University of Calgary
Department of Electrical and Computer Engineering
ENSF 480: Principles of Software Design
Midterm Exam Solutions – Fall 2019

SECTION 1 (9 marks) – Multiple Choice Questions. Select the best answer:

1. What type of function can be declared as a **final** function in C++.
   A. Virtual member function
   B. Pure virtual member function
   C. static member function
   D. A and B are both correct
   E. None of the above

2. Class A is a subclass of B, and B is subclass of C and C is subclass of D. If all of these classes have their own `ctor` and `dtor,` and if an object of D goes out of scope what is the order of calls to the destructors of these classes:
   A. D, C, B, A
   B. A, B, C, D
   C. Only D
   D. None of the above

3. A public static data member in a C++ class:
   A. Must be declared globally outside the class definition
   B. Will be initialized to zero automatically
   C. All of the above
   D. None of the above are correct

4. Which one of the following statements is correct? Select the best answer.
   A. An abstract class is a class that contains a virtual function.
   B. An abstract class is a class that contains at least one pure virtual function.
   C. An abstract class is class that is derived from a virtual base class
   D. A, and B are correct answers
   E. All of the above are correct answers
   F. None of the above is a correct answer

5. Which one of the following statements is correct about a static member function in a C++ class? Select the best answer.
   A. A static member function can be called even without existence of a class object.
   B. A static member function doesn't have direct access to the class private data members
   C. A static member function does not have a 'this pointer'
   D. B and C are correct answers
   E. All of the above are correct answers
   F. None of the above

6. Which one of the following statements is correct?
   A. An abstract class is a class that is at the root of the class hierarchy
   B. An abstract class is a class that cannot have an object.
   C. An abstract class can be a reference or pointer argument of a global function
   D. B and C are both correct.
   E. All of the above are correct
   F. None of the above is correct

7. Which one of the following statements is correct?
   A. An overloaded operator cannot be declared static
   B. Operators << and >> cannot be defined as a member of a class
   C. Operators =, (), and = must be defined as a member of class
   D. Only B and C are the correct answers
   E. All of the above are correct answers
   F. None of the above are correct answers

8. Which one of the following statements is correct about a final class in C++:
   A. C++ doesn't support final classes
   B. A final class in C++ is class that cannot be a base class
   C. A final class is class that its member function cannot be overridden.
   D. B, and C are both correct answers
   E. None of the above is a correct answer

9. Which one of the following statements is a correct in C++?
   A. An overloaded operator function must be always a member of a class or a friend of a class
   B. An overloaded operator function must have at least a class objec as its argument.
   C. An overloaded operator cannot return a pointer, but can return a reference.
   D. A and B are correct answers
   E. All of the above are correct answers
   F. None of the above is a correct answer

SECTION 2 (15 marks)

Consider the fillowing class definition which is similar to what we used during lecturs, then answer the following questions;

```
Class String {
    char* storageM;
    char* cursorM ;              // cursor to traverse along the stroageM elements
public:
    bool cursor_ok();            // returns true if cursorM is not pointing to '\0'
    void set_cursor();           // sets cursorM back to the beginning of storageM
    String (const char* s);      // allocates memory and copies s into storageM. Also
                                 // sets cursorM to the first element of storageM
};
```

```
int main (){
  String text = "  be aware of bugs  "; // creating an object of String
  text.set_cursor();
  while(text.cursor_ok())  // printing words in the given text
    cout << text++ << endl;
  // creating an object of String with 20 elements all filled with '\0'
  String word = 20;
  cout << "Please enter a word with no spaces (max 19 characters): " << endl;
  cin >> word;    // using istream object cin to read a word
  ...             // More code
  return 0;
}
```

// this box shows the program output. Also shows user input, which is: **Apple**

```
be
aware
of
bugs
Please enter a word with no spaces (max 19 characters):
Apple
Output is: Apple
```

In the follwing space ONLY write the definition of any fucntion that is needed to allow the given main funciotn work. **Marks will be deducted for writtng unnesary functions. Don't write any funtion that its prototype is given, and don't worry about adding prototypes of the fucntons that you write, into the class definiton.**

