[*Practice Problems for Classes*]

1. Create a class called **Animal** that holds information about the animals in a zoo. The private attributes should be **age**, which represents the age of the animal, **numOwners**, and **prevOwners**, which corresponds to the names of the animals' previous owners. You must include the default constructor, copy constructor, copy assignment, move constructor, move assignment and destructor. Create methods that set and get the age, numOwners and the i^{th} previous owner. The default constructor can accept an age and a numOwners if given, if nothing is given the default should be 0 for both. When changing the number of previous owners remember that this should also update the sequence that holds the names of the previous owners.

Solution on next page

```
#include<iostream>
#include<string>

class Animal{
private:
  int age;
  int numOwners;
  std::string* previousOwners;

public:
  Animal(const int & =0, const int & =0);
  Animal(const Animal &);
  void operator=(const Animal &);
  Animal(Animal &&);
  void operator=(Animal &&);
  ~Animal();
  void setAge(const int &);
  void setNumOwners(const int &);
  void setPreviousOwner(const int &, const std::string &);
  int getAge() const;
  int getNumOwners() const;
  std::string getPreviousOwner(const int &) const;
};

Animal::Animal(const int &a, const int &n){
  age = a;
  numOwners = n;
  if(numOwners > 0){
    previousOwners = new std::string[numOwners];
  }
  else{
    previousOwners = nullptr;
  }
}

Animal::Animal(const Animal & copyFrom){
  age = copyFrom.age;
  numOwners = copyFrom.numOwners;
  if(numOwners > 0){
    previousOwners = new std::string[numOwners];
    for(int i=0; i<numOwners; ++i){
      previousOwners[i] = copyFrom.previousOwners[i];
    }
  }
  else{
    previousOwners = nullptr;
  }
}

void Animal::operator=(const Animal & copyAssignFrom){
  age = copyAssignFrom.age;
  numOwners = copyAssignFrom.numOwners;
  if(previousOwners != nullptr){
    delete[] previousOwners;
  }
  if(numOwners > 0){
```

```
    previousOwners = new std::string[numOwners];
    for(int i=0; i<numOwners; ++i){
      previousOwners[i] = copyAssignFrom.previousOwners[i];
    }
  }
  else{
    previousOwners = nullptr;
  }
}

Animal::Animal(Animal && moveFrom){
  age = moveFrom.age;
  moveFrom.age = 0;
  numOwners = moveFrom.numOwners;
  moveFrom.numOwners = 0;
  previousOwners = moveFrom.previousOwners;
  moveFrom.previousOwners = nullptr;
}

void Animal::operator=(Animal && moveAssignFrom){
  age = moveAssignFrom.age;
  moveAssignFrom.age = 0;
  numOwners = moveAssignFrom.numOwners;
  moveAssignFrom.numOwners = 0;
  if(previousOwners != nullptr){
    delete[] previousOwners;
  }
  previousOwners = moveAssignFrom.previousOwners;
  moveAssignFrom.previousOwners = nullptr;
}

Animal::~Animal(){
  if(previousOwners != nullptr){
    delete[] previousOwners;
  }
}

void Animal::setAge(const int &a){
  age = a;
}

void Animal::setNumOwners(const int &n){
  if(n <= 0){
    if(previousOwners != nullptr){
      delete[] previousOwners;
      numOwners = 0;
    }
  }
  else{
    if(numOwners != n){
      int min = n;
      if(numOwners < n){
          min = numOwners;
      }
        std::string* newPrevOwners = new std::string[n];
        for(int i=0; i<min; ++i){
```

```
          newPrevOwners[i] = previousOwners[i];
        }
        if(previousOwners != nullptr){
          delete[] previousOwners;
        }
        previousOwners = newPrevOwners;
        numOwners = n;
    }
  }
}

void Animal::setPreviousOwner(const int &i, const std::string &name){
  previousOwners[i] = name;
}

int Animal::getAge() const{
  return age;
}

int Animal::getNumOwners() const{
  return numOwners;
}

std::string Animal::getPreviousOwner(const int &i) const{
  return previousOwners[i];
}

int main(){

  return 0;
}
```