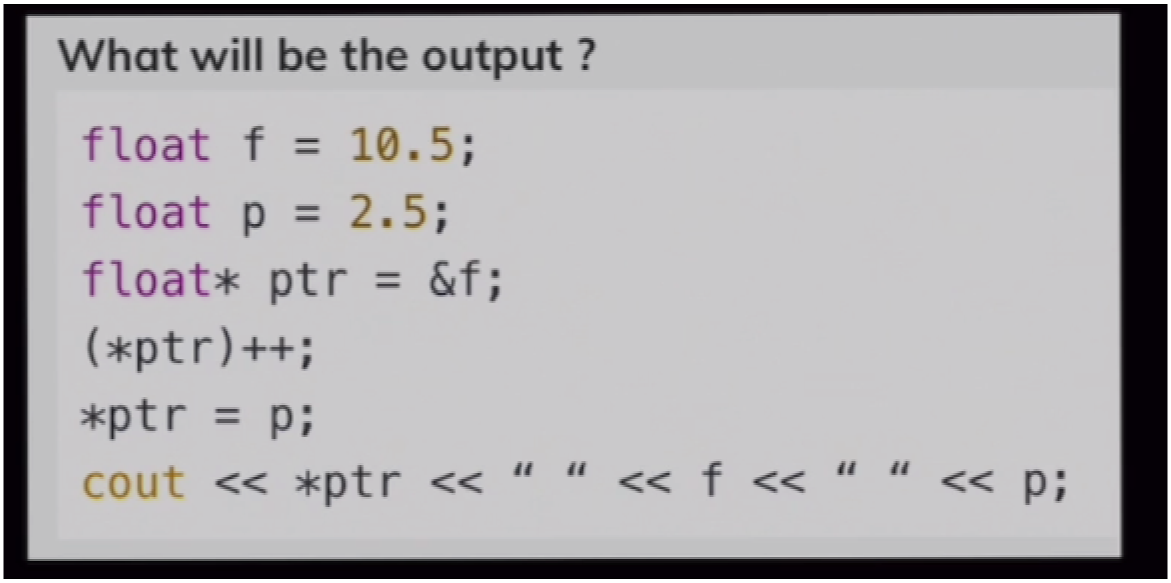
1.

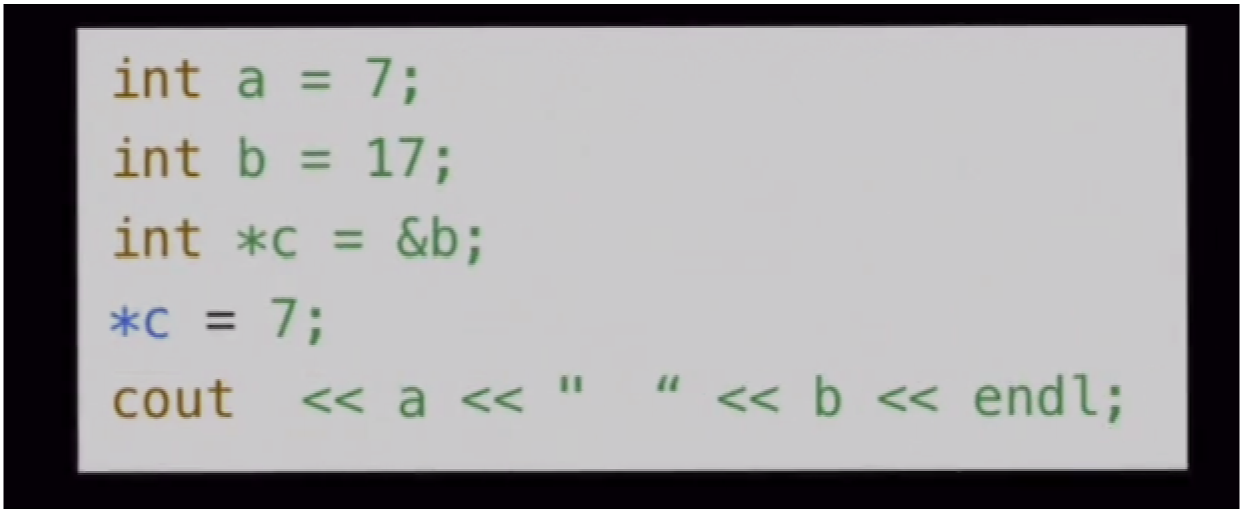


What will be the output ?

```
float f = 10.5;
float p = 2.5;
float* ptr = &f;
(*ptr)++;
*ptr = p;
cout << *ptr << “ “ << f << “ “ << p;
```

O/P: 2.5 2.5 2.5
Explanation: *ptr = 2.5, f or *ptr = 2.5, p = 2.5

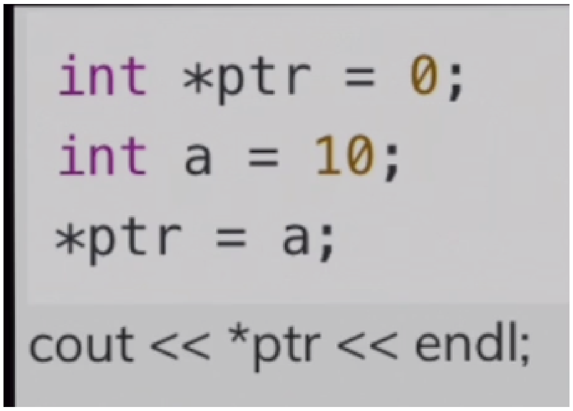
2.



```
int a = 7;
int b = 17;
int *c = &b;
*c = 7;
cout  << a << "  “ << b << endl;
```

O/P: 7 7
Explanation: a = 7, b or *c = 7

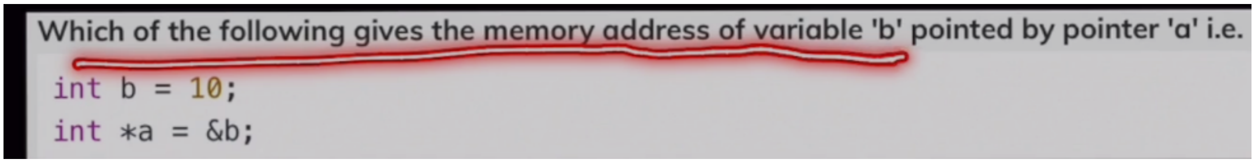
3.



```
int *ptr = 0;
int a = 10;
*ptr = a;
cout << *ptr << endl;
```

O/P: Runtime Error
Explanation: dereferencing a null pointer i.e. *(0x00000000) leads to error and also pointer cannot be pointed to integer value.

4.

