Instructions: This homework assignment focuses on enumerated types, arrays and struct variables.

For questions 1 through 4 assume the following declarations:

```
enum WoodKind {ASH, CEDAR, OAK, PINE, POPLAR, WALNUT};

struct Size {
  int Length;
  int Width;
  int Thickness;
};

struct Board {
  Size     Dimensions;
  WoodKind Kind;
  int      smoothSurfaces;
};

Board oneBoard = {{2, 4, 8}, PINE, 4};  // Legal initialization.
const int MAXBOARDS = 1000;
Board Lumber[MAXBOARDS];
```

1) The number of smooth surfaces of `oneBoard` could be printed by the statement(s):

1) `cout << Board.smoothSurfaces;`
2) `cout << oneBoard.smoothSurfaces;`
3) `cout << smoothSurfaces;`
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

2) The width of `oneBoard` could be printed by the statement(s):

1) `cout << oneBoard.Size.Width;`
2) `cout << oneBoard.Dimensions.Width;`
3) `cout << oneBoard.Width;`
4) All of the above
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

Consider implementing a loop to count the number of cedar boards that occur in the array `Lumber`, assuming the array has been initialized to hold data about `numBoards` boards:

```
int numCedar = 0;
for (int Idx = 0; Idx < numBoards; Idx++) {
   if ( _________________ == _________ )
       numCedar++;
}
```

3) How should the first blank in the `if` condition be filled?

1) `Lumber[Idx].WoodKind`
2) `Lumber[Idx].Kind`
3) `WoodKind`
4) `Kind`
5) None of these

4) How should the second blank in the `if` condition be filled?

1) `CEDAR`
2) `" CEDAR " && "cedar"`
3) `"CEDAR"`
4) Either 1 or 2
5) None of these

For questions 5 through 7, assume the type definitions:

```
enum Month {
    JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY,
    AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER
};

struct Date {
  Month month;
  int day;
  int year;
};
```

5) Which of the following is/are valid for creating and initializing a `Date` variable?

1)
```
Date gradDay;
gradDay.month = MAY;
gradDay.day   = 12;
gradDay.year  = 2001;
```

2)
```
Date gradDay  = {MAY, 12, 2001};
```

3)
```
Date.month    = MAY;
Date.day      = 12;
Date.year     = 2001;
```

4) All of these

5) 1 and 2 only

6) 1 and 3 only

7) 2 and 3 only

8) None of these

6) Given the variable declarations: `Date myBirthday, yourBirthday;`
and the function prototype: `void printDate(Date date);`

and assuming the two variables have been properly initialized, which of the following is/are valid expressions or statements involving `Date` variables?

1) `yourBirthday = myBirthday;`
2) `yourBirthday == myBirthday;`
3) `printDate(myBirthday);`
4) All of these

5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
7) None of these

7) Given the variable declarations: `Date gradDay, leaveDay;`

and assuming the two variables have been properly initialized, which of the following is/are valid expressions or statements involving `gradDay` and `leaveDay`?

1) `gradDay++;`
2) `gradDay[0] = MAY;`
3) `gradDay.year <= leaveDay.year`
4) All of these

5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

For questions 8 through 11, consider the following code:

```
const int MAX_EMPL = 1000;
struct Employee {
  string name;
  int    age;
  double hwage;
};

Employee Emp5, Emp7;
Employee Personnel[MAX_EMPL];
Emp5.name = "Joe Bob Hokie";  Emp5.age = 28; Emp5.hwage = 19.87;
Emp7.name = "Haskell Hoo IV"; Emp7.age = 33; Emp7.hwage = 9.32;
```

Assume that the array elements in `Employee` have been initialized so that all fields in each element have valid values.

