1)

```
//standard without dynamic memory
class Account {
private:
   std::string name;
   std::vector<double> balances;
public:
   Account(std::string name) : name{name} {
       std::cout << "default constructor" << std::endl;
   }
   Account(const Account& other): name{other.name} {
       std::cout << "copy constructor" << std::endl;
   };
   Account& operator=(Account other) {
       std::cout << "copy assignment" << std::endl;
       swap(*this, other);
       return *this;
   }
   Account(Account&& other): name{move(other.name)}, balances{move(other.balances)} {
       std::cout << "move constructor" << std::endl;
   }
   Account& operator=(Account&& other) {
       std::cout << "move assignment operator" << std::endl;
       name = move(other.name);
       balances = move(other.balances);
       return *this;
   }
   friend void swap(Account& first, Account &second) {
       std::swap(first.name, second.name);
       std::swap(first.balances, second.balances);
   }
   friend std::ostream& operator<<(std::ostream& os, const Account& account) {
       os << account.name;
       return os;
   }
   friend std::istream& operator>>(std::istream& is, Account& account){
       is >> account.name;
       return is;
   }
   ~Account(){};
};


//with dynamic memory
class AccountDynamic {
private:
   std::string* name;
   std::vector<double>* balances;
public:
   AccountDynamic(std::string name) : name{&name} {
       std::cout << "default constructor" << std::endl;
   }
   AccountDynamic(const AccountDynamic& other): name{other.name} {
       std::cout << "copy constructor" << std::endl;
   };
   AccountDynamic& operator=(AccountDynamic other) {
       std::cout << "copy assignment" << std::endl;
       swap(*this, other);
       return *this;
   }
   AccountDynamic(AccountDynamic&& other): name{move(other.name)} {
       balances = other.balances;
       other.balances = nullptr;
       std::cout << "move constructor" << std::endl;
   }
   AccountDynamic& operator=(AccountDynamic&& other) {
       std::cout << "move assignment operator" << std::endl;
       name = move(other.name);
       balances = other.balances;
       other.balances = nullptr;
       return *this;
   }
   friend void swap(AccountDynamic& first, AccountDynamic &second) {
       std::swap(first.name, second.name);
       std::swap(first.balances, second.balances);
   }
   friend std::ostream& operator<<(std::ostream& os, const AccountDynamic& account) {
       os << *account.name;
       return os;
   }
   friend std::istream& operator>>(std::istream& is, AccountDynamic& account){
       is >> *(account.name);
       return is;
   }
   ~AccountDynamic(){
       std::cout << "deconstructor" << std::endl;
       delete balances;
       delete name;
   };
};
```

2) What is the difference between an rvalue and an lvalue? Write code that demonstrates the difference. Label your code carefully.

lvalue: exists beyond single expression, has an address.
Rvalue: temporary value, no address

```
int n;
n = 7; //OK because n is lvalue
(n+1) = 9; //Error because n+1 is rvalue
```

3) What is the difference between an rvalue reference and a pointer?
Rvalue references converts an lvalue to an rvalue, even if that something has a name.
Rvalue references indicates that an object may be "moved from"
A pointer does not force "move semantics" and does not convert lvalue into an rvalue

4) What is/are the difference(s) between value semantics and move semantics?
**Value semantics** lets us pass objects by value instead of just passing references to objects
**Move semantics** provide a way for the contents of objects to be 'moved' between objects, rather than copied like in value semantics

5) How do enumerations in c++ deffer from enumerations in Java?

6) Write a snippet of code that uses a lambda expression to sort a list of string according to the length of the string, i.e., strings that are longer are "greater than" strings that are Shorter.

7. What is the difference between passing parameters to a lambda and capturing variables for a lambda? Why would we choose one over the other?

8. Define a template class called Record that is parameterized over two typenames K and V. A Record contains a K called key and a vector of V called values. Record has a single constructor which accepts a K – assign this K to key. Record has a member function called add which accepts a parameter of type V and adds it to values.

9. (Write a program that demonstrates the correct use of dynamic_cast to perform a cross cast operation. You will need to create a small inheritance hierarchy. Comment your code.

10. Implement a functor called Prime. Each time the operator( ) is invoked, the next prime number is returned. Assume the first prime number is 2. For example,
Prime p;
p( ); // returns 2
p( ); // returns 3
p( ); // returns 5
p( ); // returns 7