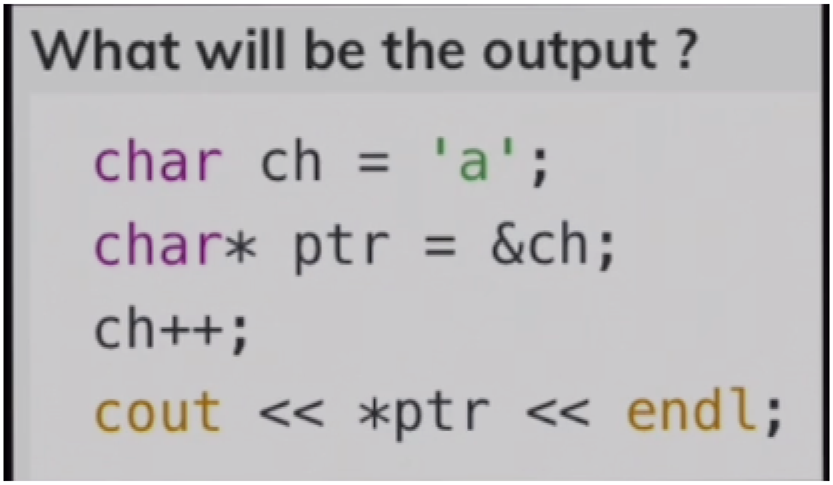


Which of the following gives the memory address of variable 'b' pointed by pointer 'a' i.e.

```
int b = 10;
int *a = &b;
```

Ans: cout << a;
Explanation: a stores the memory address of b.

5.

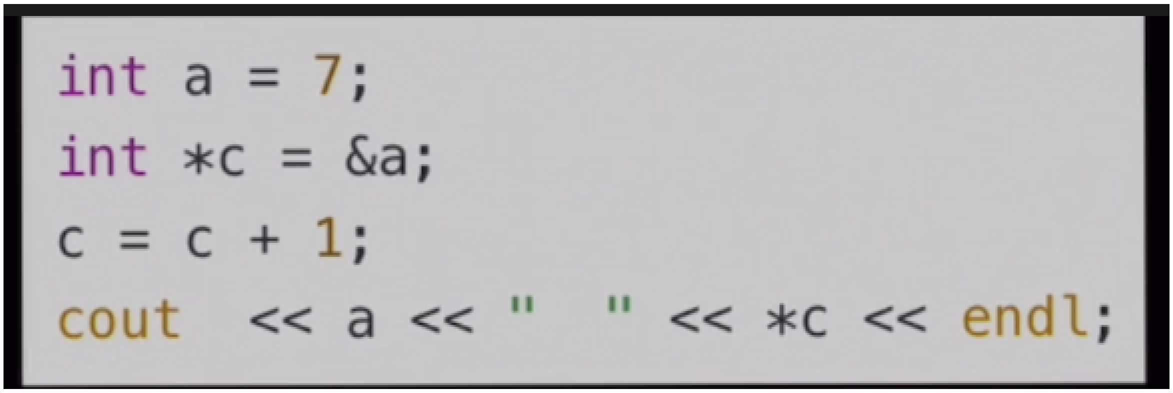


What will be the output ?

```
char ch = 'a';
char* ptr = &ch;
ch++;
cout << *ptr << endl;
```

O/P: b
Explanation: ch++ -> a changes to b i.e in terms of ASCII 97(a) to 98(b).

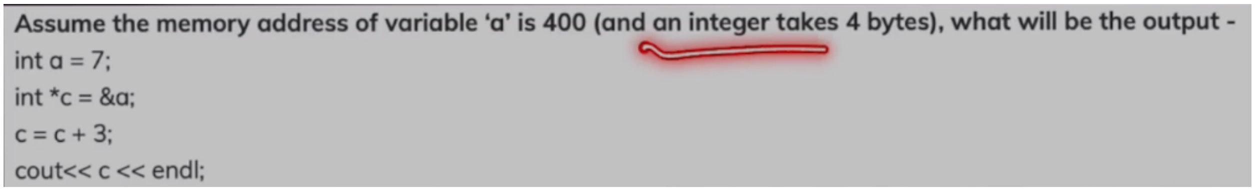
6. Concept: Pointer Arithmetic



```
int a = 7;
int *c = &a;
c = c + 1;
cout  << a << "  " << *c << endl;
```

O/P: a -> 7 ; *c -> Garbage value or value stored in Memory address
Explanation: lets say a memory address is at 300 i.e c = 300. Now c = c+1 = 300 + 1
= 300 + 1 * 4 = 304.

7.

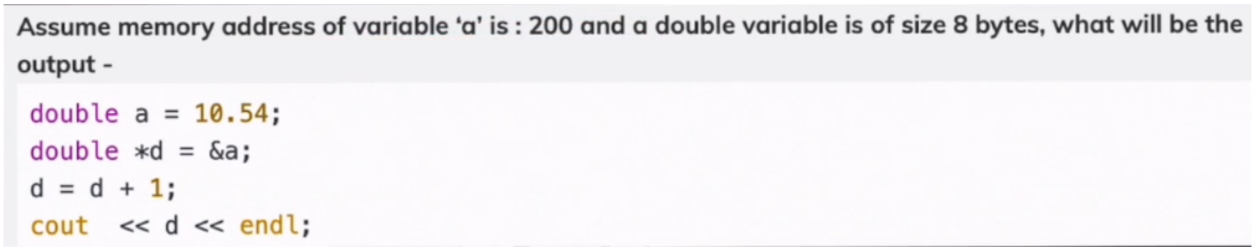


Assume the memory address of variable ‘a’ is 400 (and an integer takes 4 bytes), what will be the output -

```
int a = 7;
int *c = &a;
c = c + 3;
cout<< c << endl;
```

O/P: 412
Explanation: &a = 400, c = 400, c = c+3 = 400+3
= 400 + 3 * 4 (int_size) = 412.

8.

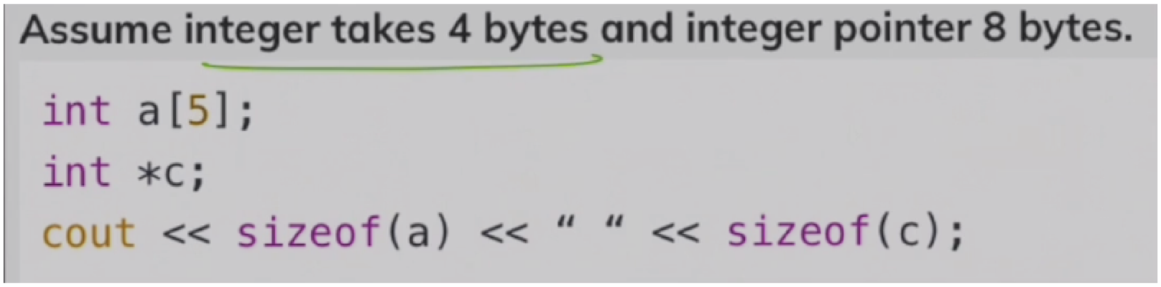


Assume memory address of variable ‘a’ is : 200 and a double variable is of size 8 bytes, what will be the output -

```
double a = 10.54;
double *d = &a;
d = d + 1;
cout  << d << endl;
```

O/P: 208
Explanation: &a = 200, d = 200, d = d + 1 = 200 + 1 = 200 + 1 * 8 (double_size) = 208.

