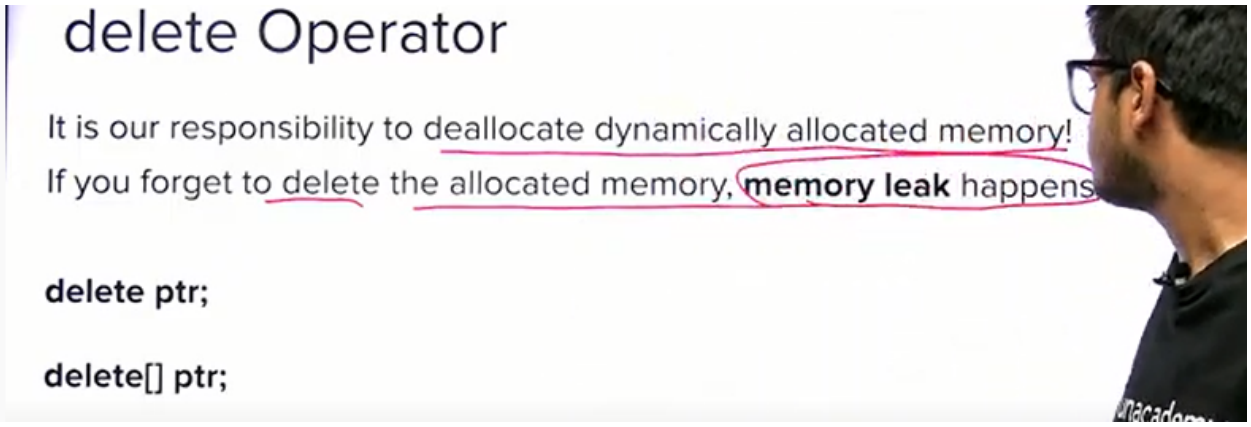

..



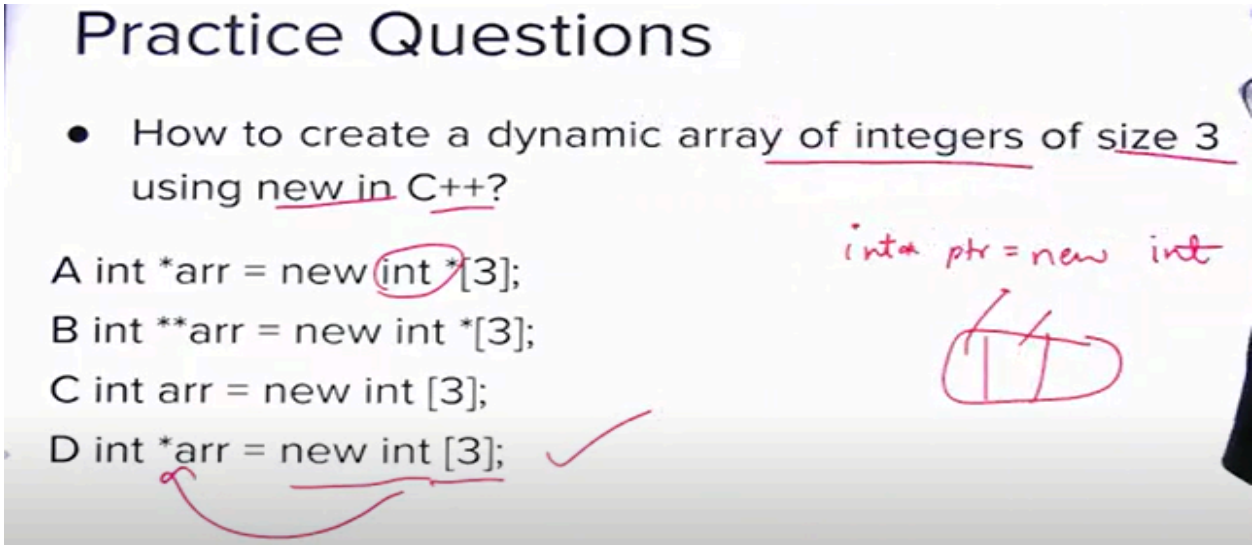

...




new Operator
when new needs memory on the free store for an object
operator:
new(sizeof(X))
Similarly, when new needs memory on the free store for an
of type X, it calls operator:
new[](N*sizeof(X))
int *p = new int[10]
9:10 / 46:12
an array of N objects


delete Operator
It is our responsibility to deallocate dynamically allocated memory!
If you forget to delete the allocated memory, memory leak happens
delete ptr;
delete[] ptr;

..


Practice Questions
How to create a dynamic array of integers of size 3 using new in C++?
A int *arr = new int *[3];
B int **arr = new int *[3];
C int arr = new int [3];
D int *arr = new int [3];
int* ptr = new int

## Practice Questions

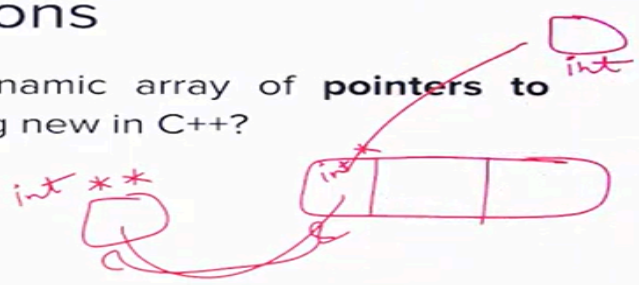
- How to create a dynamic array of **pointers** to integers of size 3 using new in C++?

