1)

(a) F

(b) T

(c) F T

(d) T

(e) F

(f) F

(g) F

(h) T

(i) T

(j) F

2)

<typename T, int N>

T val;

Ntree subtree[N];

node() {}

node(const T& x) : value(x) {}

3)

(a)

for (; first != last; first++) {

if (p(\*first)) \*first = new\_value;

}

(b)

class LessThan {

private:

int threshold;

public:

LessThan(int t) :threshold(t) {}

bool operator()(int val) {return (val < threshold);}

};

(c)

replace\_if(marks.begin(), marks.end(), LessThan(0), 0);

4)

25

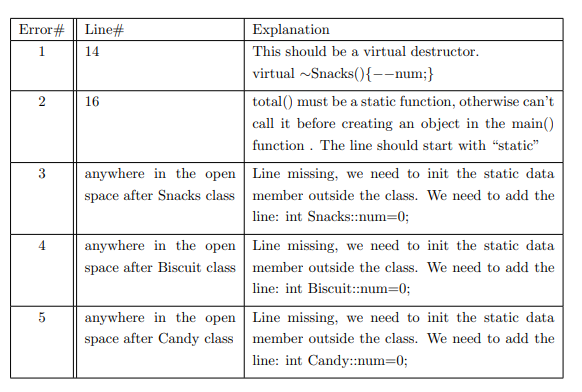
Biscuit::num

26

38

Candy::num

39



5)

template <typename T>

bool Set::operator==(const Set& s) const {

if (set.size() != s.set.size()) return false;

for (int i = 0; i < set.size(); i++)

if (set[i] != s.set[i]) return false;

return true;

}

template <typename T>

Set Set::operator+(const Set& s) const {

Set t; t.set = set;

for (int i = 0; i < s.set.size(); i++) {

if (!t.contains(s.set[i])) t.set.push\_back(s.set[i]);

}

sort(t.set.begin(), t.set.end());

return t;

}

missed one

template <typename T>

Set& Set::operator+=(const Set& s) {

return \*this = \*this + s;

}

template <typename T>

Set& Set::operator+=(const T& item) {

if (!contains(item)) {

set.push\_back(item);

sort(set.begin(), set.end());

}

return \*this;

}

template <typename T>

bool Set::contains(const T& item) const {

return (find(set.begin(), set.end(), item) != set.end());

}

template <typename T>

friend ostream& operator<<(ostream& os, const Set& s) {

os << "[ ";

vector<int>::const\_iterator it = s.set.cbegin();

os << \*it;

for (it++; it != s.set.cend(); it++) {

os << ", " << \*it;

}

os << " ]";

return os;

}

6)

(c)

template <typename T>

int BST::height() const {

if (empty()) return -1;

return (root->left.height() > right.height()) ? (left.height() + 1) : (1 + right.height());

}

template <typename T>

void print\_leaves() const {

if (empty()) return;

if (root->left.empty() && right.empty()) cout << root->val << " ";

left.print\_leaves();

right.print\_leaves();

}

template <typename T>

bool is\_avl() {

if (empty()) return true;

if (left.height() - right.height() >= 2 || left.height() - right.height() <= -2) {

return false;

} return true; root->left.is\_avl() && root->right.is\_avl();

}

template <template T>

void inorder\_print\_if(bool (\*f)(int)) {

if (empty()) return;

left.inorder\_print\_if(f);

if (f(val)) cout << val << " ";

right.inorder\_print\_if(f);

}

7)