**Homework 3 Test Data**

For linear.cpp, there were 51 test cases, each worth 1 point; for tree.cpp, 11 cases, each worth 2 points; for maze.cpp, 4 test cases, each worth 3 points; and for investment.cpp, 5 criteria, each worth 3 points.

The investment.cpp criteria were

1. There must not be any values X and Y for which new Investment(X, Y) compiles — Investment must be abstract.
2. Investment must not have a default constructor.
3. Investment must not have a constructor with other than two arguments.
4. Investment must have a virtual destructor.
5. No two function with non-empty bodies may have the same effect. In particular, this prohibition forbade you from implementing both Painting::fungible and House::fungible as { return false; }; they should have inherited that implementation from a common base class (probably Investment).

We concatenated these lines:

#include <string>

using namespace std;

bool somePredicate(double d);

then linear.cpp, tree.cpp, maze.cpp, and this file to run the tests of those three files:

#include <iostream>

#include <streambuf>

#include <string>

#include <algorithm>

#include <cstdlib>

#include <cassert>

using namespace std;

bool somePredicate(double x)

{

return x > -10;

}

void testone(int n)

{

static string maze1[10] = {

{ "XXXXXXXXXX" },

{ "X.X..X...X" },

{ "X.XX.X.XXX" },

{ "X....X.X.X" },

{ "XX.X.X...X" },

{ "XXX..X.X.X" },

{ "X...X...XX" },

{ "X.XX..X.XX" },

{ "X....X...X" },

{ "XXXXXXXXXX" },

};

static string maze2[10] = {

{ "XXXXXXXXXX" },

{ "X.X..X...X" },

{ "X.XX.X.XXX" },

{ "X....X.X.X" },

{ "XX.X.X...X" },

{ "XXX.XX.X.X" },

{ "X...X...XX" },

{ "X.XX..X.XX" },

{ "X....X...X" },

{ "XXXXXXXXXX" },

};

static string maze3[10] = {

{ "XXXXXXXXXX" },

{ "XX.....XXX" },

{ "X..XX....X" },

{ "X...X...XX" },

{ "X.X.XXX..X" },

{ "XXXX..X..X" },

{ "XX....X..X" },

{ "X.......XX" },

{ "X..XXXXXXX" },

{ "XXXXXXXXXX" },

};

static string maze4[10] = {

{ "XXXXXXXXXX" },

{ "XX.....XXX" },

{ "X..XX....X" },

{ "X...X...XX" },

{ "X.X.XXX..X" },

{ "XXXX..X..X" },

{ "XX....X..X" },

{ "X.....X.XX" },

{ "X..XXXXXXX" },

{ "XXXXXXXXXX" },

};

double a[10] = { 5, -5, -10, 6, -10, -6, -10, -10, -7, 7 };

double x[10] = { 2, 7, 4, 8, -10, 0, -10, -10, 9, 6 };

double y[10] = { 8, 10, 0, 9, 4, 5, -10, 2, 12, 6 };

double b[4] = { 10, 0, 4, 2 };

double c[8] = { 1, 9, 7, 3, 9, 7, 5 };

double d[12] = { 1, 3, 7, 1, 7, 5, 3, 1, 7, 9, 7, 5 };

double f[16] = { 4, 7, 0, 15, 2, 4, 5, 14, 10, 1, 0, 6, 12, 14, 4, 8 };

double g[16] = { 15, 14, 14, 12, 10, 8, 7, 6, 5, 4, 4, 4, 2, 1, 0, 0 };

switch (n)

{

default: {

assert(false);

} break; case 1: {

assert(!allTrue(a, 10));

} break; case 2: {

assert(!allTrue(a, 8));

} break; case 3: {

assert(!allTrue(a+2, 8));

} break; case 4: {

assert(allTrue(a+8, 2));

} break; case 5: {

assert(!allTrue(a+6, 2));

} break; case 6: {

assert(allTrue(a, 1));

} break; case 7: {

assert(!allTrue(a+2, 1));

} break; case 8: {

assert(allTrue(a, 0) && allTrue(a+2, 0));

} break; case 9: {

assert(countFalse(a, 10) == 4);

} break; case 10: {

assert(countFalse(a, 8) == 4);

} break; case 11: {

assert(countFalse(a+2, 8) == 4);

} break; case 12: {

assert(countFalse(a+8, 2) == 0);

} break; case 13: {

assert(countFalse(a+6, 2) == 2);