A int *arr = new int *[3];

B int **arr = new int *[3];

C int arr = new int [3];

D int *arr = new int [3];




14:12 / 46:12

.....

## Practice Questions

- Correct statement(s)?

A new is an operator

B malloc() is a function

C new allocates memory from free store

D malloc() allocates memory from heap

E None of the above

.....

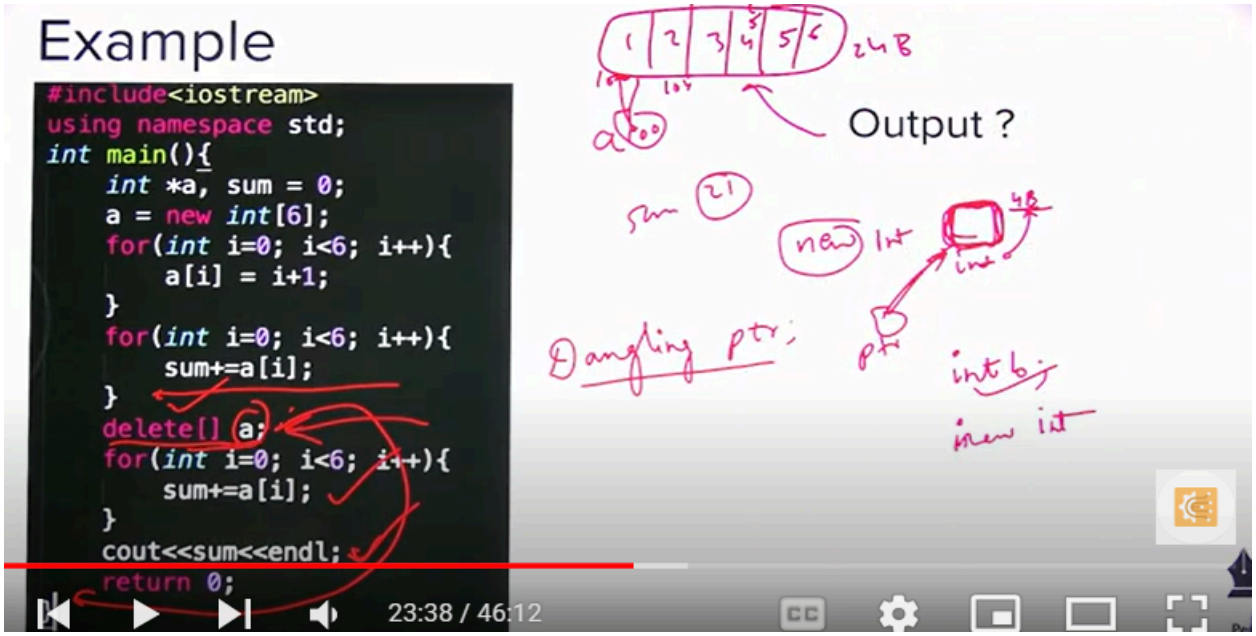
## Example

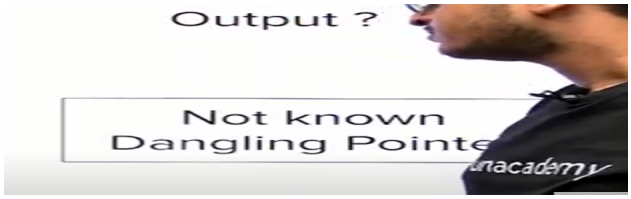
```
#include<iostream>
using namespace std;
int main(){
    int *a, sum = 0;
    a = new int[6];
    for(int i=0; i<6; i++){
        a[i] = i+1;
    }
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    cout<<sum<<endl;
    return 0;
}
```

