These answers are not entirely correct, do try to understand what is right and what might be wrong. The concepts, however, are mostly correct, remember what each question is looking for. Sometimes, questions may change minor details, so do not memorize the answers.

-Sundeep K.

C++ Final Review

EC1)

template <typename typeA>

int count(typeA\* start, typeA\* end, typeA target){

        int sum = 0;

        for(typeA\* i = start; i != end; i++){

                if(\*i == target){

                        sum += 1;

                }

        }

        return sum;

}

8) Object operator+(const Object& lhs, const Object& rhs){

Object temp;

temp += lhs;

temp += rhs;

return temp;

}

12) a)

Derived(const Derived& rhs) : Base(rhs) { foo = rhs.foo; }

b)

Derived& operator=(const Derived& rhs) {

foo = rhs.foo;

}

13) Use iterators to sum up a list. Then print the result.

list<int> iL;

list<int>::iterator iLIter;

int sum = 0;

for (iLIter = iL.begin(); iLIter!= iL.end(); iLIter++){

sum += \*iLIter;

}

14a) Recursion to sum up a tree.

struct TNode{

data;

leftBr;

rightBr;

}

int treeSum(TNode\* root){

if (root == nullptr){

return 0;

}

return (root 🡪 data + treeSum(root 🡪 left) + treeSum(root 🡪 right))

}

14b) Recursion to copy list.

Node\* copyList(Node\* headPtr){

Node\* temp;

if (headPtr == nullptr){return nullptr;}

temp 🡪 data = headPtr 🡪 data;

temp 🡪 link = copyList(headPtr 🡪 link;

return temp;

}

14c) Palindrome. See if a word is a palindrome and if yes, return a bool true.

bool palindrome(string word){

if (word.size() < 2){return true;}

if (word[0] == word[word.size()-1])

{

return palindrome(word.substr(word[1], word[word.size() – 2]));

}

return false;

}

15) Write a function to add two lists, store in a 3rd, and return it.

Node\* addLists(Node\* list1, Node\* list2){

if(list1 == nullptr || list2 == nullptr){

return nullptr;

}

Node\* temp;

temp 🡪 data = list1 🡪 data + list2 🡪 data;

temp 🡪 link = new Node(\*addLists(list1🡪 link, list2 🡪 link));

return temp;

16)

class Skyrim{

vector<Dragon\*> dragons;

};

Write = operator.

Skyrim& operator=(const Skyrim& rhs){

for (size\_t i = 0; i < dragons.size(), i++){

delete dragons[i];

}

dragons.clear();

for (size\_t i = 0; i < rhs.dragons.size(); i++){

dragons.push\_back(new Dragon(\*rhs.dragons[i]));

}

return \*this;

}

Write << operator.

friend ostream& operator<<(ostream& os, const Skyrim& rhs){

for (size\_t i = 0; i < dragons.size(); i++){

os << \*dragons[i] << endl;

}

return os;

}

17. class Performer{

public:

Performer(const string& name) : myName() {}

virtual void perform() = 0;

private:

string name;

};

class Actor : public Performer{

public:

Actor(const string& name, const string& script) : Performer(name) {}

void perform(){

cout << "I am " << name << ". Script: " << script << endl;

}

private:

string script;

};

class Mime : public Performer{

public:

Actor(const string& name) : Performer(name) {}

void perform(){

cout << "BLANK" << endl;

}

private:

};

class Theater{

public:

Theater() {}

void showtime(){

for (size\_t i = 0; i < performers.size(); i++){

performers->perform();

}

}

void hire(Performer\* newPerformer){

performers.push\_back(newPerformer);

}

private:

vector<Performer\*> performers;

};