9.

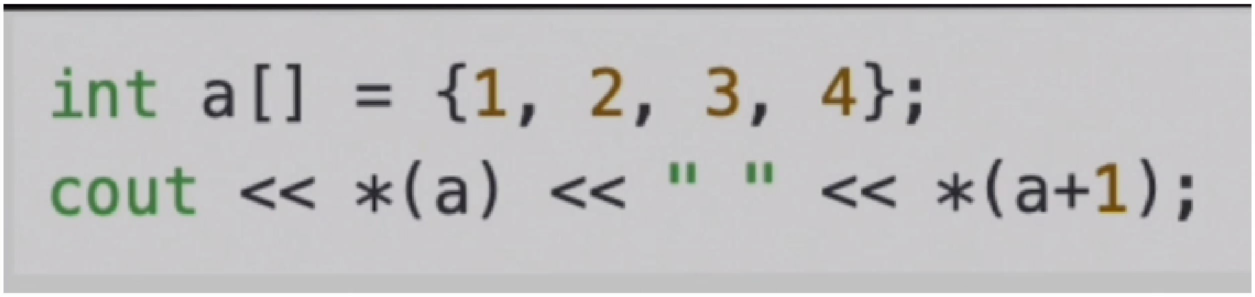


Assume integer takes 4 bytes and integer pointer 8 bytes.

```
int a[5];
int *c;
cout << sizeof(a) << “ “ << sizeof(c);
```

O/P: sizeof(a) -> 20   sizeof(c) -> 8
Explanation: sizeof(a) -> 20 since array consists of 5 integer elements i.e 5 * 4 = 20 bytes and int *c -> pointer to integer has size of 8 bytes irrespective of the data type, cuz memory address is of 8 bytes in all the modern 64 bit systems and pointers only return address of the memory.

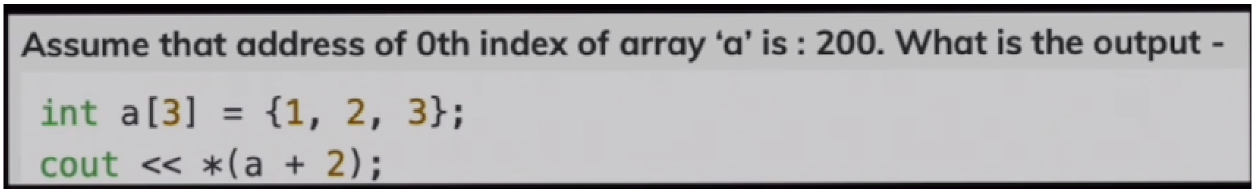
10.



```
int a[] = {1, 2, 3, 4};
cout << *(a) << " " << *(a+1);
```

O/P: 1 2
Explanation: *a = value at base location of a, *(a+1) = *(base_loc + 1)
= (base_loc + 1*4(int_size)) = 2

11.

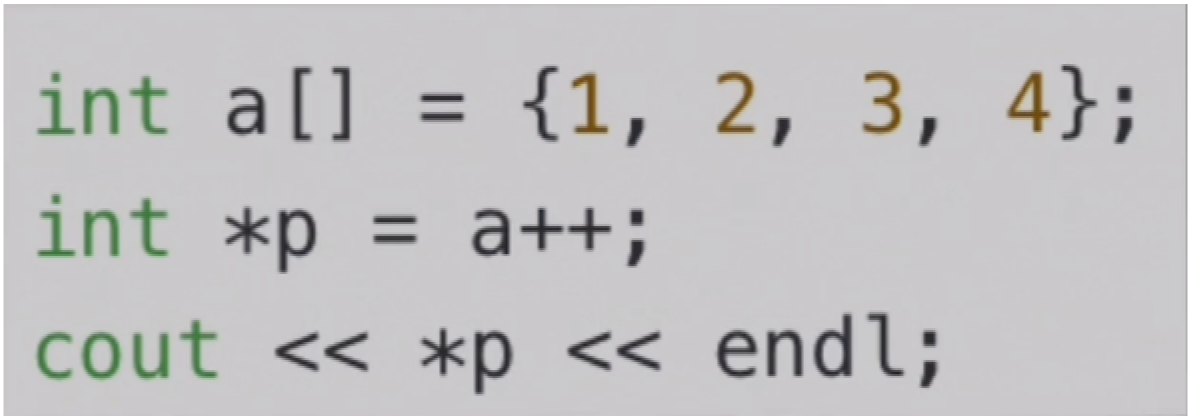


Assume that address of 0th index of array ‘a’ is : 200. What is the output -

```
int a[3] = {1, 2, 3};
cout << *(a + 2);
```

O/P: 3
Explanation: &a[0] = 200, *(a+2) = *(200+2) = *(200+2*4) = *(200+8) = *(208) = 3

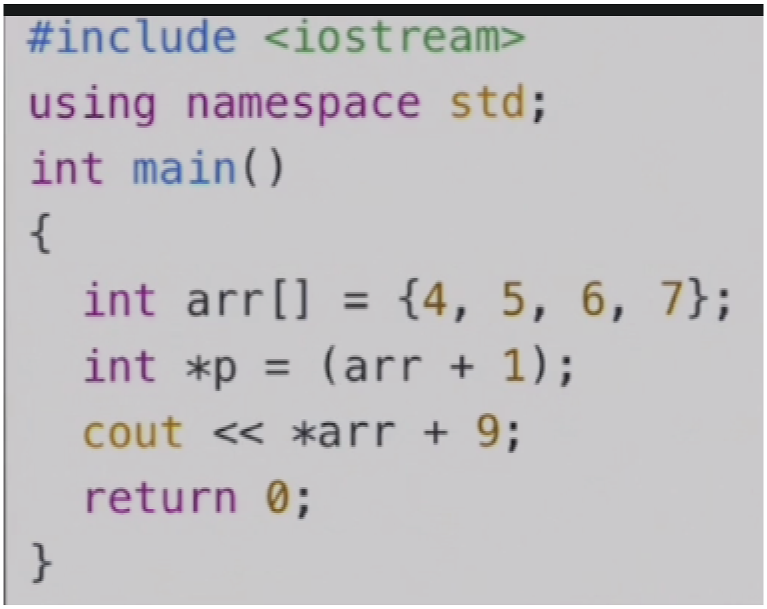
12.



```
int a[] = {1, 2, 3, 4};
int *p = a++;
cout << *p << endl;
```

O/P: Runtime Error
Explanation: a++ -> a = a + 1, here a is a constant pointer thus we can’t modify it.