Because of 3 lines highlighed in red we need to define ONLY three operators/funcitons to allow the given main function work:

- `String::operstor ++  (int)`
- `operator >> (istream&, const String&)`
- `String::String(int)  // this constructor works like implicit type cast operator`

9 marks – This is one possible solution

```
char* String::operator ++(int){
        char* end = cursorM;
        char* sub;
        // skip the leading spaces
        while(isspace(*end) and *end != '\0')
            end++;
        // find the end of next word
        cursorM = end;
        while (*end != '\0') {
            if(!isspace(*end))
                end++;
            else
                break;
        }
        // find the length of word
        int length = int(end - cursorM);
        // allocate memory and copy the word
        sub = new char[length + 1];
        for(int i=0; i < length; i++)
            sub[i] = cursorM[i];
        sub[length] = '\0';

        cursorM = end;
        // return a word
        return sub;
}
```

3 marks

```
istream& operator >> (istream& in, const String& text){
    in >> text.storageM;
    return in;
}
```

3 marks

```
String::String (int size){
   storageM = new char[size + 1];
   assert(storageM !- nullptr);
   for(int i=0 ; i < size ; i++) storageM[i] = '\0';
   cursorM = storageM;
}
```

SESCTION 3 (12 marks)
Consider the following code segment, which belongs to a portion of legacy software. Now you should refactor it to conform to basic four pillars of object-oriented design (abstraction, encapsulation, modularity, hierarchy):

```
struct Company{
    string companyName;
    string companyAdderss;
    vector<string> employees;       // vector of information about employees
                                    // (name, address, date of birth)
    string dateStableisEstablished; // the date that company was established
    vector <string> employeeState;  // information about employees' current states
                                    //(active, suspended, retired, fired)
    vector <string> customers       // vector of information about customers
};                                  // (name, address, email)
```

In the following space write your refactored code.
Note: For the simplicity purposes you don't need to worry about any functions in your code.
NOTE TO MARKER: 1) consider 2 marks for each class. 2)If using struct instead of class deduct 2 marks if access inside the structures are not explicitly declared private or protected. If access is declared private or protected deduct 1 mark. 3) Make sure each class has proper access level. 4) Deduct 1 mark if a class for employeeState is provided.

```
class Name {
private:
    string firstname;
    string midlename;
    string lastname;
};

class Address {
private:
    int number;
    string street;
    string city;
    string province
    string postalCode;
};


class Person{
protected:
    Name name;
    // other info if needed
};


class Customer: public Person{
private:
    Address address;
    // other info as needed
};


class Employee: public Person{
private:
    Address address;
    string employeesStates;
    // other info as needed
};


class Date {
private:
    int day;
    int month;
    int year;
};


class Company {
private:
    string companyTitle;
    Address address;
    vector<Employee> employees;
    vector<Customer> custormers;
    Date dateStableished;
};
```

SECTION 4 – Using Strategy Pattern (12 marks)

Consider the following simple C++ code that uses three class called Animal, Cat and Fish. As given code shows each animal's move is strongly bound to its implementation and if you want to change the animal's move behaviour there is no way other than changing the given the code in the animal classes. In this section you should refactor this program using **Strategy Pattern** to allow more flexibility for changes. In other words we should be able to set or change the move behaviour of an animal at the runtime. For example if a cat's move style is currently "walking", we should be able to change it to "swimming" at the runtime (within main function).

```
class Animal {
protected:
    string name;
public:
  Animal(string name){this->name=name;}
  virtual void move() = 0;
};

class Cat: public Animal {
public:
  Cat(string name): Animal(name){}
  void move(){
     cout<<"Walking-"<<name<<endl;;
  }
};

class Fish: public Animal {
public:
  Fish(string name): Animal(name){}
  void move(){
     cout<<"Swimming-"<<name <<endl;
  }
};
```

