COMP 2012H Final Exam - Fall 2021 - HKUST

3hrs

1:31

1)

(a) F

(b) F

(c) T

(d) F

(e) T

(f) T

(g) ? F

(h) ? T F

(i) T

(j) F (?) T

2)

(a)

stack<double> temp;

double tempElm;

if (s.empty()) {

s.push();

} else {

while (!s.empty()) {

tempElm = s.top(); s.pop();

if (tempElm < x) {

s.push(tempElm);

s.push(x);

while (!tempElm.empty()) {s.push(temp.top()); temp.pop();}

break;

} else {

temp.push(tempElm);

}

if (s.empty()) {

s.push(x);

while(!tempElm.empty()) {s.push(temp.top()); temp.pop();}

}

}

}

(b)

stack<double> temp;

while (!s.empty()) {

sortedInsert(temp, s.top()); s.pop();

}

s = temp;

3)

C's default

B's conv

D's default

---- Before mystery ----

C's default

B's conv

A's conv

D's other

C's copy

B's copy

A's copy

D's copy

D's dtor

A's dtor

After delete

B's dtor

C's dtor

---- After mystery ----

D's dtor

A's dtor

After delete

B's dtor

C's dtor

D's dtor

~~A's dtor~~

After delete

B's dtor

C's dtor

4)

(a)

Iterator temp = sfirst;

int count = 0;

while (temp != slast) {

if (predicate(\*temp)) {

\*(dfirst++) = \*temp;

count++;

}

temp++;

}

return count;

(b)

class IsPrimeGreaterThan {

private:

unsigned int threshold;

public:

IsPrimeGreaterThan(unsigned int thresh) : threshold(thresh) {}

bool operator()(unsigned int x) {

if (x == 0 || x == 1) return false;

if (x <= threshold) return false;

for (int i = 2; i <= x / 2; i++) {

if (x % i == 0) return false;

}

return true;

}

}

(c)

copy\_if

int count = count\_if(v.begin(), v.end(), prime.begin(), IsPrimeGreaterThan(6));

5)

0

1 52

2 18

3 78

4

5 109

6

7 40

8 8

9 31

10 19

19: 1

18: 2

109: 1

52: 1

78: 2

total = 7

10

6)

(a)

CourseShoppingCart::CourseShoppingCart(const int capacity) {

courseList = new Course\*[capacity];

this->capacity = capacity;

size = 0;

}

(b)

CourseShoppingCart::CourseShoppingCart(const CourseShoppingCart& csc) {

courseList = nullptr; size = 0;

if (this != &csc) {

\*this = csc;

}

}

(c)

~CourseShoppingCart::CourseShoppingCart() {

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

delete courseList[i];

delete [] courseList;

}

(d)

const Course\* CourseShoppingCart::operator[](int i) const {

if (i >= 0 && i < size)

return courseList[i];

exit(-1);

}

(e)

CourseShoppingCart& CourseShoppingCart::operator=(const CourseShoppingCart\* csc) {

if (this != &csc) {

for (int i = 0; i < size; i++) delete courseList[i];

delete [] courseList;

courseList = new Course\*[csc.capacity];

capacity = csc.capacity;

size = csc.size;

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

if (typeid(csc.courseList[i]) == typeid(CommonCoreCourse\*)) {

courseList[i] = new CommonCoreCourse(&dynamic\_cast<CommonCoreCourse\*>(csc.courseList[i]))

} else {

courseList[i] = new CompCourse(&dynamic\_cast<CompCourse\*>(csc.courseList[i]));

}

}

}

return \*this;

}

(f)

void CourseShoppingCart::operator+=(Course\* course) {

if (size == capacity) {

Course\*\* newArray = new Course\*[capacity \* 2];

for (int i = 0; i < size; i++) newArray[i] = courseList[i];

delete [] courseList;

courseList = newArray;

capacity \*= 2;

}

courseList[size++] = course;

}

(g)

friend ostream& operator<<(ostream& os, const CourseShoppingCart& csc) {

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

courseList[i]->print();

~~cout~~ os << " (For all? " << courseList[i]->isForAll() ? "true" : "false" << ")" << endl;

}

return os;

}

7)

(a)

template <typename T1, typename T2>

const pair<T1,T2> BSTD::find\_min() const {

BSTDnode\* node = root;

while (!root->left.is\_empty()) {

node = node->left.root;

}

while (!root->same.is\_empty()) {

node = node->same.root;

}

return pair<T1,T2>(node->key, node->data);

}

(b)

template <typename T1, typename T2>

void BSTD::insert(const T1& x, const T2& y) {

BSTDNode\* node = root;

while (true) {

if (node->val > x) {

if (!node->left.is\_empty()) {

node = node->left.node;

} else {

node->left.node = new BSTNode{x, y};

return;

}

} else if (node->val < x) {

if (!node->right.is\_empty()) {

node = node->right.node;

} else {

node->right.node = new BSTNode{x, y};

return;

}

} else {

while (!node->same.is\_empty())

node = node->same.node;

node->same.node = BSTNode{x, y};

return;

}

}

}

(c)

later