Output

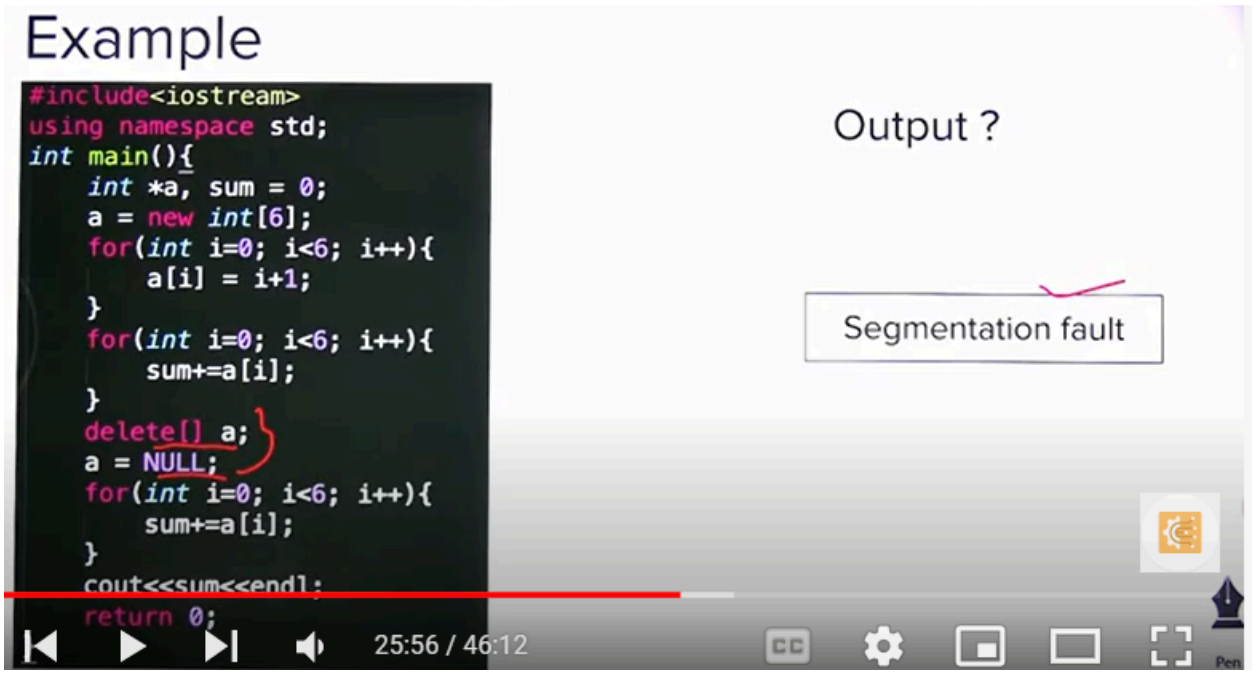
42

18:09 / 46:12


Example
#include<iostream>
using namespace std;
int main(){
int *a, sum = 0;
a = new int[6];
for(int i=0; i<6; i++){
a[i] = i+1;
}
for(int i=0; i<6; i++){
sum+=a[i];
}
delete[] a;
for(int i=0; i<6; i++){
sum+=a[i];
}
cout<<sum<<endl;
return 0;
}
Output ?
23:38 / 46:12


Output ?
Not known
Dangling Pointe

---


Example
#include<iostream>
using namespace std;
int main(){
int *a, sum = 0;
a = new int[6];
for(int i=0; i<6; i++){
a[i] = i+1;
}
for(int i=0; i<6; i++){
sum+=a[i];
}
delete[] a;
a = NULL;
for(int i=0; i<6; i++){
sum+=a[i];
}
cout<<sum<<endl;
return 0;
}
Output ?
Segmentation fault
25:56 / 46:12

.....

# Typecasting

---

## Typecasting

Typecasting is making a variable of one type, such as an int, act like another type, a char, for one single operation

To typecast something, **put the type of variable** you want the actual variable to act as **inside parentheses** in front of the actual variable

```
#include <iostream>
using namespace std;
int main() {
    cout<< (char)66 <<endl;
}
```

## Implicit type conversion in c++

C++ supports other types to typecast as well.

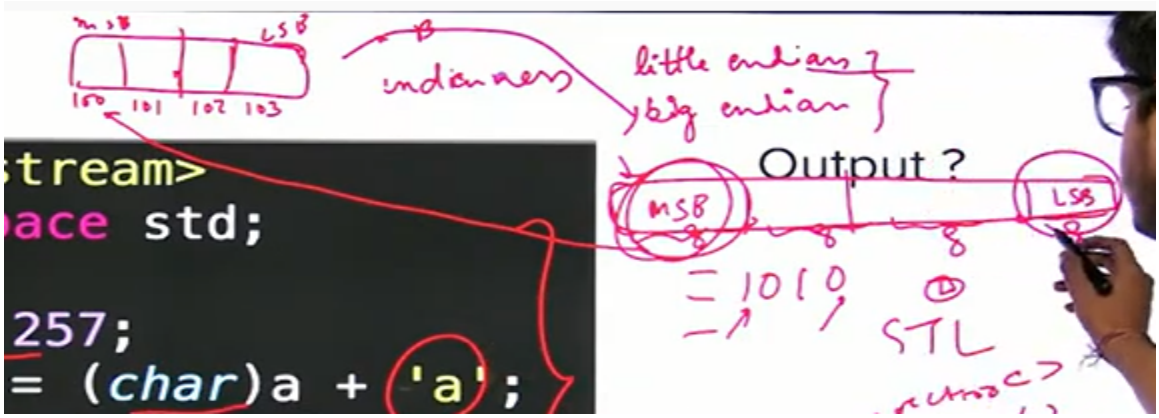
**bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double**

Data loss can occur (Signs, overflow)