} break; case 14: {

assert(countFalse(a, 1) == 0);

} break; case 15: {

assert(countFalse(a+2, 1) == 1);

} break; case 16: {

assert(countFalse(a, 0) == 0 && countFalse(a+2, 0) == 0);

} break; case 17: {

assert(firstFalse(a, 10) == 2);

} break; case 18: {

assert(firstFalse(a, 8) == 2);

} break; case 19: {

assert(firstFalse(a+2, 8) == 0);

} break; case 20: {

assert(firstFalse(a+8, 2) == -1);

} break; case 21: {

assert(firstFalse(a+6, 2) == 0);

} break; case 22: {

assert(firstFalse(a, 1) == -1);

} break; case 23: {

assert(firstFalse(a+2, 1) == 0);

} break; case 24: {

assert(firstFalse(a, 0) == -1 && firstFalse(a+2, 0) == -1);

} break; case 25: {

assert(indexOfMin(x, 6) == 4);

} break; case 26: {

assert(indexOfMin(x, 5) == 4);

} break; case 27: {

assert(indexOfMin(x+7, 3) == 0);

} break; case 28: {

assert(indexOfMin(x+8, 2) == 1);

} break; case 29: {

assert(indexOfMin(x+2, 2) == 0);

} break; case 30: {

assert(indexOfMin(x, 1) == 0);

} break; case 31: {

assert(indexOfMin(x+4, 1) == 0);

} break; case 32: {

assert(indexOfMin(x, 10) == 4);

} break; case 33: {

assert(indexOfMin(x, 0) == -1 && indexOfMin(x+2, 0) == -1);

} break; case 34: {

assert(includes(y, 10, b, 4));

} break; case 35: {

assert(includes(y+1, 9, b, 4));

} break; case 36: {

assert(!includes(y+2, 8, b, 4));

} break; case 37: {

assert(!includes(y+1, 6, b, 4));

} break; case 38: {

assert(includes(y+1, 7, b, 4));

} break; case 39: {

assert(includes(y, 10, b+2, 1));

} break; case 40: {

assert(includes(y+1, 9, b, 1));

} break; case 41: {

assert(!includes(y+2, 8, b, 1));

} break; case 42: {

assert(includes(y, 10, b, 0));

} break; case 43: {

assert(includes(y, 0, b, 0));

} break; case 44: {

assert(includes(y+1, 2, b, 2));

} break; case 45: {

assert(!includes(y+2, 2, b+1, 2));

} break; case 46: {

assert(!includes(y+2, 2, b+1, 3));

} break; case 47: {

assert(!includes(y+1, 1, b, 2));

} break; case 48: {

assert(!includes(y+1, 0, b, 1));

} break; case 49: {

assert(includes(c, 7, d+3, 3));

} break; case 50: {

assert(!includes(c, 7, d+6, 3));

} break; case 51: {

assert(includes(c, 7, d+9, 3));

} break; case 52: {

assert(countIncludes(c, 7, d, 3) == 1);

} break; case 53: {

assert(countIncludes(c, 7, d+3, 3) == 2);

} break; case 54: {

assert(countIncludes(c, 7, d+6, 3) == 0);

} break; case 55: {

assert(countIncludes(c, 7, d+9, 3) == 3);

} break; case 56: {

assert(countIncludes(c, 7, d+2, 1) == 2);

} break; case 57: {

assert(countIncludes(c, 2, d+2, 1) == 0);

} break; case 58: {

assert(countIncludes(c, 7, d, 0) == 1);

} break; case 59: {

assert(countIncludes(c, 0, d, 0) == 1);

} break; case 60: {

order(f, 16);

assert(equal(f, f+16, g));

} break; case 61: {

auto i = f[0];

order(f, 1);

assert(f[0] == i);

} break; case 62: {

auto i = f[0];

order(f, 0);

assert(f[0] == i);

} break; case 63: {

assert(pathExists(maze1, 10, 10, 8, 6, 1, 1));

} break; case 64: {

assert(!pathExists(maze2, 10, 10, 8, 6, 1, 1));

} break; case 65: {

assert(pathExists(maze3, 10, 10, 4, 3, 7, 1));

} break; case 66: {

assert(!pathExists(maze4, 10, 10, 4, 3, 7, 1));

}

}

}

int main()

{

cout << "Enter test number: ";

int n;

cin >> n;

testone(n);

cout << "Passed" << endl;

}