1)

#include <array>

#include <cstdlib>

#include <iostream>

using namespace std;

class Person {

public:

  Person() : age(0) {}

  Person(int \_age) : age(\_age) {}

  int age;

};

template <class ValueType, int N> void sort(std::array<ValueType, N>& array) {

  int i;

  int j;

  ValueType v;

  for (i = 1; i < N; i++) {

    v = array[i];

    j = i;

    while (j >= 1 && array[j - 1] > v) {

      array[j] = array[j - 1];

      j--;

    }

    array[j] = v;

  }

}

int main(int argc, char \*argv[]) {

  std::array<Person, 3> myArray = {Person(50), Person(12), Person(39)};

  sort<Person, 3>(myArray);

  for (const auto& element : myArray)

  {

     cout << element.age << endl;

  }

  return 0;

}

Q: Sort people by their age using template sort function

---------------------------------------------------------------------------------------------

2)

#include <cstdio>

struct Foo {

Foo(int i) : c{i}, a{i++}, b{++i} {}

int a, b, c;

};

int main() {

Foo f{1};

printf("%i, %i, %i \n", f.a, f.b, f.c);

}

1,3,3

Reason: Order of declaration in the structure influences the initialization list in the structure

-----------------------------------------------------------------------

3)

#include <iostream>

int f() {

std::cout << "f" << std::endl;

return 0;

}

int g() {

std::cout << "g" << std::endl;

return 1;

}

void h(const int, const int) {

std::cout << "h" << std::endl;

}

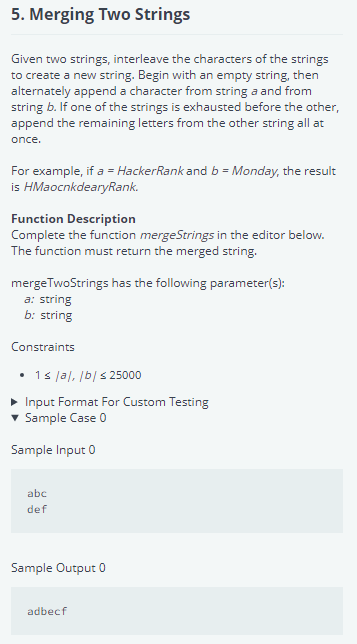
int main(int argc, char \*argv[]) {

h(f(), g());

}

Whats result? And pitpalls?

Answer: g, f, h. main function doesn’t return int type.



Answer:

#include <map>

#include <set>

#include <list>

#include <cmath>

#include <ctime>

#include <deque>

#include <queue>

#include <stack>

#include <string>

#include <bitset>

#include <cstdio>

#include <limits>

#include <vector>

#include <climits>

#include <cstring>

#include <cstdlib>

#include <fstream>

#include <numeric>

#include <sstream>

#include <iostream>

#include <algorithm>

#include <unordered\_map>

using namespace std;

int main() {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

string a, b;

cin>>a;

cin>>b;

int lena = a.length();

int lenb = b.length();

int maxl = std::max(lena, lenb);

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

if(i< lena){

std::cout<<a[i];

}

if(i<lenb){

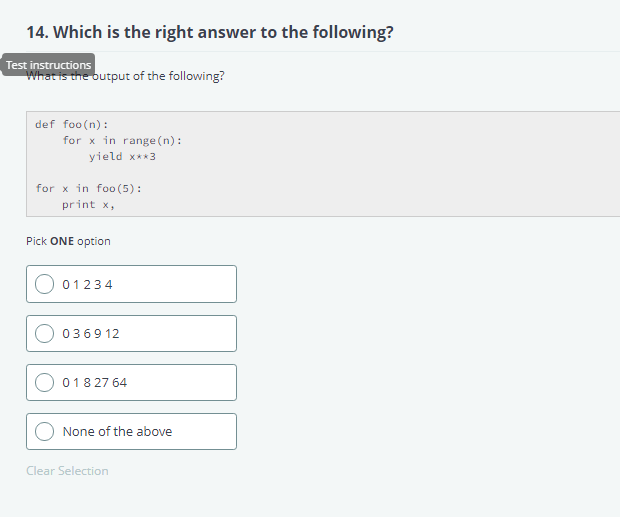
std::cout<<b[i];

}

}

return 0;

}



5) does the code compile? What is the result and explain the reason?

