For the next three questions, consider the following short program:

```cpp
#include <iostream>
#include <new>
using namespace std;

int main() {
    char* pC = NULL;   // Line 1
    pC = new char;     //      2
    *pC = 'x';         //      3

    return 0;
}
```

1. Which of the following best describes the situation after Line 1 has been executed?
   1) `pC` does not have a target, and `pC` does not have a known value.
   2) `pC` does not have a target, but `pC` does have a known value.
   3) `pC` has a target, but its target does not have a known value.
   4) `pC` has a target, and its target does have a known value.
   5) None of these

2. Which of the following best describes the situation after Line 2 has been executed?
   1) `pC` does not have a target, and `pC` does not have a known value.
   2) `pC` does not have a target, but `pC` does have a known value.
   3) `pC` has a target, but its target does not have a known value.
   4) `pC` has a target, and its target does have a known value.
   5) None of these

3. Which of the following best describes the situation after Line 3 has been executed?
   1) `pC` does not have a target, and `pC` does not have a known value.
   2) `pC` does not have a target, but `pC` does have a known value.
   3) `pC` has a target, but its target does not have a known value.
   4) `pC` has a target, and its target does have a known value.
   5) None of these

4. Assume the variable declarations:

   ```cpp
   int Foo = 0;
   int *ptr = &Foo;
   ```

Which of the following statements will change the value of `Foo` to 1?

   1) `ptr++;
   2) `Foo++;
   3) `(*Foo)++;
   4) `(*ptr)++;
   5) All of these
   6) 1 and 2 only
   7) 1 and 4 only
   8) 2 and 4 only
   9) 3 and 4 only
   10) None of these
For questions 5 through 9, consider the following implementation of a function to find the maximum value in an array of integers. The for loop is required to be implemented using pointers to access elements rather than direct array indexing.

```cpp
int maxEntry(const int* const Data, int Sz) { // Line 1
    if ( Data == NULL || Sz <= 0 ) return INT_MIN; // 2
    int Count = 0; // 3
    // Set hiSoFar to point to the first array element:
    const int *hiSoFar = ________; // 4
    // Set Current to point to the second array element:
    const int *Current = ________; // 5
    for ( ; Count < Sz; __________ ) { // 6
        if ( __________ ) // 7
            hiSoFar = Current; // 8
    }
    return ( ____________ ); // 9
}
```

5. How should the blank in Line 4 be filled?
   1) Data
   2) *Data
   3) &Data
   4) &Data[0]
   5) Data[0]
   6) 3 or 5 only
   7) 3 or 4 only
   8) None of these

6. How should the blank in Line 5 be filled?
   1) hiSoFar
   2) hiSoFar++
   3) Data++
   4) &Data[1]
   5) Data[1]
   6) 2 or 4 only
   7) 2 or 5 only
   8) None of these

7. How should the blank in Line 6 be filled?
   1) Count++
   2) Current++
   3) Count++, Current++
   4) It should be left blank.
   5) None of these

8. How should the blank in Line 7 be filled?
   1) Current > hiSoFar
   2) &Current > &hiSoFar
   3) *Current > *hiSoFar
   4) *Current < *hiSoFar
   5) None of these
9. How should the blank in Line 9 be filled?

1) *hiSoFar
2) &hiSoFar
3) hiSoFar
4) It should be left blank.
5) None of these

For questions 10 through 12, assume that \(P\) and \(Q\) are pointers of the same type, and that each has been assigned a value.

10. What comparison would determine whether \(P\) and \(Q\) have targets with the same value?

1) \&P == \&Q
2) \*P == \*Q
3) \(P == Q\)
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

11. What comparison would determine whether \(P\) and \(Q\) have the same target?

1) \&P == \&Q
2) \*P == \*Q
3) \(P == Q\)
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

12. What comparison would determine whether \(P\) and \(Q\) store the same value?

1) \&P == \&Q
2) \*P == \*Q
3) \(P == Q\)
4) All of them
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

13. Consider implementing a function to dynamically allocate an array of integers and set all its elements to zero:

```cpp
void ZeroIt(int* A, const int Size) {
    A = new int[Size];
    for (int Idx = 0; Idx < Size; Idx++) {
        A[Idx] = 0;
    }
}
```

Which of the following choices for the blank preceding the formal parameter \(A\) is best?

1) int*
2) int*&
3) const int*
4) int* const
5) const int* const
6) All of the above
For questions 14 through 17 assume that we have a dynamically allocated array \( A \) of integers of dimension \( \text{Size} \), with memory layout as shown:

```cpp
const int Size = 5;
int *A = new int[Size];
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>007D0E70</td>
</tr>
<tr>
<td>1</td>
<td>007D0E74</td>
</tr>
<tr>
<td>2</td>
<td>007D0E78</td>
</tr>
<tr>
<td>3</td>
<td>007D0E7C</td>
</tr>
<tr>
<td>4</td>
<td>007D0E80</td>
</tr>
</tbody>
</table>

14. Which code fragment(s) could be inserted in the blank in order to safely initialize each element of \( A \) to zero?