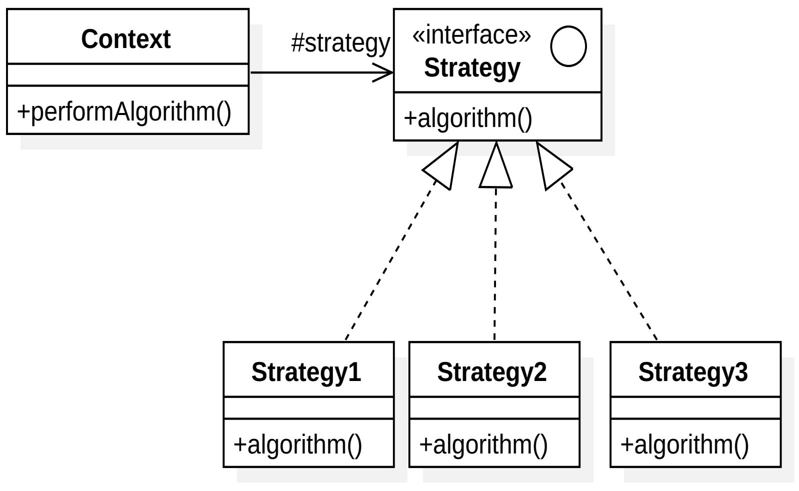
Using Animal objects

```
int main(){
    Cat kitty ("Cat");
    kitty.move();
    Fish fishy ("Fish");
    fishy.move();
    return 0;

}
```

Program output

```
Walking-Cat
Swimming-Fish
```

To refresh your memory here is the example model of Strategy Design Pattern:



To better understand how your refactored code should work, the following pseudo-code of a main function and its expected output is given:

Pseudo-code for a main function that uses strategy pattern

```
Create an object of Cat called kitty
Set move-strategy of walking to kitty
kitty performs move (means prints Moving-Cat)
Create an object Fish called fishy;
Set move-strategy of swimming to fishy
fishy performs move (means prints Swimming-Fish)
Now set the move strategy of walking to fishy
fishy performs move (means prints Walking-Fish)
```

Output

```
Walking-Cat
Swimming-Fish
Walking-Fish
```

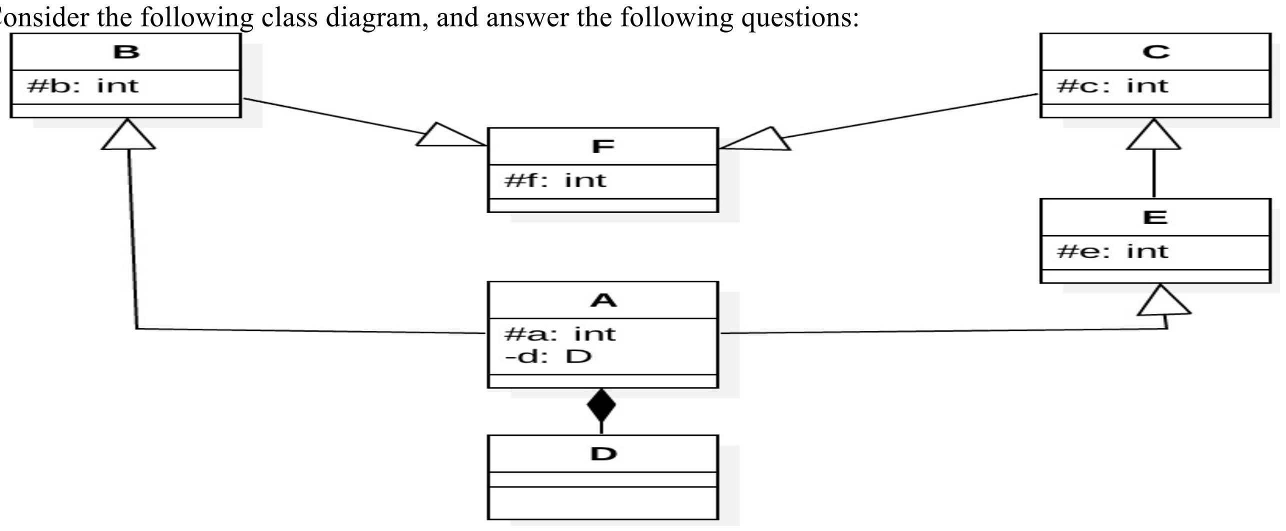
Now on the following page (page 7), rewrite the above classes based on using Strategy Pattern. Also write a main function that applies the above pseudo-code, and shows how your strategy pattern works.

