CS 1044 Homework Assignment 7 **Spring 1999**

Instructions: This homework assignment focuses on enumerated types, arrays and struct variables. The answers to the following questions can be determined from Chapters 3 through 11 of the lecture notes and Chapters 11 through 14 of the text. Assume any `#include` directives, variable declarations, etc, which are needed to make the given code syntactically correct.

Print your name and code your ID number correctly on the opscan form.

Mark Group 1 if you are in the 8TTh section and Group 2 if you are in the 10MWF section.

Turn in your completed opscan at class on April 29 or 30 or at the McB 116 lab between 1:00 and 3:00 pm on April 30. Opscans will not be accepted late or at any other time.

For questions 1 through 3, consider the array declaration

```
const int Size = 10;
int Poirot[Size] = {42, 37, 29, 43, 42, 17, 75};
```

and the following function definition:

```
int Mystery( const int List[], int listLength, int Alpha) {

    int Idx = 0;

    while (Idx < listLength && Alpha != List[Idx])
        Idx++;

    return Idx;
}
```

1) What value would the following statement assign to the variable `Clue`?

```
int Clue = Mystery(Poirot, Size, 29);

1)  0        2)  1       3)  2       4)  3       5)  4
6)  10       7)  11      8)  a runtime error would occur
9)  none of these
```

2) What value would the following statement assign to the variable `Clue`?

```
int Clue = Mystery(Poirot, Size, 13);

1)  0        2)  1       3)  2       4)  3       5)  4
6)  10       7)  11      8)  a runtime error would occur
9)  none of these
```

3) What value would the following statement assign to the variable `Clue`?

```
int Clue = Mystery(Poirot, Size, 42);

1)  0        2)  1       3)  2       4)  3       5)  4
6)  10       7)  11      8)  a runtime error would occur
9)  none of these
```

For questions 4 and 5, consider the array declaration

```
const int Size = 10;
int myList[Size] = {17, 29, 37, 42, 43, 54, 75};
```

and the following binary search function definition:

```
int BinSearch(const int aList[], int xElem, int& Lo, int& Hi )  {

   int Mid;

   while ( Lo <= Hi ) {

       Mid = ( Lo + Hi ) / 2;          // locate middle

       if ( aList[Mid] == xElem )      // check for target
          return Mid;
       else if ( xElem < aList[Mid] )
          Hi = Mid - 1;                // look in lower half
       else
          Lo = Mid + 1;                // look in upper half
   }

  return -1;
}
```

4) What value would the variable `Low` have after the function call:

```
int Low = 0, High = 6;
BinSearch(myList, 43, Low, High);

1)  0        2)  1       3)  2       4)  3       5)  4
6)  5        7)  6       8)  a runtime error would occur
9)  none of these
```

5) What value would the variable `High` have after the function call:

```
int Low = 0, High = 6;
BinSearch(myList, 30, Low, High);

1)  0        2)  1       3)  2       4)  3       5)  4
6)  5        7)  6       8)  a runtime error would occur
9)  none of these
```

---

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

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

For questions 7 through 9, assume the type definition:

```
typedef struct {
  int      Age;
  float    Height;
  int      Weight;
} PersonRec;
```

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

1)
```
PersonRec me;
me.Age    = 19;
me.Height = 66.5;
me.Weight = 140;
```

2)
```
PersonRec me = {19, 66.5, 140};
```

3)
```
PersonRec.Age    = 19;
PersonRec.Height = 66.5;
PersonRec.Weight = 140;
```

4) all of the these

5) 1 and 2 only

6) none of these

8) Given the variable declarations: `PersonRec You, Me;`

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

1) `You = Me;`
2) `You == Me`
3) `cout << You;`
4) all of the above
5) 1 and 2 only
6) 1 and 3 only
7) none of these

9) Given the variable declarations: `PersonRec You, Me;`

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

1) `You.Age++;`
2) `Me[2] = 4;`
3) `Me.Age >= You.Age`
4) all of the above
5) 1 and 2 only
6) 1 and 3 only
7) none of these

---

10) 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 11 through 15, consider the declarations:

```
const int NameLength   =   50;
const int MaxEmployees = 1000;
typedef struct {
  char   Name[NameLength + 1];
  int    IDNum;
  double HourlyPayRate;
} IDType;

IDType Emp5 = {"Joe Bob Hokie", 4242, 19.87},
       Emp7 = {"Haskell Hoo IV", 1024, 9.32};
IDType Personnel[MaxEmployees];
```

Assume that the array `Personnel[]` has been initialized so that all fields of all the array elements have logically correct values.

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

1) `Emp5.Name == Emp7.Name`
2) `Emp7.Name =  Emp7.Name`
3) `strcmp(Emp5.Name, Emp7.Name)`
4) All of the above
5) 1 and 2 only
6) 2 and 3 only
7) 1 and 3 only
8) None of these

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

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

13) The statement: `cout << Personnel[17].Name;`

1) prints the name of the employee with ID number 17
2) prints the name of the employee whose `IDType` record is stored at index 17
3) is syntactically illegal since you can't dump a `struct` variable into an output stream
4) None of these

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

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

15) Assuming that an int variable occupies 4 bytes, a char variable occupies 1 byte, and a double variable occupies 8 bytes, how many bytes of memory are needed to store the array `Personnel[]` declared above?

1) 62
2) 63
3) 6200
4) 62000
5) 63000
6) none of these

For questions 16 through 18, consider the declarations and function:

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

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;
}
```

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

1) copies 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 logically illegal because `y.Foo` isn't big enough to hold `x.Foo`
4) is syntactically legal, but has a different effect than 1)
5) None of these

17) 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

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

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

1) 0 2) 1 3) 2 4) 3
5) 4 6) 5 7) none of these

For questions 19 and 20 assume the following declarations:

```
enum WoodKind {HARDWOOD, SOFTWOOD};

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

typedef struct {
  Size     Dimensions;
  WoodKind Kind;
  int      smoothSurfaces;
} Wood;

Wood oneBoard = {{2, 4, 8}, HARDWOOD, 4};
```

19) The `smoothSurfaces` field of the variable `oneBoard` could be printed by the statement(s):

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

20) The `Width` field of the variable `oneBoard` could be printed by the statement(s):

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