CMPS 2020 S’18 Sample Midterm I Hw6

1. Define encapsulation, abstraction and interfaces. Explain how these three concepts relate to classes and object-oriented programming.
2. You have a program with four variables: p1,p2,v1,v2. p1 and p2 are pointers while v1 and v2 are normal variables. Currently, the variables are stored in memory as shown in the following:

Variable Name	p1	p2	v1	v2
Value Stored	0x0	0x0	50	40
Memory Address	0x20	0x24	0x28	0x32

Show how each of the following lines of code would affect the above picture. Each of the lines of code is evaluated in the order presented (i.e. before (b) is executed, (a) has occurred).

a) p1 = &v1;

Variable Name	p1	p2	v1	v2
Value Stored				
Memory Address	0x20	0x24	0x28	0x32

b) p2= p1;

Variable Name	p1	p2	v1	v2
Value Stored				
Memory Address	0x20	0x24	0x28	0x32

c) *p2 = 99;

Variable Name	p1	p2	v1	v2
Value Stored				
Memory Address	0x20	0x24	0x28	0x32

d) p2 = &v2;

Variable Name	p1	p2	v1	v2
Value Stored				
Memory Address	0x20	0x24	0x28	0x32

e) What is outputted: cout << *p2 << “ “ << p1 << “ “ << v1 << endl;

3. You have the class EmployeeRecord with the member variables name, ID, hours and rate. You wish to add the input operator >> to the class.
   a) Show the line you would need to add to the class definition for the operator.
   b) Show the body of the operator function
4. Define the following terms and state the purpose of using the item in your class code when a class contains a *dynamic member variable*:
   a) default constructor
   b) copy constructor
   c) destructor
   d) overloaded assignment operator
5. Define the following terms by stating their purpose when used in your code:
   a) Parent / Base Class
   b) Polymorphic Function
   c) Redefined Function
6. Polymorphic functions differ from normal functions due to runtime binding of the function call to the function body. Describe how dynamic/runtime/late binding differs from the static binding which is used for normal or redefined functions.
7. When using inheritance, there are two forms of protection for member variables and functions. The first form of protection is the protection tag used for the section within the parent class. The second form is the inheritance protection tag used by the child class when inheriting from the parent. For the following inheritance scenarios, answer ”Yes” if the section is accessible or ”No” if it is not accessible:

	Accessible to child?	Accessible to grandchild?	Accessible to world?
Protected section in parent, inherited publicly			
Public section in parent, inherited publicly			

8. Add the copy constructor and destructor to the following MyString class. Make sure that your code properly handles the pointer for the dynamically sized character array.

```
class MyString {
  private:
    char* str;
    int size;
    void allocateArray(int);
  public:
    MyString();
    /****Add prototypes for copy constructor and destructor****/
};
MyString::MyString() {
  str = NULL;
  size = 0;
  allocateArray(21);
}
```

```
void MyString::allocateArray(int num) {
  if(num < 0 || num <= size) return;
  if(str!=NULL) {
    delete [] str;
    str = NULL;
    size = 0;
  }
  try{
    str = new char[num];
  }catch(bad_alloc) {
    cout << “Bad Allocation\nExiting…\n”;
    str = NULL;
    exit(1);
  }
  size = num;
  str[0] = ‘\0’;
}
/****Add the bodies for the copy constructor and destructor ****/
```

9. You have a parent class Employee which contains a protected member variable for the name. There is also a derived class *SalariedEmployee* which contains a member variable for the monthly salary. Add the following specified features to these two classes.
   a) Give the function prototype that would be added to both class definitions to add a polymorphic void printEmployee() function.
   b) Give the class *SalariedEmployee* line which would inherit from Employee publicly.
   c) Assume that Employee contains a default constructor which sets the name to an empty string. Give the body of the default constructor for *SalariedEmployee* which invokes Employee’s default constructor, then sets *salary* to zero.
   d) Assume that Employee contains an assignment operator which appropriately copies the name from the source object. Give the body of the assignment operator for *SalariedEmployee*. The assignment operator for *SalariedEmployee* must call Employee’s assignment operator.

10. Write an empty exception class called DivideByZero. Next write a function called *divide* that accepts two doubles and returns the quotient. If the second parameter (the denominator) is equal to zero, throw an instance of a DivideByZero object, otherwise, return the quotient. Next, write a snippet of code using a try catch block that will attempt to call the divide function. Setup a catch block that will catch a DivideByZero object and print to the screen “Cannot divide by zero\n”.