SECTION 4 Solution

```
//--------------------- Strategy Classes ----------------------------------
class MoveStrategy {
public:
    virtual void move(string s) = 0;
};

class Walker: public MoveStrategy {
    void move(string s) {
        cout << "Walking" << "-" << s << endl;
    }
};

class Swimmer: public MoveStrategy {
    void move(string s) {
        cout << "Swimming" << "-" << s << endl;
    }
};

// ---------------------- Animal Classes ----------------------------------
class Animal {
protected:
    MoveStrategy* moveStrategy; // if you don't declare it as pointer, then you
                                // MUST initialize it in the constructor
    string name;
public:
    Animal(string name) {this->name = name;}        // It is OK to set the strategy here
    virtual void setStrategy(MoveStrategy& m) = 0; // It is OK to implement the function here
    virtual void performMove() = 0;                // must be virtual
};


class Cat: public Animal {
public:
    Cat(string name): Animal(name){}
    void setStrategy(MoveStrategy& m){
        moveStrategy = &m;
   }
   void performMove() {
        moveStrategy->move(name);
   }
};


class Fish: public Animal {
public:
    Fish(string name): Animal(name){}

    void setStrategy(MoveStrategy& m){
        moveStrategy = &m;
    }

    void performMove() {
        moveStrategy->move(name);
    }
};

// ----------------------- Client Method or Class -----------------------------
int main(){
    Cat kitty ("Cat");
    Walker w;
    kitty.setStrategy(w);
    kitty.performMove();
    Fish fishy ("Fish");
    Swimmer s;
    fishy.setStrategy(s);
    // prints Swimming Fish
    fishy.performMove();
    fishy.setStrategy(w);
    fishy.performMove();
    return 0;
}
```

SECTION 5 (5 marks):

Consider the following class diagram, and answer the following questions:



a) Assuming that we need to use objects of classes A, B, C, D, E, and F, which class(s) need to have a virtual base class. (2 marks)?

Class F must be virtual base class for both classes B and C

b) Assume each class has its own `ctor` for initializing its data member and its own `dtor` for some sort of cleanup. Also for the simplicity purposes assume that general format of class `ctors` is: `X(int x)`.
In the following space write the implementation of `ctor` for class A (2 marks).

```
A::A(int x, int y, int z, int w): F(x), B(y), E(z), d(w) {

}
```

c) Class A is declared with the label {leaf}. What this label mean (1 mark)

Leaf label indicates that class A is a final class.

SECTION 6 (7 marks)

Consider the following program and determine what is the program's output. Write your answer in the lower box on the right:

```
class A {
public:
 void fun1(){
     cout << "A fun1\n";
 }
 virtual void fun2(){
     cout << "A fun2\n";
 }
 void fun3(){
     cout << "A fun3\n";
 }
};
```

```
class B: public A {
public:
   void fun1(){
       cout << "B fun1\n";
   }
   void fun2(){
       cout << "B fun2\n";
   }
   virtual void fun3(){
       cout << "B fun3\n";
   }
};
```

```
class C:public  B {
public:
   void fun1(){
       cout << "C fun1\n";
   }
   void fun2(){
       cout << "C fun2\n";
   }
   void fun3(){
       cout << "C fun3\n";
   }
};
```

```
void bar(A& x, A& y, B& z)
{
    x.fun1();
    x.fun2();
    y.fun1();
    y.fun2();
    y.fun3();
    z.fun1();
    z.fun2();
}
```

```
int main() {
    A myA;
    B myB;
    C myC;
    bar(myA, myB, myC);
    return 0;
}
```

// Write your answer here:

```
A fun1
A fun2
A fun1
B fun2
A fun3
B fun1
C fun2
```