Q1 The auto keyword can be used in place of 'int', 'double' or other data types to automatically determine the data type of the variable.
[Automatic type deduction].

Q2 auto a = *y and auto &b = *y
will return errors. Since y is of type int, and * is used to return value of an address. y is not an address ⟶ Error.
auto b = &y . Here b has type ⟶ int*.
& is a pointer to an integer.

Q3 Size of struct is sum of sizes of its members, but sometimes larger because of padding.

(a) double ⟶ 8 bytes
int ⟶ 4
char ⟶ 1
char* ⟶ 4 . [Address]
17

With padding, it will be greater than this, a multiple of 8. ⟶ 24.

(b) int ⟶ 4
double* ⟶ 4 [Address]
char* ⟶ 4
char ⟶ 1
13

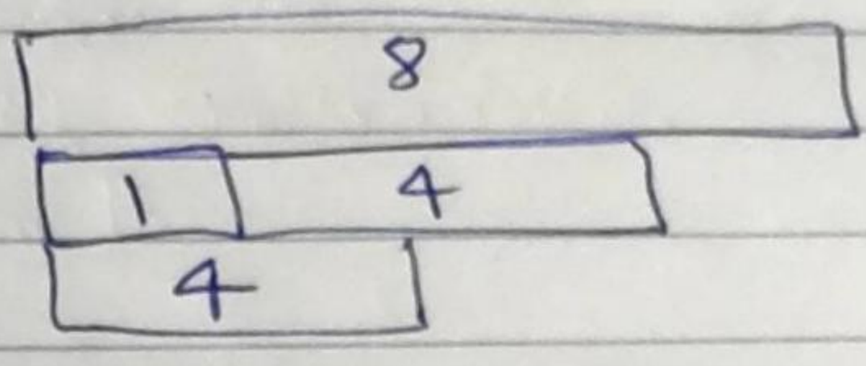
Ans ⟶ 16

[Multiple of highest individual size].

(c) ~~✓~~

double → 8
char → 1
uint → 4
int* → 4
17

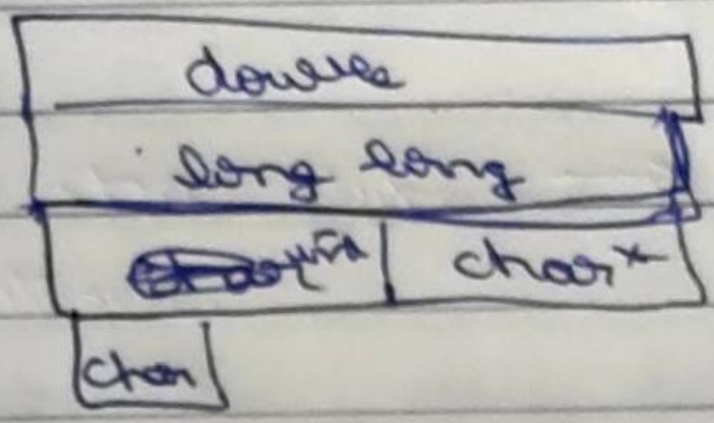
Ans. → 24 bytes.



Q4

double → 8
char → 1
int → 4
long long → 8
~~int~~ char* → 4

(2)



Min. size → 32 bytes.

```
double;   - 8 —
long long; - 8 —
uint;     - 4 —
char*;    - 4 —
char;     - 1 —
```

Q5 (a)

Due to PADDING.

(1)

Q6 (a) int *p = new int ;

Memory is dynamically allocated for pointer p but never deleted. ~~So after the program~~ After the scope of p ends, the memory address is lost. This is a memory leak.

(c) ~~(b)~~ int * p = nullptr ;

This pointer has address 0.

cout << p ; ⟶ Outputs 0.

A pointer at address 0 is a null pointer.

(b) ~~(c)~~ A dynamically allocated pointer, once deleted, becomes a dangling pointer.

int * p = new int ;

delete p ; ⟶ // p is now dangling pointer.

Q7 ~~(1)~~ global scope ⟶ ①, ④

Function scope ⟶ ②

local scope ⟶ ③, ⑤, ⑥, ⑦, ⑧

All variables inside curly braces have local scope.

1 ✓

2 ✓

3 X

4 ✓

5 X

6 ✓

7 X

8 ✓

Q8 'const'. Non-changing values are best defined as global variables. If was inside Jobe

Q9 Multiple values can be returned in a tuple.

```
tuple <int, float, string> func ()
{
    return make_tuple (1, 2.3, "Longthy faher");
}
```

STL

Q10 Initialization → Can be used to define a variable and give it a value in one step.

Assignment → Assigns values to a variable that has already been defined.

We can initialise a particular variable only once, but can assign it values multiple times.

```
int x = 5;    // Init.
  x = 109;    // Ass.

int arr[3] = {4, 2, 0};    // Init
  arr[0] = 92;    // Assign.
```

Q11

```
int main() {
    int x = 3;
    { int x = 5;
      cout << x; }
    cout << x; }
```

Output ⟶ 5 3

The outer x was hidden when another x was defined.

Global variables can be accessed outside the scope of the local variable that caused hiding. For example by using a function outside local scope.

→ Target was to access hidden variable, not at the place where it is not hidden

Q12 (a) and (b) should give stack overflow or more memory is needed than is available.

(c) will not give stack overflow. The NEW keyword lets us dynamically allocate memory from outside the stack.

Q.13 delete is used to remove a dynamically allocated pointer.

delete [] is used to delete the entire array.

Using 'delete' on an array can cause memory leak as it will only delete the pointer to the first value in the array and leave the rest.

Q.16 Default arguments can be passed in function declaration with default values provided.
This default value is used if a value is not given when function is called.

Rule → Default arguments can be set only one, either during declaration or definition.

Q.17 Many standard elements of code such as cout, cin, vector, make_tuple become easier to write if we enter the 'using namespace std;' line.

Question was about why namespace std even exists.

Q18 ~~If integer~~ x is ~~a~~ initialised twice, and the parameter n is also initialised twice.
→ 4 integer copies are made.

~~Q19 Output → 30.~~ Q19 Output → 30

Q20 arr+1 → gives pointer to the element at index 1.

Dereference of & arr is arr+1.
i.e. → (&arr) + 1 also points ~~to~~.

(&arr) + 1 points to the end of the array.
~~*[(&arr)+1]~~
*(ptr-1) will point to the last element

Output → 25.

Q21 Output → 6 ✓
5 ✓

NOT a garbage value.

(2)

(1) In function, a copy of the array is passed.
* arr = arr+1 means the arr pointer now points two [illegible] index [1].

(2) However, outside the function, arr still points to index 0.

Q22

(2.5)

[5] ✓ delete[] 4 X
(5) ✓ delete 4 ✓
{5} ✓ delete 4 ✓

Q23 (1) The code decreases the value of n by a certain amount each step.
Let's say n, in binary, is 10111000.
After first loop, n becomes 10110000
second → 10100000
third → 10000000
fourth → 00000000
Hence, each time it removes the last set bit.

$2^3 - 1 \rightarrow 7$, 111

Max possible value for count is :-

Since max value of unsigned int is $2^{32} - 1$, which is 1111 .... (32 times).

Count can be 32. ✓

If count = α, the operator α is called α times.

Q24 (6)

```
int delta (char* c, int size) {
        bitset <26> b;
        for( int i = 0; i <=size < i ++) {
                b.set ( *(c+i) - 'a'); }
// ~~cout << b; {~~
// Now b has unity representation. We will do
// binary to decimal conversion.
    int ans = 0;
    int m = 1;
    for( int i=0; i<26; i++) {
        if (b[i] == 1) ans += m;
        m = m*2;
    }
    return ans;
}
```

→ ERR, logic + Compiles (−2)

Can be optimised by using OR operator directly to integer