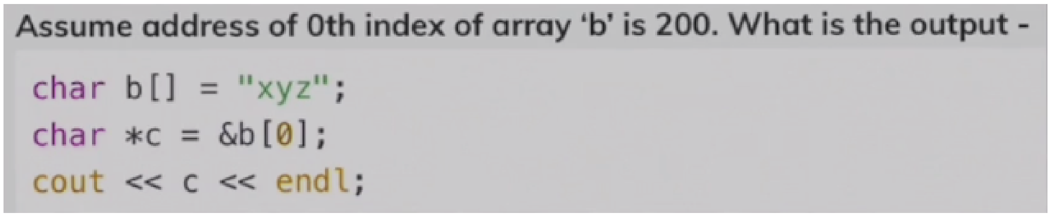
13.



```
#include <iostream>
using namespace std;
int main()
{
   int arr[] = {4, 5, 6, 7};
   int *p = (arr + 1);
   cout << *arr + 9;
   return 0;
}
```

O/P: 13
Explanation: *p = base_loc + 1 * 4(int_size), *arr = value at base_loc = 4, 4 + 9 = 13

14. Cout implementation in character array is different. i.e cout prints everything till null character is found.



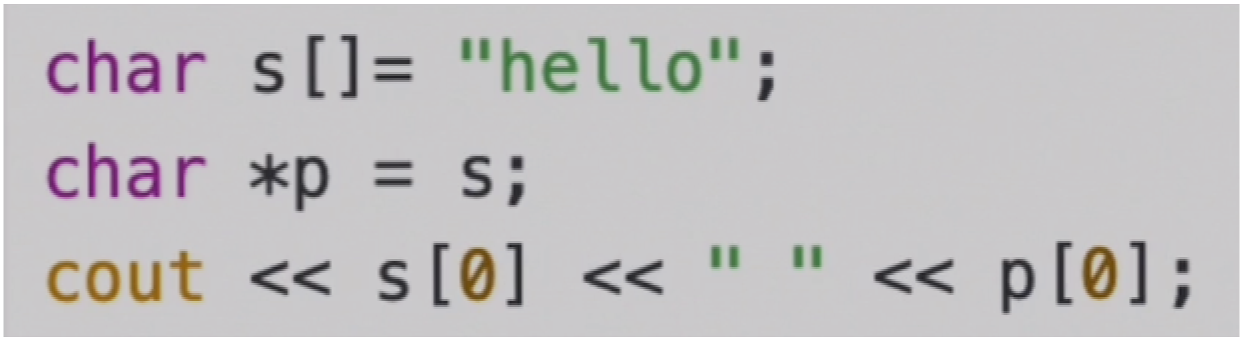
Assume address of 0th index of array 'b' is 200. What is the output -

```
char b[] = "xyz";
char *c = &b[0];
cout << c << endl;
```

O/P: xyz

Explanation: cout << b -> prints xyz, here b is base address similarity c is also base address therefore cout << c -> prints xyz

15.

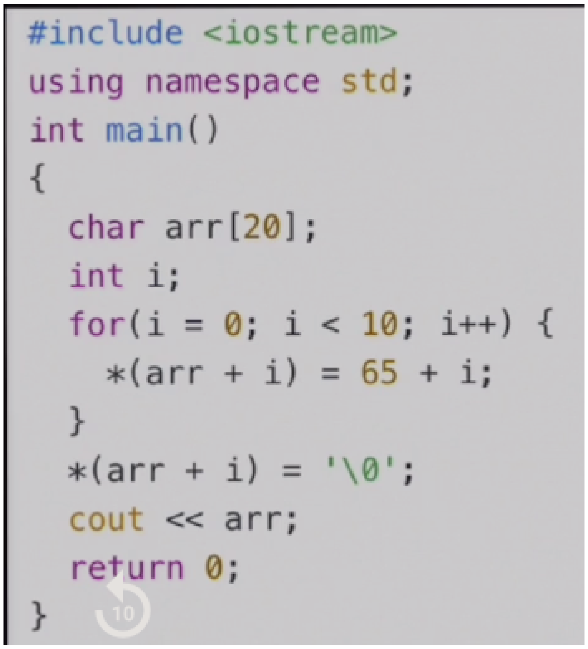


```
char s[]= "hello";
char *p = s;
cout << s[0] << " " << p[0];
```

O/P: h h

Explanation: *p = s -> s is base address, thus p[0] = h.

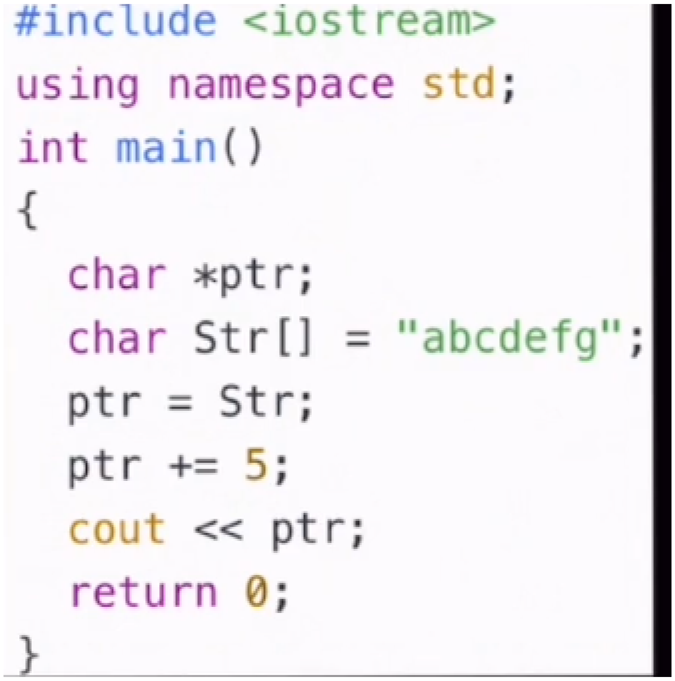
16.



```
#include <iostream>
using namespace std;
int main()
{
   char arr[20];
   int i;
   for(i = 0; i < 10; i++) {
     *(arr + i) = 65 + i;
   }
   *(arr + i) = '\0';
   cout << arr;
   return 0;
}
```

O/P: ABCDEFGHIJ
Explanation: for loops runs till i = 9 i.e when i = 10 it terminates then using *(arr + i) = ‘ \0 ‘ where i = 10, we assign that location a null character and print the arr.