8) The name fields of the variables `Emp5` and `Emp7` could be compared by the expression(s):

1) `Emp5.name = Emp7.name`
2) `Emp5.name == Emp7.name`
3) `Emp7[name] == Emp7[name]`
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

9) The variables `Emp5` and `Emp7` could be compared by the expression(s):

1) `Emp5 <= Emp7`
2) `Emp7 == Emp7`
3) `Emp5 = Emp7`
4) All of the above
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

10) The statement: `cout << Personnel[17].name;`

1) prints the name of an employee that is 17 years old
2) prints the name of the employee whose `Employee` record is stored at index 17
3) is syntactically illegal since you can't print a `struct` variable into an output stream
4) None of these

11) The statement: `Personnel[17] = Emp5;`

1) copies the contents of the `Employee` record at index 17 into the variable `Emp5`
2) copies the contents of the variable `Emp5` into the `Employee` record at index 17
3) swaps the contents of the variable `Emp5` and the `Employee` record at index 17
4) is syntactically illegal
5) None of these

12) Suppose that a `struct` parameter is to be passed to a function, and that the function does not need to modify either the formal parameter or the actual parameter. What advantage(s) could be gained by passing the parameter by constant reference instead of passing it by value in this situation?

1) This would probably require less execution time.
2) This would require less space in memory.
3) The syntax of the function call would be simpler.
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

---

13) Passing a `struct` parameter by constant reference instead of by reference:

1) reduces the amount of memory needed
2) increases the amount of memory needed
3) reduces the time required for the function call
4) increases the time required for the function call
5) 1 and 3 only
6) 2 and 4 only
7) None of these

For questions 14 through 16 assume the following enumerated type declaration:

```
enum Status {NEGATIVE, OK, UNUSED, TOOBIG};
```

The function `checkIndex()` is intended to validate an array index, as you did in Project 8, and indicate to the caller what is determined:

```
_______ checkIndex(const int Idx,           // index to be checked
                   const int Used,          // # of used cells in array
                   const int Dimension) {   // dimension of array

   if (Idx < 0)
      return __________;     // line 1
   if (Idx < Used)
      return OK;
   if (Idx < Dimension)
      return UNUSED;
   return ________;          // line 2
}
```

14) What should the return type of the function be?

1) `UNUSED`
2) `void`
3) `int`
4) `Status`
5) None of these

15) How should the blank in line 1 be filled?

1) `NEGATIVE`
2) `Status`
3) `-1`
4) `0`
5) None of these

16) How should the blank in line 2 be filled?

1) `TOOBIG`
2) `OK`
3) `Status`
4) It should be left blank.
5) None of these

For questions 17 through 20, consider the declarations and function:

```
struct Bar {
  int  Foo[5];
  int  Size;
};

Bar x = {{2, 5, 9, 11, 17}, 5},  // This is a legal initialization.
    y = {{0, 3, 6}, 3};

Bar AddBar(const Bar& x, const Bar& y) {
   Bar NewBar;
   if (x.Size <= y.Size)
      NewBar.Size = x.Size;
   else
      NewBar.Size = y.Size;

   for (int Idx = 0; Idx < NewBar.Size; Idx++)
      NewBar.Foo[Idx] = x.Foo[Idx] + y.Foo[Idx];

   return NewBar;
}
```

17) The statement: `y = x;`

1) would copy the contents of the `Bar` variable `x` into the `Bar` variable `y`
2) is syntactically illegal because you can't assign an array to an array
3) is syntactically legal, but has a different effect than 1)
4) None of these

18) The statement: `y.Foo = x.Foo;`

1) would copy the contents of the `Foo` field of `x` into the `Foo` field of `y`
2) is syntactically illegal because you can't assign an array to an array
3) is syntactically legal, but has a different effect than 1)
4) None of these

19) Suppose the following statement is executed: `Bar z = AddBar(x, y);`

Then the value of `z.Foo[2]` would be:

1) `0`
2) `2`
3) `8`
4) `9`
5) `15`
6) None of these

20) Again, suppose the following statement is executed: `Bar z = AddBar(x, y);`

Then the value of `z.Size` would be:

1) `1`
2) `2`
3) `3`
4) `4`
5) `5`
6) `0`
7) None of these