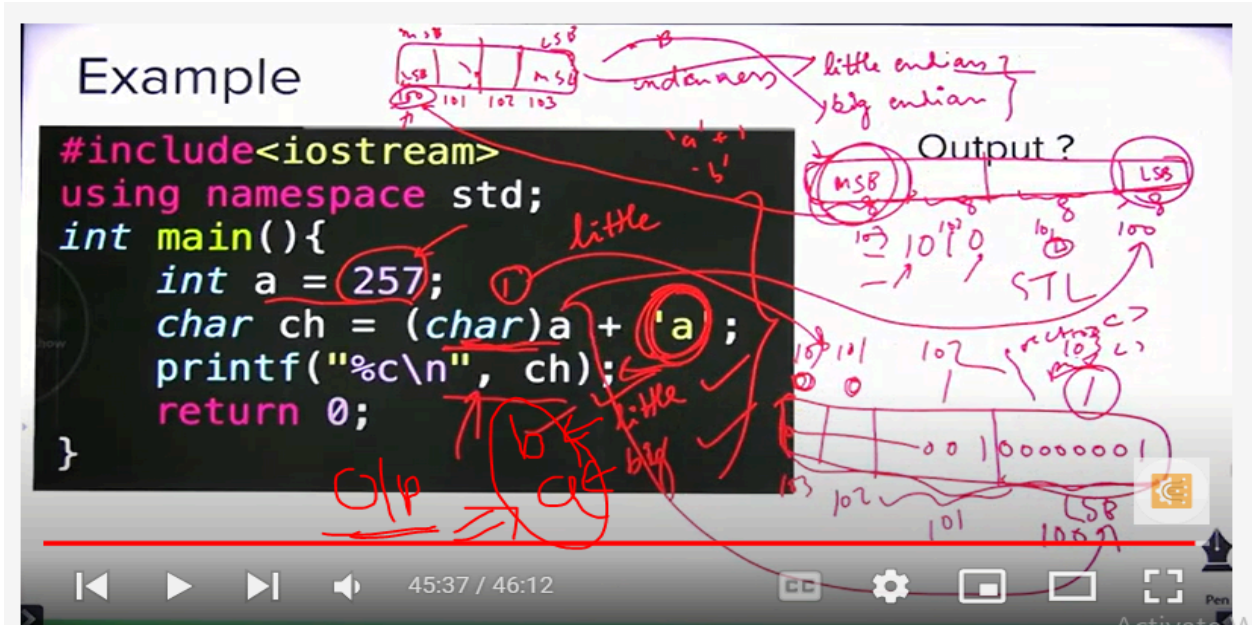
## Practice Questions

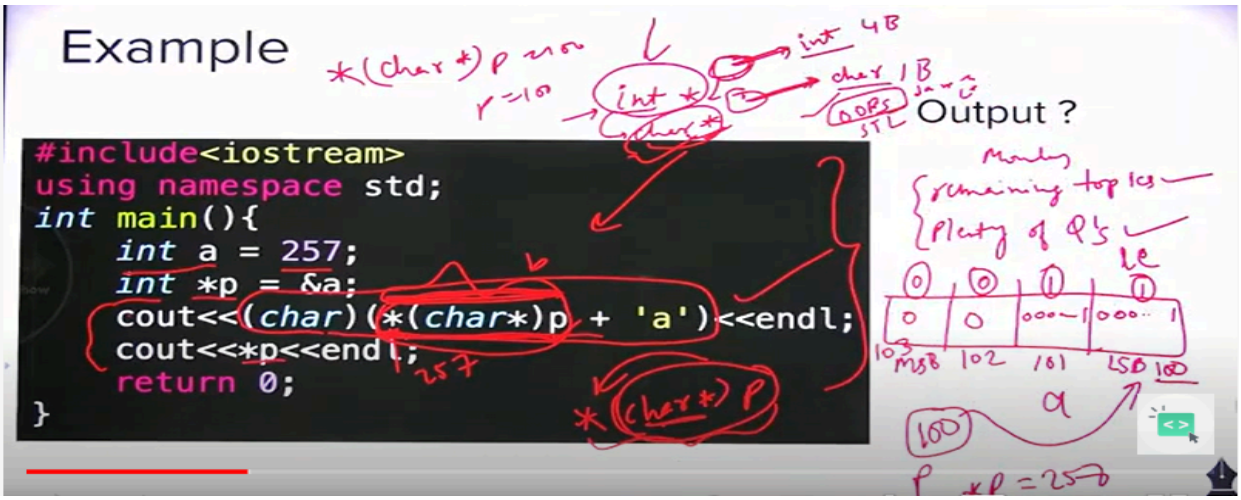
Which of these is/are a valid typecast in C++?

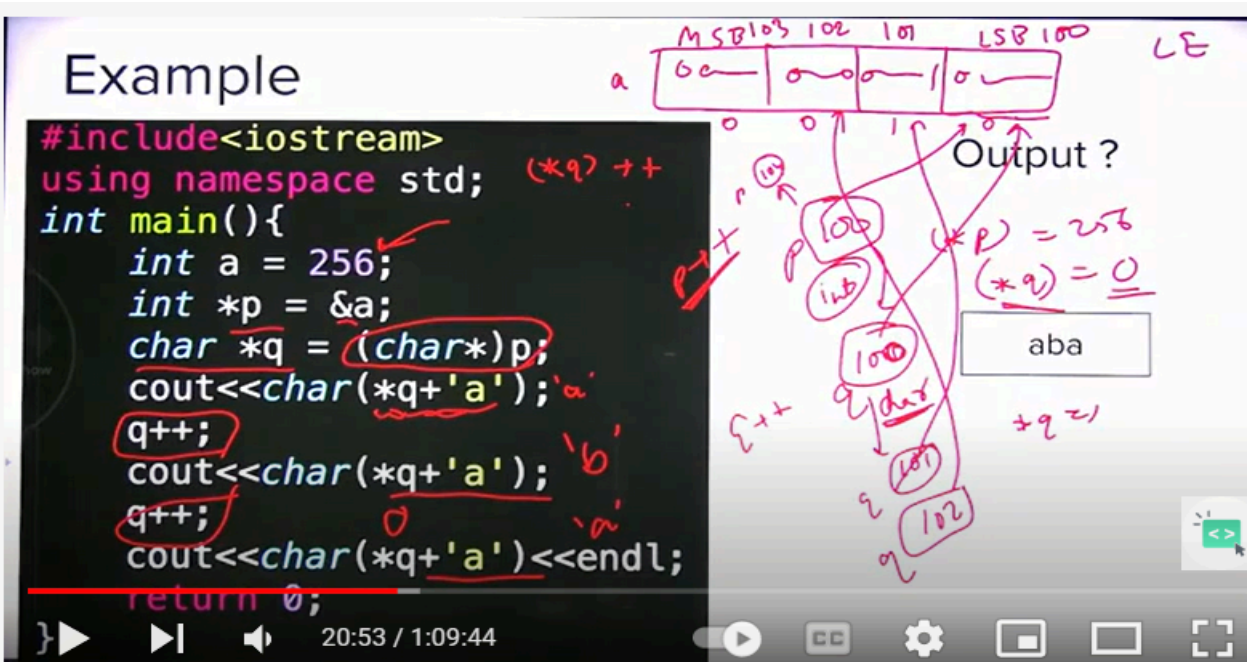
A. a(char);
B. char:a;
C. (char)a;
D. to(char, a);
E. char(a);