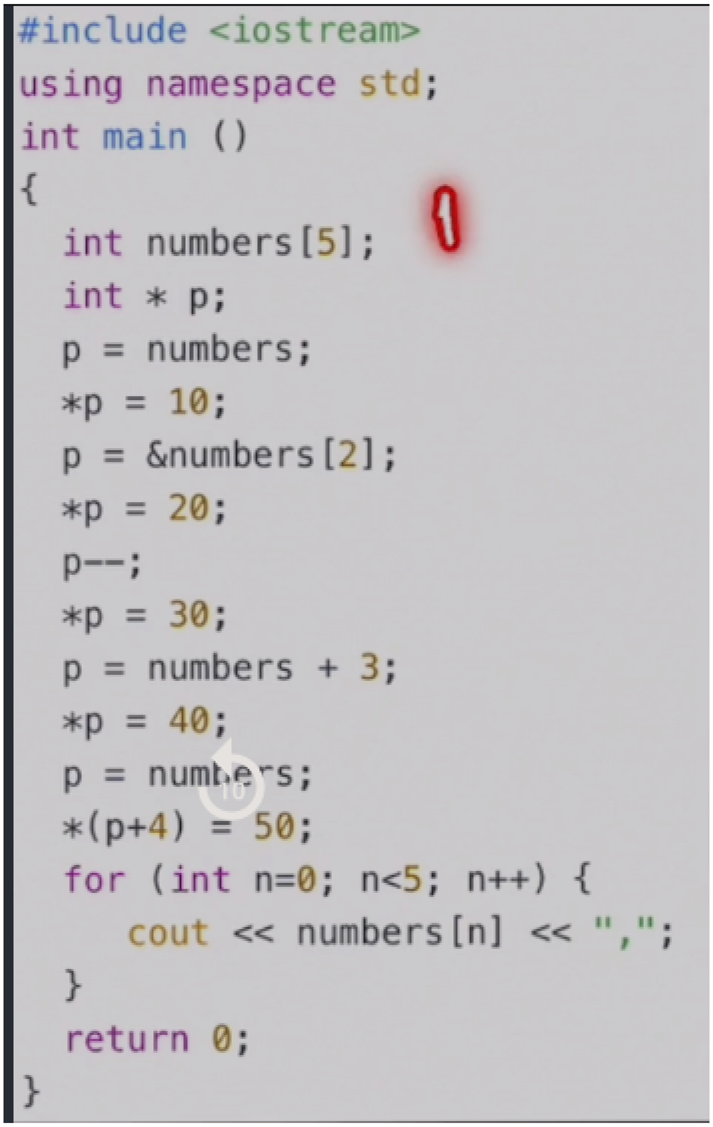
17.



```
#include <iostream>
using namespace std;
int main()
{
   char *ptr;
   char Str[] = "abcdefg";
   ptr = Str;
   ptr += 5;
   cout << ptr;
   return 0;
}
```

O/P: fg
Explanation: ptr = str // base addr, ptr += 5 -> f, then cout << ptr, cout prints everything from 5th index element till null character is found.

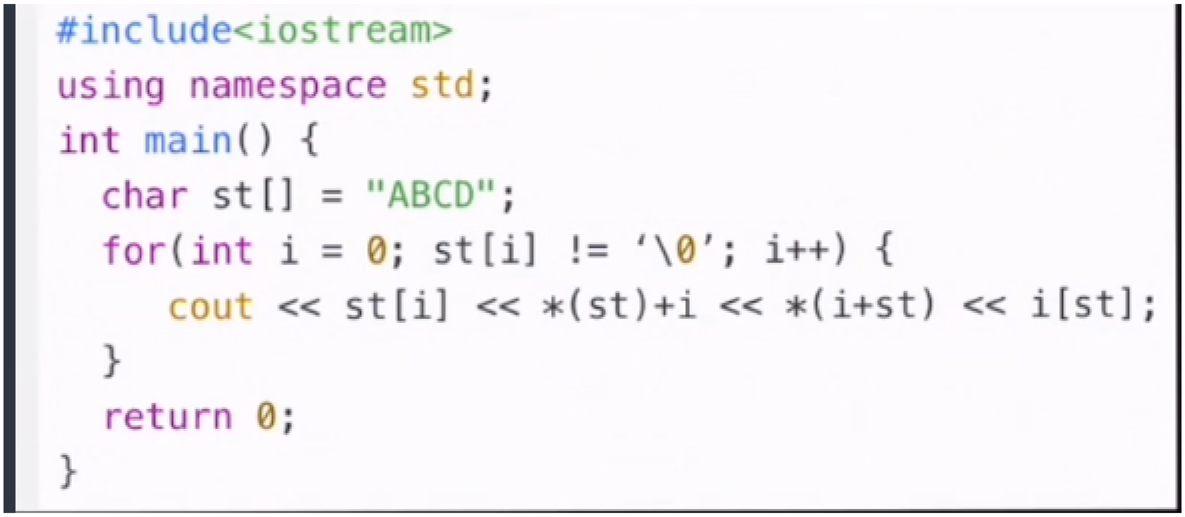
18.



```
#include <iostream>
using namespace std;
int main ()
{
   int numbers[5];
   int * p;
   p = numbers;
   *p = 10;
   p = &numbers[2];
   *p = 20;
   p--;
   *p = 30;
   p = numbers + 3;
   *p = 40;
   p = numbers;
   *(p+4) = 50;
   for (int n=0; n<5; n++) {
      cout << numbers[n] << ",";
   }
   return 0;
}
```

O/P: 10 30 20 40 50
Explanation:

19.



```
#include<iostream>
using namespace std;
int main() {
  char st[] = "ABCD";
  for(int i = 0; st[i] != ‘\0’; i++) {
     cout << st[i] << *(st)+i << *(i+st) << i[st];
  }
  return 0;
}
```

O/P: A65AAB66BBC67CCD68DD
Explanation:

For i = 0:

st[i] = A
*(st) + i = ‘A’ + 0 = 65 + 0 = 65 ; // if we add character to int and then cout it directly we get answer in integer. But if we store ‘A’ + 0 in a char var and then cout it then we get A
*(i + st) = *(0 + st) = *(st) = A
i[st] = A since i[st] = st[i]

For i = 1:

st[i] = B
*(st) + i = ‘A’ + 1 = 65 + 1 = 66
*(i + st) = *(1 + st) = *(1 + st) = B
i[st] = B; // since i[st] = st[i]

.
.