```cpp
int* p = &A[0];
for (int Idx = 0; Idx < Size; Idx++, p++) {
    *p = 0;
}
```

1) *A = 0;
2) A[Idx] = 0;
3) *p = 0;
4) *Idx = 0;
5) All of the above
6) 1 and 2 only
7) 1 and 3 only
8) 2 and 3 only
9) 1 and 4 only
10) None of these

15. What value will be printed by the code fragment:

```cpp
for (int Idx = 0; Idx < Size; Idx++) {
    A[Idx] = int(&A[Idx]); // typecast converts address to int
}
```

1) 007D0E70
2) 007D0E74
3) 007D0E78
4) 007D0E7C
5) 007D0E80
6) Unknown
7) None of these

16. Assuming only the initial declarations given above, what logical error(s) would result if the following statement were executed: \( A = \text{new int}[2 \times \text{Size}] \)?

1) A dangling pointer would result (a pointer whose value is the address of memory that the program no longer owns).
2) A memory leak would result (the program would own memory that it could no longer access).
3) Both a dangling pointer and a memory leak would result.
4) Neither a dangling pointer nor a memory leak, but some other logical error would result.
5) No logical error would result.

17. Assuming only the initial declarations given above, and execution of the correctly completed code given in question 15, what logical error(s) would result if the following statement were executed: `delete [] p;`

1) A dangling pointer would result (a pointer whose value is the address of memory that the program no longer owns).
2) A memory leak would result (the program would own memory that it could no longer access).
3) Both a dangling pointer and a memory leak would result.
4) Neither a dangling pointer nor a memory leak, but some other logical error would result.
5) No logical error would result.
For questions 18 through 24, consider the following main function, which deals with a dynamically allocated array of pointers to string objects:

```cpp
int main() {
    ifstream In("Text.data");

    int numNames;
    In >> numNames;
    In.ignore(INT_MAX, 'n');

    // Allocate an array of pointers to string objects:
    string** Name = new string*[numNames];
    // Set each array cell to NULL:
    initArray(Name, numNames);
    // Read strings from input file:
    acquireData(Name, numNames, In);
    // Display strings to verify input success:
    writeArray(Name, numNames);

    // Search for strings matching "bar":
    string Sought = "bar";
    int Matches = numMatches(Name, numNames, Sought);
    cout << "Number of matches for " << Sought
    << " is " << Matches << endl;

    In.close();
    return 0;
}
```

For the next two questions, consider the implementation of the input function:

```cpp
bool acquireData(string** const A, int Sz, ifstream& In) {
    if ( A == NULL || Sz <= 0 ) return false;
    string Temp;
    for (int Pos = 0; In && Pos < Sz; Pos++) {
        getline(In, Temp);
        if ( A[Pos] != NULL ) delete A[Pos];
        A[Pos] = new string(Temp);
    }
    return ( Pos == Sz );
}
```


1) To prevent the program from writing data to an invalid address.
2) To prevent the program from reading data from an invalid address.
3) To prevent a memory leak; i.e., losing access to memory without deallocating it.
4) None of these
19. What is the purpose of the comparison used in Line 8?

1) To determine whether the expected number of data values was read.
2) To prevent the program from reading more than the specified number of data values.
3) To prevent the program from reading fewer than the specified number of data values.
4) None of these

For the next two questions, consider the implementation of the function to write the strings:

```cpp
void writeArray(string** const A, int Sz) { // Line 1
    if ( A == NULL || Sz <= 0 ) return; // 2
    for (int Pos = 0; Pos < Sz; Pos++) { // 3
        cout << setw(5) << Pos << ": "; // 4
        else cout << "null pointer" << endl; // 6
    }
}
```

20. Consider the statement in Line 2. Assuming that \( Sz \) is expected to be the number of values stored in the array \( A \), what is the purpose of the statement in Line 2?

1) To prevent an access violation in Line 5 if the array hasn't been allocated.
2) To prevent an access violation in Line 5 if the array contains pointers that don't have targets.
3) 1 and 2
4) None of these

21. What is the purpose of the test in Line 5?

1) To prevent an access violation in Line 6 if the array hasn't been allocated.
2) To prevent an access violation in Line 6 if the array contains pointers that don't have targets.
3) 1 and 2
4) None of these

For the next three questions, consider the function to search the array and count string matches:

```cpp
int numMatches(string** const A, int Sz, const string& toMatch) { // Line 1
    if ( A == NULL || Sz <= 0 ) return 0; // 2
    int Count = 0; // 3
    for (int Pos = 0; Pos < Sz; Pos++) { // 4
        // 5
        Count++; // 6
    }
    return Count; // 7
}
```
22. In order to prevent an access violation, how should the first blank in Line 5 be filled?

1) A != NULL
2) A[Pos] != NULL
3) toMatch != ""
4) None is needed; it should be left blank and the && should be removed.
5) None of these

23. In order to correctly detect a match, how should the second blank in Line 5 be filled?

1) toMatch == *A[Pos]
2) toMatch == A[Pos]
3) &toMatch == *A[Pos]
4) 1 or 2 only
5) 2 or 3 only
6) 1 or 3 only
7) None of these

24. Does the order of the two parts of the and-expression in Line 5 matter?

1) No.
2) Yes, if the order is reversed then valid matches may not be detected.
3) Yes, if the order is reversed then matches may be reported when none occurred.
4) Yes, if the order is reversed then access violations could occur if the array did not contain the expected number of pointers that do have targets.
5) None of these

25. Assuming all the allocations are granted, what error(s) occur in the following code fragment?

```cpp
string *p1 = new string("one");    // Line 1
string *p2 = new string("two");    // 2
string *p2 = new string("three");  // 3
string *p2 = p1;                   // 4
delete p2;                         // 5
```

1) A dangling pointer is created in line 3.
2) A dangling pointer is created in line 5.
3) A memory leak occurs in line 2.
4) A memory leak occurs in line 5.
5) All of them
6) 1 and 3 only
7) 1 and 4 only
8) 2 and 3 only
9) 2 and 4 only
10) None of these