Storing the msb in least address ===> big indian
Storing the lsb in least address⇒ small indian



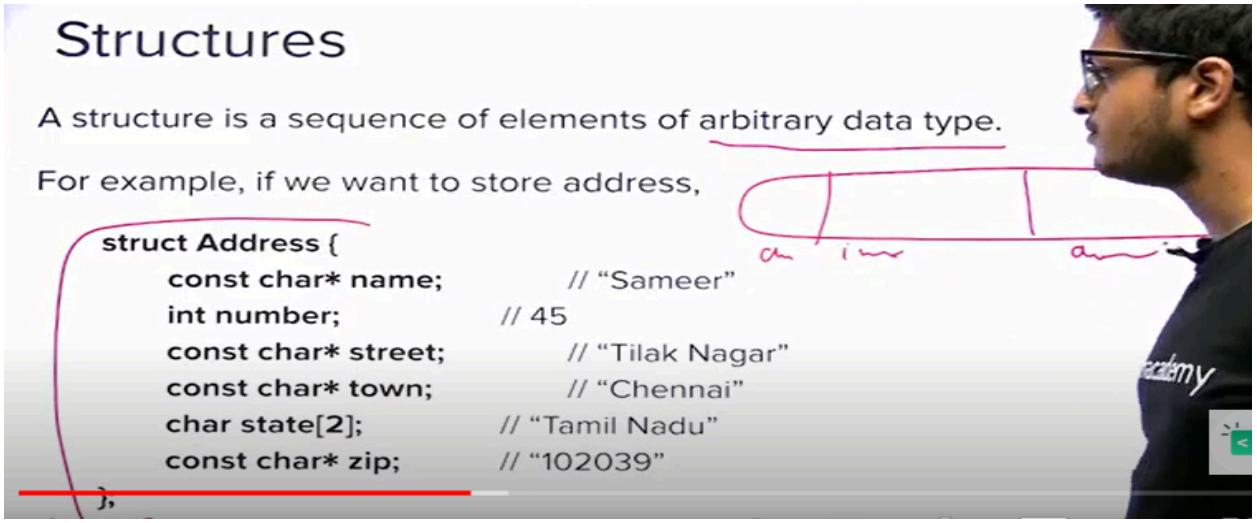



**Output depend on how the value are stored ⇒ little indian or big indian**

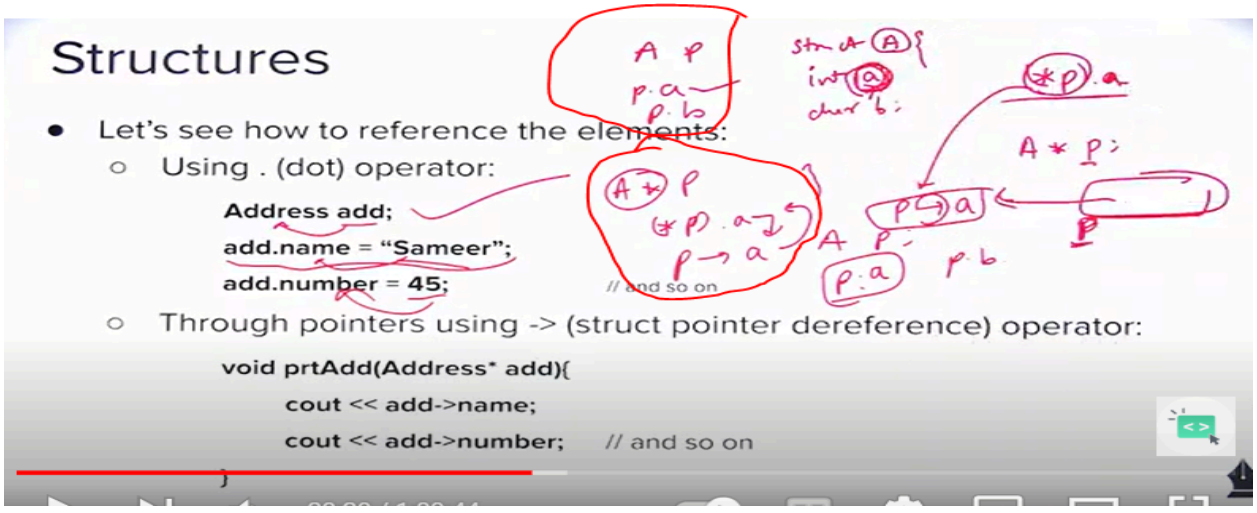
## Structures, Unions, and Enumerations | C++

—----------


.....