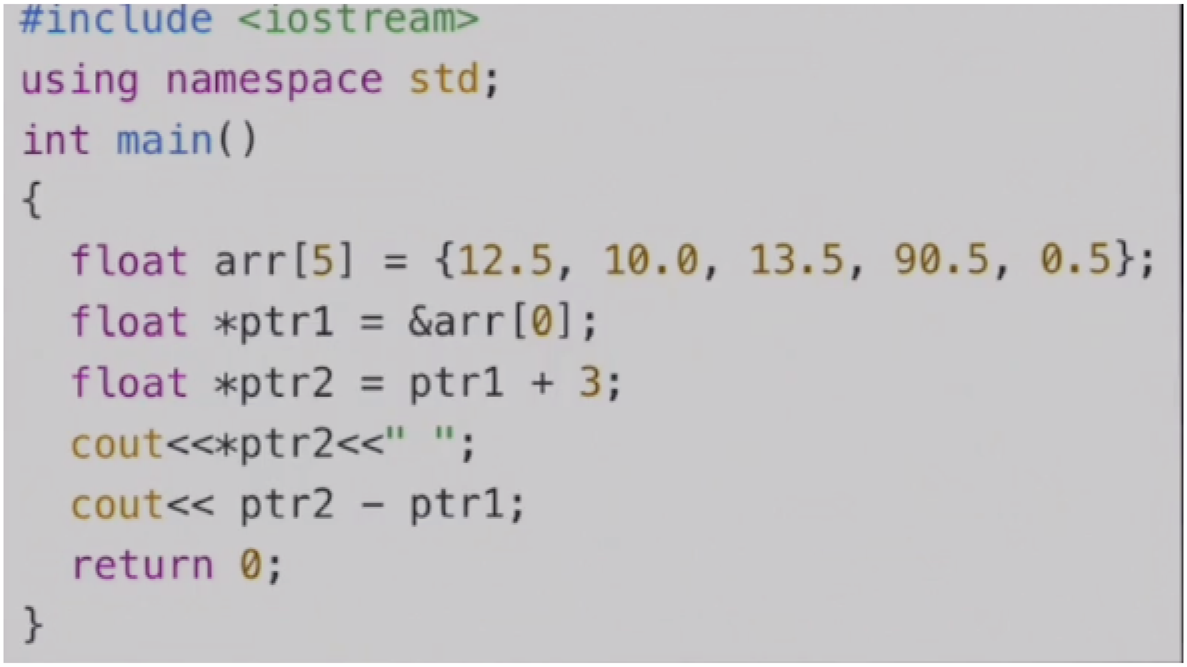
For i = 2:

st[i] = C
*(st) + i = ‘A’ + 2 = 65 + 2 = 67
*(i + st) = *(2 + st) = *(2 + st) = C
i[st] = C; // since i[st] = st[i]

For i = 3:

st[i] = D
*(st) + i = ‘A’ + 3 = 65 + 3 = 68
*(i + st) = *(3 + st) = *(3 + st) = D
i[st] = D; // since i[st] = st[i]

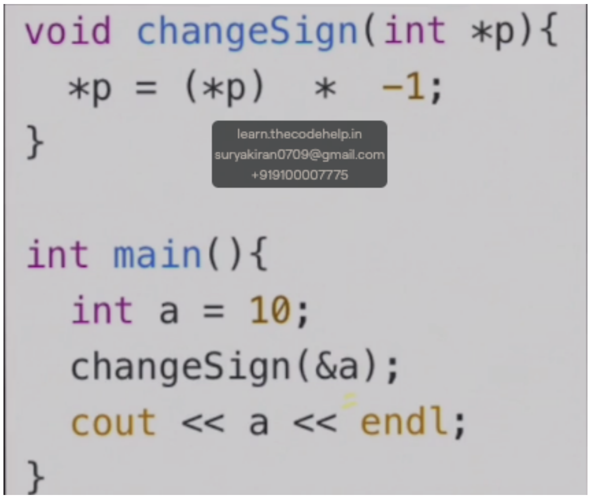
20.



```
#include <iostream>
using namespace std;
int main()
{
  float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
  float *ptr1 = &arr[0];
  float *ptr2 = ptr1 + 3;
  cout<<*ptr2<<" ";
  cout<< ptr2 - ptr1;
  return 0;
}
```

O/P: *ptr2 -> 90.5, ptr2 - ptr1 -> ptr2EleAddress - ptr1EleAddress = 3
Explanation: lets say &arr = 200, &arr[3] = 212, ptr1 = 200, ptr2 = 212
ptr2 - ptr1 -> ptr2EleAddress - ptr1EleAddress = 212 - 200 = 12 bytes -> 12 / 4 = 3 (Pointer Arithmetic). Since 1 Address is 4 bytes.

21.

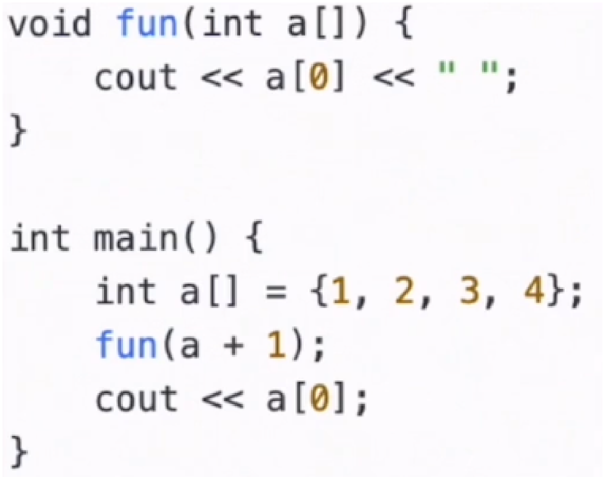


```
void changeSign(int *p){
  *p = (*p)  *  -1;
}

int main(){
  int a = 10;
  changeSign(&a);
  cout << a << endl;
}
```

O/P: -10
Explanation: pass by reference

22.

