Sample Exam Questions

**True/False:** For each statement, circle True if the statement is ALWAYS true or False if the statement is false.

1. A **for** loop will execute at least once **always**. ~~True~~ **False**
*Do-while loops execute at least once.*

2. The following two statements are identical: ~~True~~ **False**
```
int ABC, DEF;
int abc, def;
```
*C++ is case sensitive.*

3. **While** and **Do-While** loops are exactly the same. ~~True~~ **False**
*Do-while loops execute at least once, while loops need not*

4. Given the declaration: int X = 123; ~~True~~ **False**
The statement cout >> x; will display the number 123 on the screen.
*The correct form is cout << x; information flows from the variable.*

5. In a **for** loop, the starting counter value can be larger than the ending counter value. **True** ~~False~~
*For(int counter = 10; counter >=0; counter--)*

6. The expressions i *= a+b and i = i*a+b produce identical values. ~~True~~ **False**
*The first is i=i*(a+b) the second is i=(i*a)+b.*

7. The expression: for(;;) is illegal and will cause a compiler error. ~~True~~ **False**
*This is perfectly valid and creates a (truly) infinite loop.*

8. The empty statement **;**, a semicolon only, is allowed in C++. **True** ~~False~~
*Yes, and this feature can cause some nasty bugs in your code!*

9. The expressions cout << "\n"; and cout << endl; produce the same results. **True** ~~False~~
*Both move the output to a new line.*

10. The expression (rand()%6) generates a random number and could be any value between 1 and 6. ~~True~~ **False**
*It produces a number between 0 and 5.*

11. Arrays are always passed by reference into functions. **True** ~~False~~
*Always, because the array is actually a pointer.*

12. The argument type double* or double[] can be used in function definitions to transfer an array address to a function. **True** ~~False~~
*Because arrays are pointers.*

13. The statement int* p1, p2; creates two pointers to integers p1 and p2. ~~True~~ **False**
*To get two pointers you need int *p1, *p2;*

14. For every **new** statement you should have a **delete** statement to prevent memory leaks. **True** ~~False~~
*Yes. It is good practice and a good habit to form.*

15. char a[] = {'w', 'o', 'r', 'l', 'd'}; will actually create an array with six elements. ~~True~~ **False**
*Just 5 elements here, but char a[] = "world" would be 6. Why?*

16. If ptr is a pointer then ptr++ is allowed, but ptr*=2 is not allowed. **True** ~~False~~
*Pointers are not integers, you can move a pointer with ++, but not *=.*

17. Because a double is twice as large as a float, a double* or pointer to a double is also twice as large as a float* or pointer to a float. ~~True~~ **False**
*No, both are addresses, same size.*

18. If p1 and p2 are pointers of the same type then *p1 = *p2; is the same as p1 = p2;. ~~True~~ **False**
*The first shares the value the second shares the address. Very different!*

**Short Answer:** Identify the output for each set of expressions and write it in the space provided. Assume the expressions are well formed and all necessary declarations and includes are in place. (Hint: all code segments compile and run)

1.
```
int x=1;

while(x<5)
{
        x++;
        cout<<"x= "<<x<<endl;
}

cout<<"The final value of x is: "<<x<<endl;
```

Answer:

*x=2*
*x=3*
*x=4*
*x=5*
*The final value of x is: 5*

2.
```
int i=1, j=7;
while(i < j)
{
        cout << "i= " << i++ << " j= " <<--j  << endl;
}
```

Answer:

*i=1 j=6*
*i=2 j=5*
*i=3 j=4*

3. int x=1, sum = 0;

```
for(x = 0; x < 7; x+=2)
{
        sum+= x;
        cout<<"sum = " << sum << endl;
}

cout<<"The final value of x is: "<<x<<endl;
```

Answer:

*sum = 0*
*sum = 2*
*sum = 6*
*sum = 12*
*The final value of x is: 8*

4.
```
int x=0, y=0;
for(x=1; x< y; x++)
{
        cout<<"x= "<<x;
}

cout << endl <<"The final value of x is: "<<x<<endl;
```

Answer:

*The final value of x is: 1*

5. int sum=0;

```
    while(sum <= 7)
    {
            sum += 2;

            if(sum%2 == 0)
                    Even(sum--);
            else
                    Odd(++sum);
    }


void Odd(int number)
{
        cout << "O" << number << endl;
}

void Even(int number)
{
        cout << "E" << number << endl;
}
```

Answer:

*E2*
*O4*
*E6*
*O8*

6.
```
int count = 5;

for(int row = 0; row < count; row++)
{
    for(int col = 0; col < count; col++)
    {
        if(row > col)
            cout << "+";
        else if(row < col)
            cout << "-";
        else
            cout << "0";
    }
    cout << endl;
}
```

Answer:

```
0----
+0---
++0--
+++0-
++++0
```

7.
```
int arr[6] = {1, 2, 6};

for(int index = 5; index >= 0; index-=2)
{
        cout<< arr[index]<<endl;
}
```

Answer:

*0*
*0*
*2*

8.
```
int arr[5] = {1, 2, 3, 4, 5};
int* pb = arr;
int* pe = &arr[4];
while(pb != pe)
{
        cout << *pe << endl;
        pe--;
}
```

Answer:

*5*
*4*
*3*
*2*

9.
```
int arr[] = {1, 2, 3, 4, 5, 6};
int* b = &arr[1];
for(int i = 1; i <= 4; i++)
{
        cout << *b << " -> ";
        (b++);
        cout << *b << endl;
}
```

Answer:

*2 -> 3*
*3 -> 4*
*4 -> 5*
*5 -> 6*

10. The following code compiles correctly, but contains a logical error. Write the **output**, **describe the logical error** and **describe how you would correct the error**.

```
int* a = new int;
int* b =a;


*a = 5;
cout << "a=" << *a << endl;
*b = 10;
cout << "a= " << *a << endl;
delete a;
cout <<"b=" << *b << endl;
```

Answer:

*a=5*
*a= 10*
*b= some garbage number*

*The problem is that a and b point to the same memory location. When you delete a, that memory is free. This is called a* ***Dangling Pointer****.*

*To fix this, we need to move the delete down after we use b, or not use b after the delete. We should also set a and b to the nullptr.*

11.
```
int arr[] = {1, 2, 3, -1,  0, -1};

int* b = &arr[0];


for(int index = 0; index < 6; index++)
{
        cout << *b <<  " -> ";
        b += *b;
        cout << *b << endl;
}
```

Answer:

*1 -> 2*
*2 -> -1*
*-1 -> 3*
*3 -> -1*
*-1 -> 0*
*0 -> 0*

*This one is a bit tricky, think about what it is doing and come with questions.*

**Programming**

1. The code snippet below will randomly generate three numbers, num1, num2, and num3. Fill in the missing code below so that the largest of the three numbers can be displayed, i.e. so that the variable **largest** contains the largest value among the three numbers.

```
int num1 = 0, num2 = 0, num3 = 0, largest = 0;
num1 = rand();
num2 = rand();
num3 = rand();
cout << "The numbers are : " << num1 << ", " << num2 << " , and " << num3 << ".\n";
// Add your code here
```

```
cout << "The largest number is : << largest << endl;
```

2. The code snippet below uses a for loop to print out some numbers. Rewrite the code to use a **while loop** to display the same numbers. Hint: you may need to use a different initialization value for the counter variable.

```
int count = 0;
for(count = 11; count > 0; count-= 3)
{
        cout << count << endl;
}


// Add your code here
```

3. You have been asked to write some software for **We're-Not-Iffy Burger** to assist in preparing the main menu item, tasty cheese burgers. The program will ask the patron for the number of burger patties and display the corresponding message as shown below. You elect to use a function to output the message and prepare to code a solution. Just as you accept, the owner tells you that using **if** statements is against company policy thus , **no if-statements allowed in your solution**

| If the patron requests…. | You output |
|---|---|
| 0 | Enjoy your grilled cheese… |
| 1 | Enjoy the regular! |
| 2 | Enjoy the Double! |
| 3 | Enjoy the TRIPPLE! |
| Anything else, 4,5,6… | Sorry, we don't serve that here. |

```
#include <iostream>
#include <cstdlib>

using namespace std;

void OutputMessage(int);

int main()
{
        int numPat;

        cout << "Please enter the desired number of tasty burger patties :";
        cin >> numPat;

// Add your code here









        system("pause");
        return 0;
}
```

```
void OutputMessage(int)
{// Add your code here









}
```

4. Write a function, **randFill**, that fills a 1-dimensional array, **arr**, with **n** random integers. The array **arr**, is of length **n**. Each randomly generated integer, x, should be in the range 0 <= x < **n**. Your function takes in the array to fill, and the number of random integers, **n** to generate. It does not return any value. Remember that the built in function **rand()** will generate a random integer between 0 and RAND_MAX, a very large number.

```
void randFill(int arr[], int n)

{// Add your code here




}
```

5. Write a function, **getMode**, to find the mode--the most frequently occurring element--of a given 1-dimensional array, **arr**, containing n integers. The function **getMode** should return the mode, or most common element from the array. You can assume the values in the array are in the range from 0 up to, but not including **n**, just like the previous question.

```
int getMode(const int arr[], int n)

{// Add your code here




}
```

6. You have been asked to assist in the scoring of Olympic sports. To do so, you will write a function that computes the "Olympic" average of a given 1-dimensional array, **arr**, containing n integers. The "Olympic" average of a set of numbers is simply the average where the highest value and the lowest value are not considered. Because some sports are subjectively scored, removing the high and low score is thought to lower the bias in the scores. The average should be calculated over n-2 elements excluding the highest and lowest scores. Even though the array **arr** contains only integers, getOA should return a return a floating point value. (25)
Use functions to find the largest and smallest values.

```
#include <iostream>
#include <cstdlib>

using namespace std;

const int NUMSCORES = 7;

int Largest(int arr[], int n);
int Smallest(int arr[], int n);
float getOA(int arr[], int n);

void main()
{
        int scores[NUMSCORES]; = {10, 9 ,7, 9, 10, 8, 7}
        float average = 0.0;

        // Add your code here









        cout << "The Olympic average score is : " << average << endl;

        system("pause");

}
```

```
int Largest(int arr[], int n)
{// Add your code here

}

int Smallest(int arr[], int n)
{// Add your code here

}
```

```
float getOA(int arr[], int n)
{// Add your code here









}
```