.....

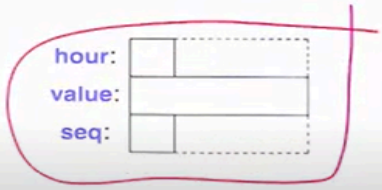


# Structures

- Total size of the structure is not always the sum of the sizes of all data members.

Let's understand this:

```
struct Readout{
    char hour;
    int value;
    char seq;
};
```




So the size of Readout on a 4 byte-int machine would be 12 bytes and not 6 bytes, this is due to the padding added by the compiler to avoid alignment issues.



# Unions

- A **union** is a **struct** in which all members are allocated at the same address so that the union occupies only as much space as its largest member.
- It can only hold value for one member at a time.

For example, if we want to store address,

```
union Address {
    const char* name;       // "Sameer"
    int number;             // 45
    const char* street;     // "Tilak Nagar"
};
```

# Unions

- Like structures, their members can also be accessed by:
  - Using . (dot) operator:

```
Address add;
add.name = "Sameer";
add.number = 45;            // and so on
```

  - Through pointers using -> (union pointer dereference) operator:

```
void prtAdd(Address* add){
    cout << add->name;
    cout << add->number;    // and so on
}
```

# Unions

- The size of a union is taken according the size of largest member in it.

Let's understand this:

```
union Readout{
        char hour;
        int value;
        char seq;
};
```

So the size of Readout on a 4 byte-int machine would be 4 bytes and not 6 bytes, this is due to using same memory for all members.

.....

# Enumerations

- An enum is a type that can hold a set of integer values specified by the user.
- These values are defined at the time of declaring the enumerated type.

For example,

```
enum Color {
    red,
    green,
    blue
};
```

Here red, green and blue are called the enumerators with implicit value as 0, 1 and 2 respectively.

35:53 / 1:09:44

# Enumerations

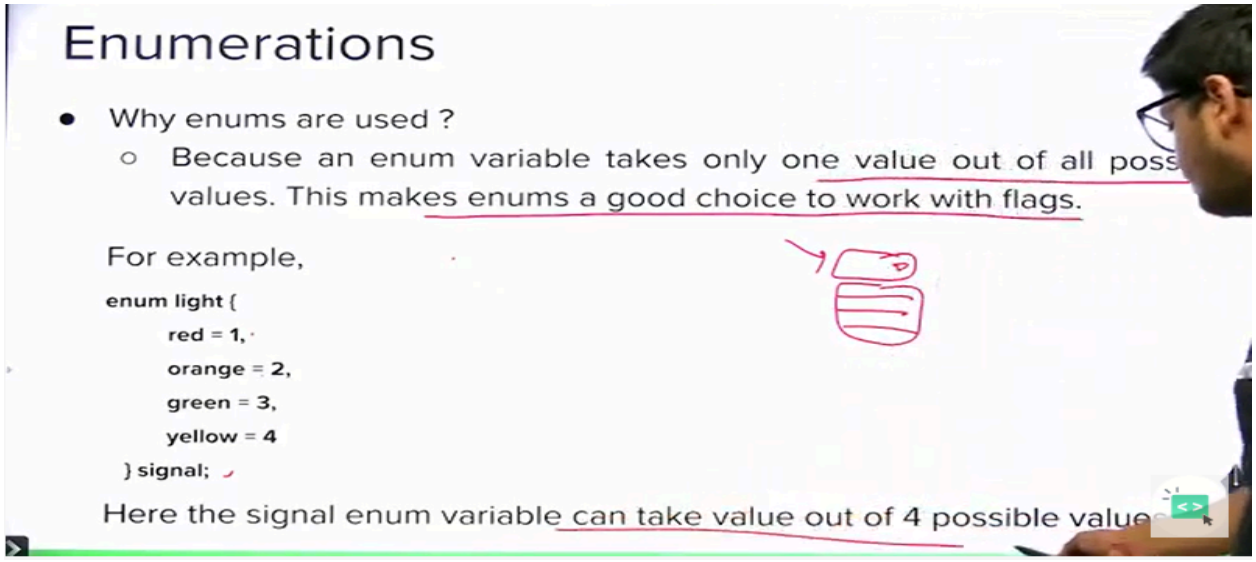
- We can also change the default value of an enum element at the time of declaration.