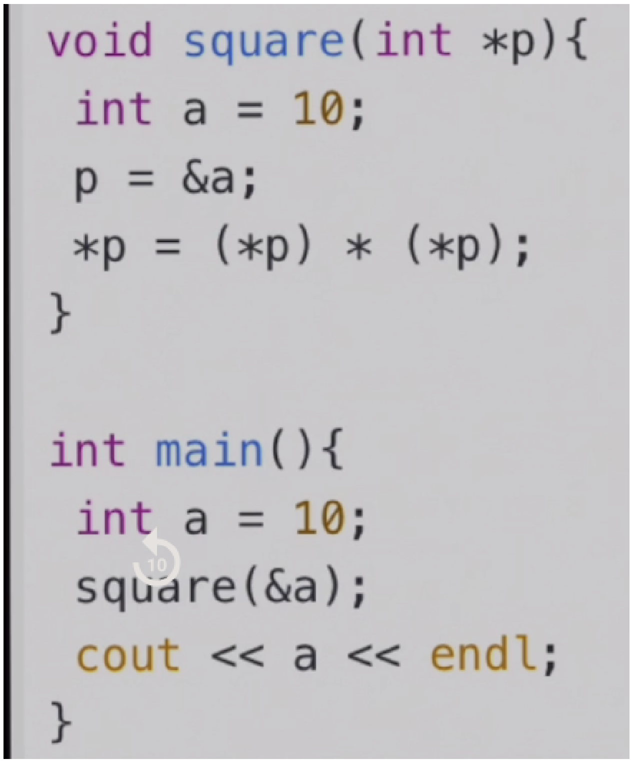


```
void fun(int a[]) {
    cout << a[0] << " ";
}

int main() {
    int a[] = {1, 2, 3, 4};
    fun(a + 1);
    cout << a[0];
}
```

O/P: 2 1
Explanation: fun(a+1) -> baseAddr + 1 = element 2 of 1st index.
fun(int arr[]) -> arr = {2} ; arr[0] = 2.

23.

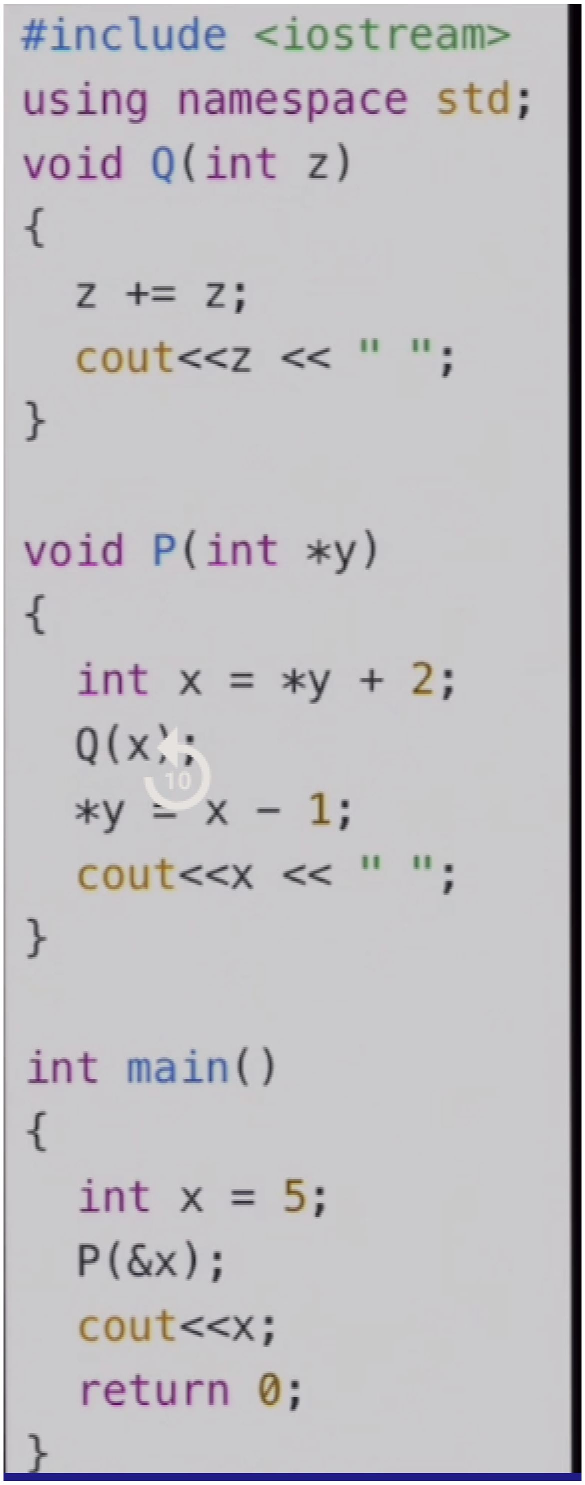


```
void square(int *p){
 int a = 10;
 p = &a;
 *p = (*p) * (*p);
}

int main(){
 int a = 10;
 square(&a);
 cout << a << endl;
}
```

O/P: 10
Explanation:

24.



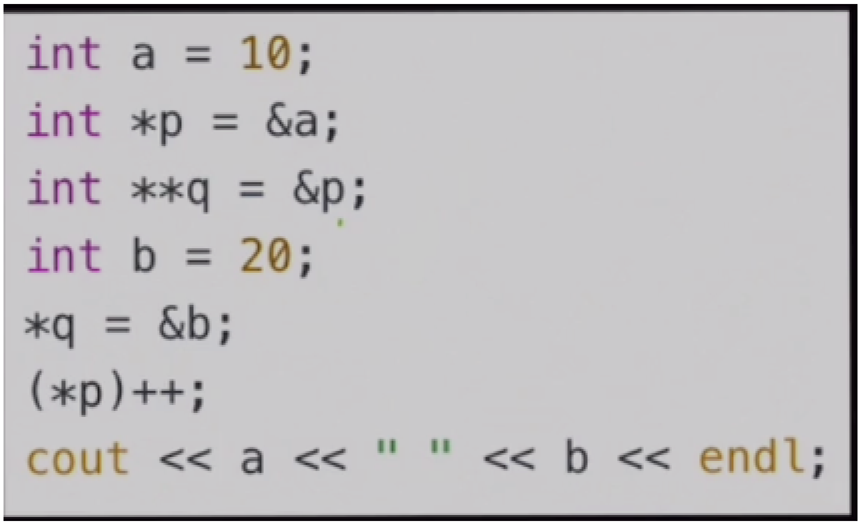
```
#include <iostream>
using namespace std;
void Q(int z)
{
   z += z;
   cout<<z << " ";
}

void P(int *y)
{
   int x = *y + 2;
   Q(x);
   *y = x - 1;
   cout<<x << " ";
}

int main()
{
   int x = 5;
   P(&x);
   cout<<x;
   return 0;
}
```

O/P: 14 7 6
Explanation:

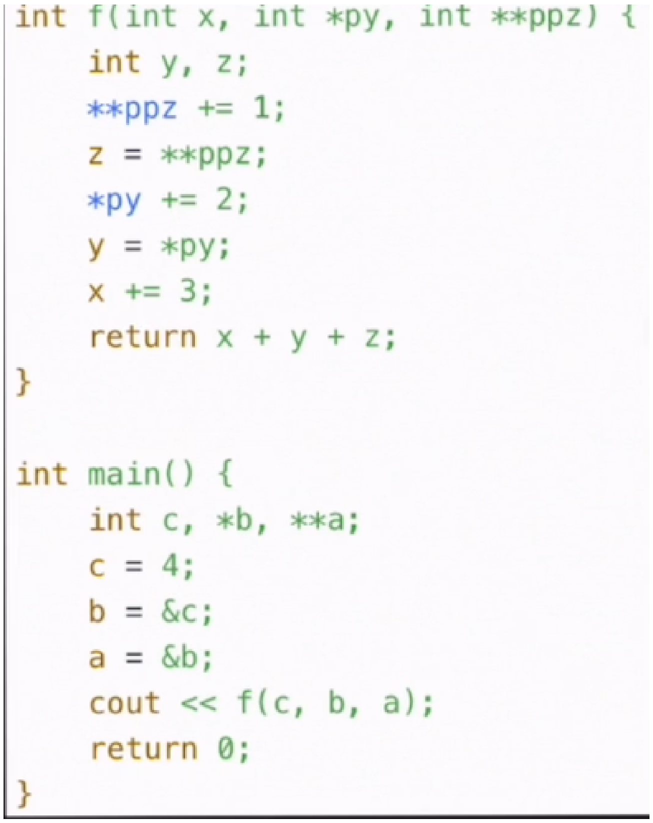
25.



```
int a = 10;
int *p = &a;
int **q = &p;
int b = 20;
*q = &b;
(*p)++;
cout << a << " " << b << endl;
```

O/P: 10 21
Explanation:

26.

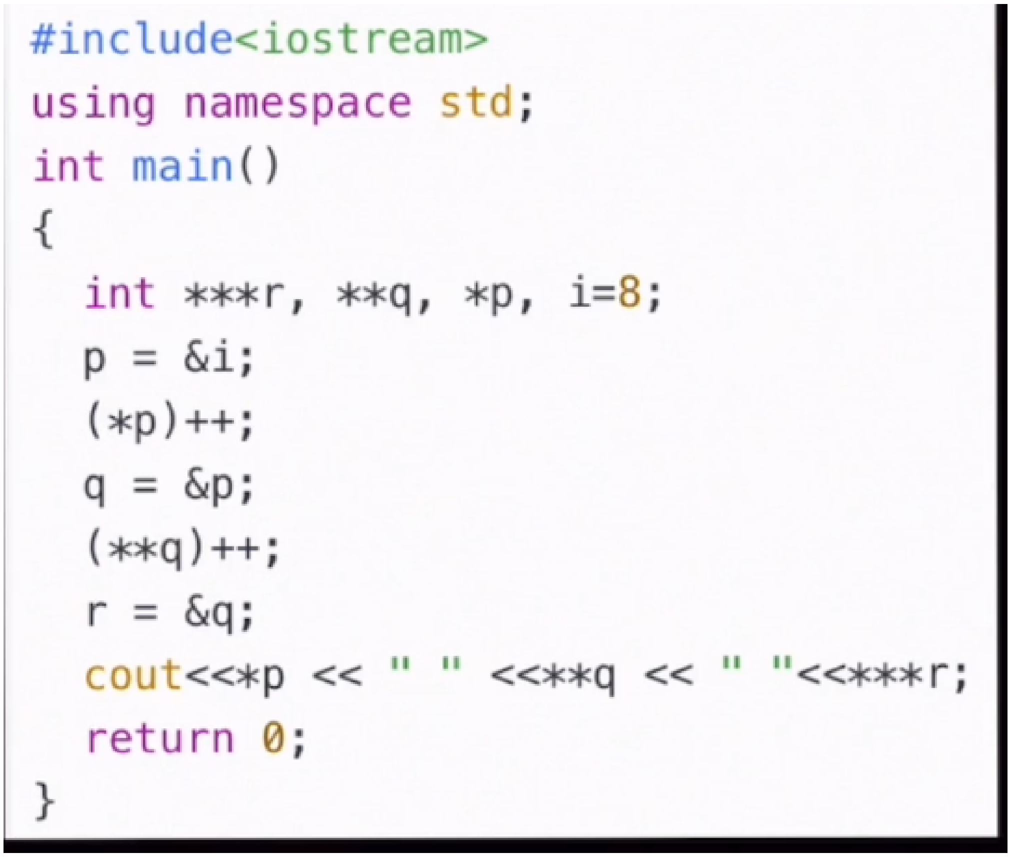


```
int f(int x, int *py, int **ppz) {
    int y, z;
    **ppz += 1;
    z = **ppz;
    *py += 2;
    y = *py;
    x += 3;
    return x + y + z;
}

int main() {
    int c, *b, **a;
    c = 4;
    b = &c;
    a = &b;
    cout << f(c, b, a);
    return 0;
}
```

O/P: 19
Explanation:

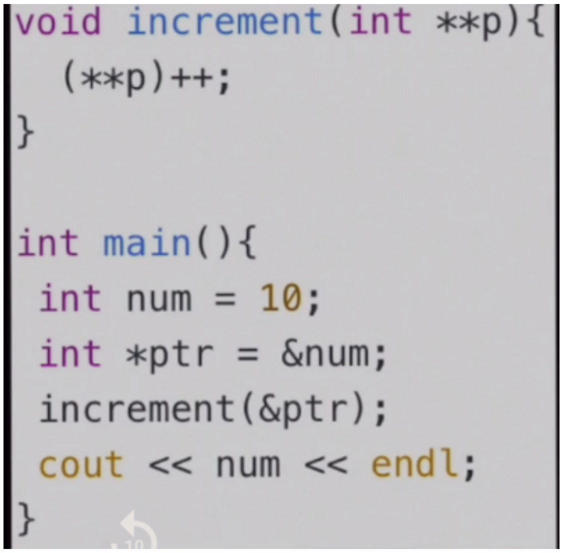
27.



```
#include<iostream>
using namespace std;
int main()
{
   int ***r, **q, *p, i=8;
   p = &i;
   (*p)++;
   q = &p;
   (**q)++;
   r = &q;
   cout<<*p << " " <<**q << " "<<***r;
   return 0;
}
```

O/P: 10 10 10
Explanation:

28.



```
void increment(int **p){
   (**p)++;
}

int main(){
 int num = 10;
 int *ptr = &num;
 increment(&ptr);
 cout << num << endl;
}
```

O/P: 11
Explanation: