For this quiz, you can assume that the sizes of the data types (in bytes) are as follows: **char** = 1, **short** = 2, **int** = 4, **long** = 8, **float** = 4, **double** = 8. You may also assume that pointers are 8 bytes and all necessary header files have been included in the code snippets.

1. Given the declarations below, if the assignment is legal (meaning there is no compiler warning or error) write **OK**. If the assignment is illegal (meaning there is a compiler warning or error) write an **X**.

```
int i = 5;
int j = 6;
const int ci = 10;
const int cj = 11;
const int *pci;
int *pi;

a) ci = 8;
b) j = ci;
c) pi = &ci;
d) pci = &cj;
e) *pci = 5;
f) pci = &i;
```

2. Given the valid declarations below, what is printed?

```
   int a[] = {15, 13, 2, 7, 9, 1, 8, 3, 6, 4};
   int *p = &a[2];
   int *q = &a[7];

a) printf("%i", p - q);
b) printf("%i", *(q - 4));
c) printf("%i", *p - *q);
d) printf("%i", *(p + 2));
```

3. What is the output from the `printf` statement below? The expression is valid and well defined.

```
int a = 6; int b = 5; int c = 4;
int d = 3; int e = 2; int f = 1;

a -= b += -c-- +d++ + - -e++ / ++f;
printf("%i, %i, %i, %i, %i, %i", a, b, c, d, e, f);
```

4. Given the valid declarations and expressions below, rewrite the expressions using the variable `a` instead of `p`. Hint: Draw a diagram to help you.

```
int a[] = { /* initializers are here */ };
int *p = a + 3;

a) p - 1
b) *p + 2
c) *(p + 1)
d) p[4]
```

5. Given the declarations below, place an **X** next to each assignment that is illegal or dangerous and an **OK** next to each assignment that is correct and safe. In other words, if the compiler will issue an error or a warning about the assignment, put an **X** next to it.

```
int i = 5;
int j = 10;
int *p = &i;
int *q = &j;

a) p = i;
b) *p = &i;
c) *p = *q;
d) p = q;
e) p = &q;
f) p = *q;
```

6. Given the declarations below, give the precise type of each expression **AND** the value of the expression. If the expression is illegal, write **ILLEGAL** in the type column and leave the value blank. Assume that the address of the array **a** is 200 and the address of **p** is 100. Note that p doesn't change as there are no assignments or side-effect operations. (Hint: Draw a diagram to help you with the questions. The first two expressions are examples.)

```
int a[] = {2, 1, 9, 0, 4, 7, 6, 5, 8, 3, 1};
int *p = a + 5;
```

	Expression	Type of the expression	Value
	`p`	**`pointer to int`**	**120**
	`p[2]`	**`int`**	**5**
a)	`p + 3`		
b)	`*p`		
c)	`*p[1]`		
d)	`p[-1]`		
e)	`*p + 5`		
f)	`&p`		
g)	`&p[4]`		
h)	`*(p + 4)`		