For example,

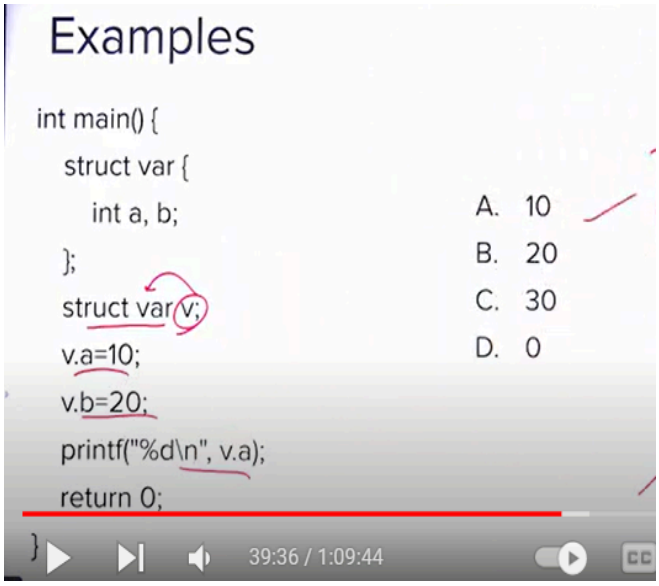
```
enum Color {
    red = 10,
    green = 20,
    blue = 30
};
```

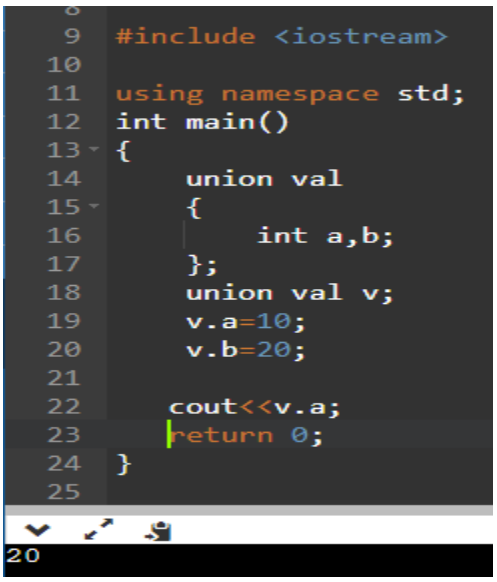
Here red, green and blue have values 10, 20 and 30 respectively.

37:51 / 1:09:44



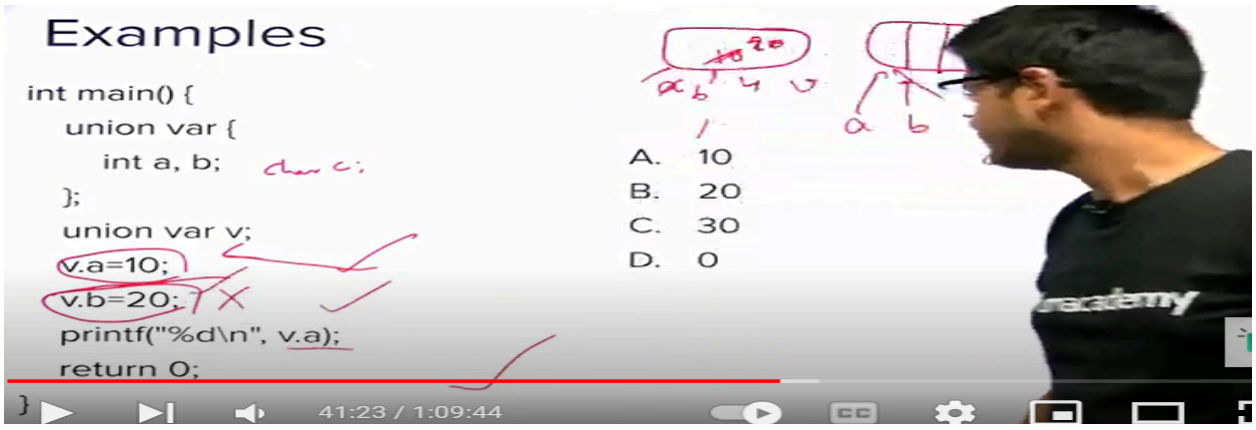

.....




.....

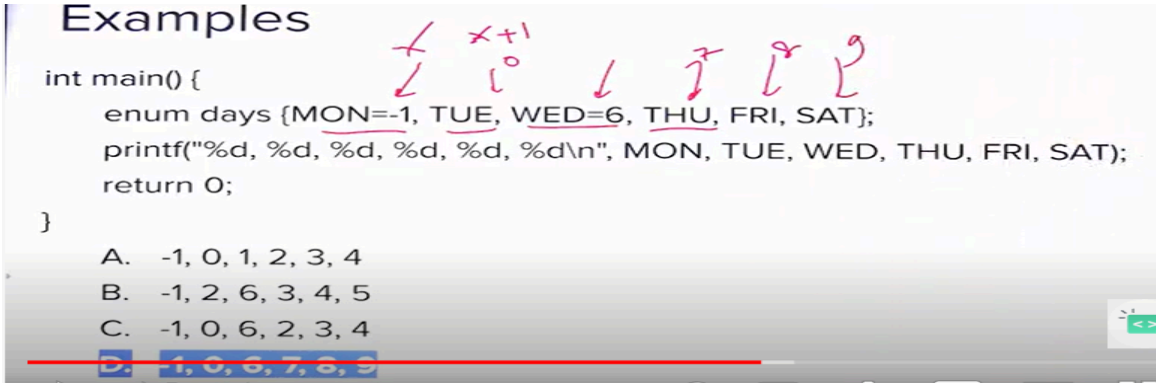
**In union last variable will survive.**



