Midterm Exam #1 – April 24, 2018

COEN 79: Object-Oriented Programming and Advanced Data Structures

Instructor: Dr. Behnam Dezfouli

Time: 80 minutes | Points: 20 + 1 (extra credit)

- **NAME:**

---

1. The `point` class is defined as follows:

[< 5 min] [2pt]

```
class point {
public:
    point(double initial_x = 0.0, double initial_y = 0.0);
    void shift(double x_amount, double y_amount);
    void rotate90();
    double get_x() const {  return x;  }
    double get_y() const {  return y;  }
    friend istream& operator>> (istream & ins, point & target);

private:
    double x; // x coordinate of this point
    double y; // y coordinate of this point
};
```

We want to:

- Check if two `point` objects are equal
- Assign a `point` object to another `point` object

Show how the header file and implementation file are modified (where/if necessary) to support these two functions?

2. What is *procedural abstraction*?
[< 2 min] [1pt]

3. What are the two methods for finding *test data that is most likely to cause errors*?
[< 5 min] [1pt]

4. Explain why passing an object to a function as a *value parameter* results in higher overhead compared to using a *reference parameter*. Explain your answer by showing the memory structure.
[< 5 min] [1pt]

5. What is the time complexity of this function? *Show the mathematical proof of your answer.*
[< 2 min] [1pt]

```
int function(int n) {
    int count = 0;
    for (int i = n; i > 0; i /= 2)
        for (int j = 0; j < i; j++)
            count ++;
    return count;
}
```

6. What is the output of this code? Explain your answer.
[< 2 min] [1pt]

```
class box {
public:
    box() {  size = 4;  }
    friend size_t getSize(box& input);
private:
    size_t size;
};

size_t getSize(box& input) {
    return this->size;
};

int main(int argc, const char* argv[]) {
    box obj;
    std::cout << "The size is: " << getSize(obj);
    return 0;
}
```

7. Does the following code run? If your answer is “no”, then please fix the code without modifying the `main` function.
[< 5 min] [1pt]

```
#include < iostream >
class point {
public:
    // CONSTRUCTOR
    point(double initial_x = 0.0, double initial_y = 0.0) {
            x = initial_x;
            y = initial_y;
        };

    // MODIFICATION MEMBER FUNCTIONS
    void set_x(double value) {  x = value;  };
    void set_y(double value) {  y = value;  };

    // CONST MEMBER FUNCTIONS
    point operator+ (double& in ) const {
        point tmp;
        tmp.set_x(x + in );
        tmp.set_y(y + in );
        return tmp;
    };

private:
    double x; // x coordinate of this point
    double y; // y coordinate of this point
};





int main(int argc, const char * argv[]) {
    point myPoint1, myPoint2, myPoint3;
    double shift = 8.5;
    myPoint1 = shift + myPoint2;
    myPoint3 = myPoint1.operator + (shift);
    myPoint1 = myPoint1 + shift;
}
```

8. In the following code, complete `operator >>`.
[< 5 min] [1pt]

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

class box {
    double width;
public:
    friend void printWidth(box input);
    void setWidth(double input_width) {  width = input_width;  };
};

istream& operator >> (istream& ins, box& target)
// Postcondition: The width of target has been read from ins.
// The return value is the istream ins.
// Library facilities used: iostream
    {
    }

void printWidth(box v) {
    cout << "Width of box: " << input.width << endl;
}

int main() {
    box myBox;
    cout << "Enter width: " << endl;
    cin >> myBox;
    printWidth(myBox);
    return 0;
}
```

9. What happens if you call `new` but the heap is out of memory?
[< 5 min] [1pt]

10. What is the output of this code?

[< 5 min] [1pt]

```
#include < iostream >

void f(int i, int& j, const int& z)
{
    i = 0;
    j = i + z;
}

int main() {
    int i = 4;
    int j = 5;
    int z = 6;
    f(i, j, z);

    std::cout << i << j << z << std::endl;
}
```

11. What is the output of this code?

[< 5 min] [1pt]

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

class Player {
private:
    int id;
public:
    static int next_id;
    int getID() {  return id;  }
    Player() {
        id = next_id * 2;
        next_id++;
    }
};

int Player::next_id = 2;

int main() {
    Player p1;
    Player p2;
    Player p3;

    cout << p1.getID() << " ";
    cout << p1.next_id << " ";
    cout << p2.getID() << " ";
    cout << p2.next_id << " ";
    cout << p3.getID() << " ";
    cout << p3.next_id << " ";
    return 0;
}
```

12. What is an *automatic default constructor*, and what does it do?
[< 5 min] [1pt]

13. When is it appropriate to use *a const reference parameter*? Give a small example as part of your answer.
[< 5 min] [1pt]

14. The bag class is defined as follows:
[< 10 min] [2pt]

```
class bag {
public:
    // TYPEDEFS and MEMBER CONSTANTS
    typedef int value_type;
    typedef std::size_t size_type;
    static const size_type CAPACITY = 30;

    // CONSTRUCTOR
    bag() {  used = 0;  }

    // MODIFICATION MEMBER FUNCTIONS
    size_type erase(const value_type & target);
    bool erase_one(const value_type & target);
    void insert(const value_type & entry);
    void operator += (const bag & addend);

    // CONSTANT MEMBER FUNCTIONS
    size_type size() const {  return used;  }
    size_type count(const value_type & target) const;

private:
    value_type data[CAPACITY]; // The array to store items
    size_type used; // How much of array is used
};
```

Please answer the following questions:

- Is this a *correct* implementation? Explain your answer and write a solution if the implementation is wrong.

```
void  bag::operator += (const bag& addend)  {
// Precondition:  size( ) + addend.size( ) <= CAPACITY.
// Postcondition: Each item in addend has been added to this bag.

    size_type  i;  // An array index

    assert(size()  +  addend.size()  <=  CAPACITY);
    for  (i  =  0;  i  <  addend.used;  ++i)  {
        data[used]  =  addend.data[i];
        ++used;
    }
}
```

- Implement the following function. Note that the ordering of items *is not* important.

```
bool erase_one(const value_type & target);
// Postcondition: One copy of target has been removed from the bag.
// The return value is true if the item has been removed successfully.
```

15. *Heap variables are essentially global is scope.* Explain why and show how a dynamic variable allocated in function `f1` can be used in function `f2`.

[< 5 min] [2pt]

16. Here is a function prototype and some possible function calls: [< 1 min] [1pt]

```
int day_of_week(int year, int month = 1, int day = 1);
// Possible function calls:
cout << day_of_week();
cout << day_of_week(1995);
cout << day_of_week(1995, 10);
cout << day_of_week(1995, 10, 4);

```

How many of the function calls are *legal*?

- A. None of them are legal
- B. 1 of them is legal
- C. 2 of them are legal
- D. 3 of them are legal
- E. All of them are legal

17. Who needs to know about the *invariant* of an ADT? [< 1 min] [1pt]

- A. Only the programmer who implements the class for the ADT.
- B. Only the programmer who uses the class for the ADT.
- C. Both the programmer who implements the class and the programmer who uses the class.
- D. Neither the programmer who implements the class nor the programmer who uses the class.

[EXTRA CREDIT]

18. When should a pointer parameter p be a *reference* parameter? [< 1 min] [1pt]

- A. When the function changes p, and you want the change to affect the actual pointer argument.
- B. When the function changes p, and you do NOT want the change to affect the actual pointer argument.
- C. When the function changes *p, and you want the change to affect the object that is pointed at.
- D. When the function changes *p, and you do NOT want the change to affect the object that is pointed at.
- E. When the pointer points to a large object.