**Examples:**

—----------

**Q1:**




```
13  int main()
14  {
15      enum days{a,b,c=6,d,e,f};
16      cout<<a<<b<<c<<d<<e<<f<<endl;
17
18      return 0;
19  }
```

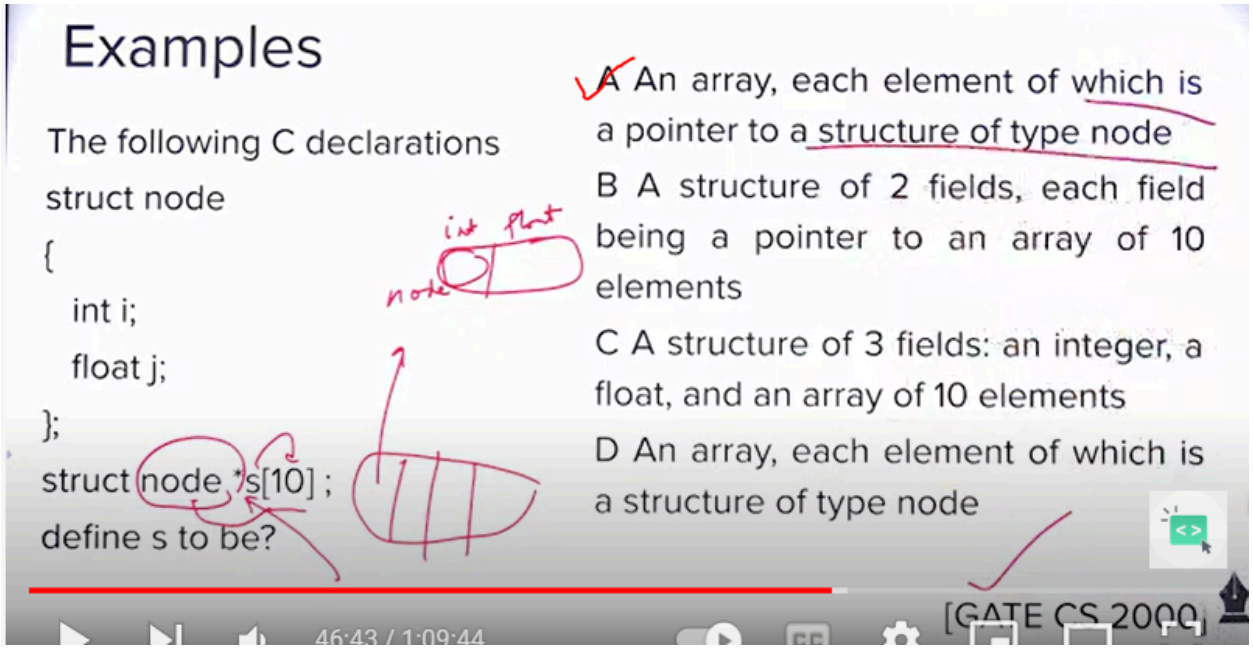
```
016789
```

```
13  int main()
14  {
15      enum days{a=-3,b,c=6,d,e,f};
16      cout<<a<<b<<c<<d<<e<<f<<endl;
17
18      return 0;
19  }
```

```
-3-26789
```

**Next value will be prev_val+1;**
**By default →start from the 0**

**Q2:**



**Q3:**

```
13  int main()
14  {
15      int a=1;
16      int b=1e4,c=1e4;
17      a=(b*c);
18      cout<<a<<endl;
19
```

```
100000000
```

```
13  int main()
14  {
15      int a=1;
16      int b=1e5,c=1e4;
17      a=(b*c);
18      cout<<a<<endl;
19
```

```
1000000000
```

```
13  int main()
14  {
15      int a=1;
16      int b=1e5,c=1e5;
17      a=(b*c);
18      cout<<a<<endl;
19
```

```
1410065408
```

**Int can hold up to 1e9;**
**1e5*1e5⇒overflow for the int**

```
2   using namespace std;
3   int main() {
4       int a;
5       int b = 1e5, c = 1e5;
6       a = b*c;
7       cout<<a<<endl;
8       return 0;
9   }
10  // int ~ 10^9
11  // binary of 10^10
12  // extract LS 4 bytes
```

```
/Users/jaybansal/CLionProjects/CP/cmake-build-debug/CP
1410065408
```

**after 1e9—>don't take int**
**long===> hold up to 1e18**

**—----------**

**Q4:**

```
13  int main()
14  {
15      long a=1;
16      int b=1e5,c=1e5;
17      a=b*c;
18      cout<<a<<endl;
19
```

```
1410065408
```

```
13  int main()
14  {
15      long a=1;
16      int b=1e5,c=1e5;
17      a=(long)(b*c);
18      cout<<a<<endl;
19
```

```
1410065408
```

```
13  int main()
14  {
15      long a=1;
16      int b=1e5,c=1e5;
17      a=(long)b*(long)c;
18      cout<<a<<endl;
19      return 0;
20  }
```

```
10000000000
```

**Long = int*int ⇒ first do the int*int and then the result will be produced and then the result will be stored into the int(largest data type given for the calculation ) and that result will be stored into the long.**