#include <iostream>

int main() {

double foo = 4.0;

const float& bar = foo;

foo += 2.0;

std::cout << bar << std::endl;

return 0;

}

Answer: 4 , bar is not getting updated with float reference casting towards double. When you cast with double reference, bar will be updated to 6.

7)

If code compiles fine ? if not what changes to be done.?

#include <string>

class Foo {

public:

Foo(int t) : var(t) {

}

private:

int var;

};

class Bar : public Foo {

};

int main() {

Bar d;

return 0;

}

Answer:

#include <string>

class Foo {

public:

Foo(int t) : var(t) {

}

private:

int var;

};

class Bar : public Foo {

public:

Bar(int t): Foo(t) {}

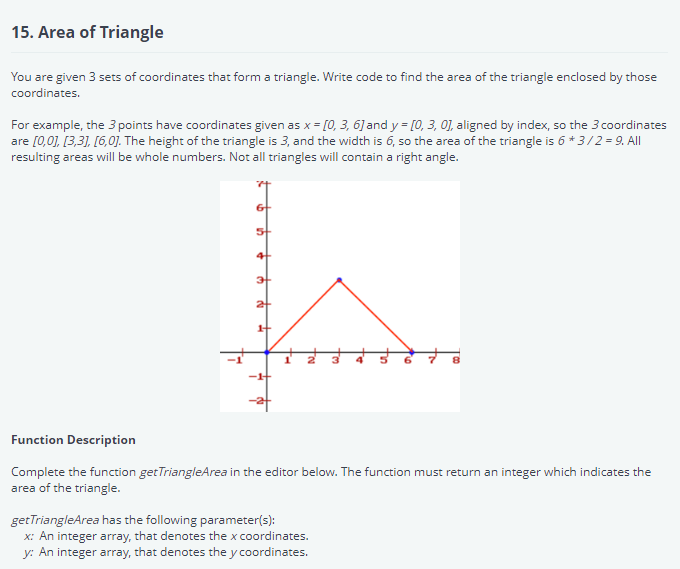
};

int main() {

Bar d(3);

return 0;

}



def getTriangleArea(x, y):

# Write your code here

l1 = sqrt((x1 - x2)\*2 + (y1 - y2)\*2)

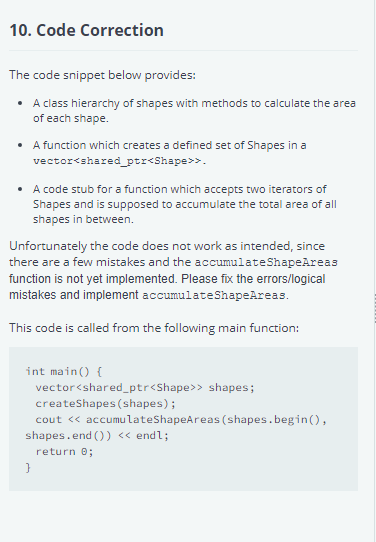
l2 = sqrt((x2 - x3)\*2 + (y2 - y3)\*2)

l3 = sqrt((x3 - x1)\*2 + (y3 - y1)\*2)

p = (l1 + l2 + l3)/2

area = sqrt(p \* (p - l1) \* (p - l2) \* (p - l3))

return int(area)



#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <vector>

#include <iostream>

#include <memory>

using namespace std;

class Shape {

protected:

  double bounding\_box\_width;

  double bounding\_box\_height;

public:

  Shape(double width, double height): bounding\_box\_width(width), bounding\_box\_height(height){}

  double getShapeArea() {

    return getBoundingBoxArea();

  }

  double getBoundingBoxArea() {

    return bounding\_box\_width \* bounding\_box\_height;

  }

};

class Circle: public Shape {

private:

  double radius;

public:

  Circle(double radius): Shape(2\*radius, 2\*radius), radius(radius) {}

  double getShapeArea(){

    return radius \* radius \* M\_PI;

  };

};

class Rectangle: public Shape {

public:

  Rectangle(double width, double height): Shape(width, height){}

};

// creates shapes and adds them to the passed vector

void createShapes(vector<shared\_ptr<Shape>> shapes) {

  shapes.push\_back( make\_shared<Circle>(5) );

  shapes.push\_back( make\_shared<Rectangle>(10, 20) );

}

// accepts two iterators and computes the sum of all areas of all shapes in the range.

template< typename IterT>

double accumulateShapeAreas( IterT start\_it, IterT end\_it) {

  //TODO

  return 0;

}

#include <limits>

#include <fstream>

int main(){

  ofstream fout(getenv("OUTPUT\_PATH"));

  u\_int testcase = 0;

  cin >> testcase;

  // if no testcase is specified use default code for the candidates testcase

  if (testcase == 0 ) {

    vector<shared\_ptr<Shape>> shapes;

    createShapes(shapes);

    fout << accumulateShapeAreas(shapes.begin(), shapes.end()) << endl;

  }

  // Testcases:

  switch( testcase ) {

    case 1: {

      // Test for correct circle area

      Circle c1(7);

      double a1 = c1.getShapeArea();

      double a2 = 7.0\*7.0\*M\_PI;

      bool result = std::abs(a1-a2) < std::numeric\_limits<double>::epsilon() \* std::abs(a1+a2);

      fout << result << endl;

      break;

    }

    case 2: {

      // Test for missing virtual

      unique\_ptr<Shape> c1( new Circle(7) );

      unique\_ptr<Circle> c2( new Circle(7) );

      bool result = c1->getShapeArea() == c2->getShapeArea();

      fout << result << endl;

      break;

    }

    case 3: {

      // Test for pass by reference

      vector<shared\_ptr<Shape>> shapes\_test;

      createShapes(shapes\_test);

      bool result = shapes\_test.size() != 0;

      fout << result << endl;

      break;

    }

    case 4: {

      // Test accumulateShapeAreas()

      Circle c0(2);

      Circle c1(12);

      Rectangle r(5,7);

      vector<shared\_ptr<Shape>> shapes\_test;

      shapes\_test.push\_back( make\_shared<Circle>(2) );

      shapes\_test.push\_back( make\_shared<Circle>(12) );

      shapes\_test.push\_back( make\_shared<Rectangle>(5, 7) );

      double total\_res = accumulateShapeAreas(shapes\_test.begin(), shapes\_test.end());

      double total\_exp = 0;

      total\_exp += c0.getShapeArea();

      total\_exp += c1.getShapeArea();

      total\_exp += r.getShapeArea();

      bool result = total\_res == total\_exp;

      fout << result << endl;

      break;

    }

    case 5: {

      // Test bounding box area of circle

      Circle c1(7);

      double bb1 = c1.getBoundingBoxArea();

      double bb2 = 7.0\*7.0\*4.0;

      bool result = std::abs(bb1-bb2) < std::numeric\_limits<double>::epsilon() \* std::abs(bb1+bb2);

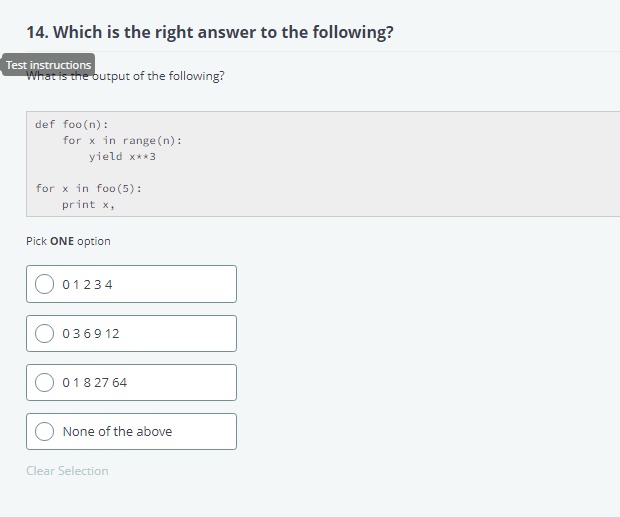
      fout << result << endl;

    }

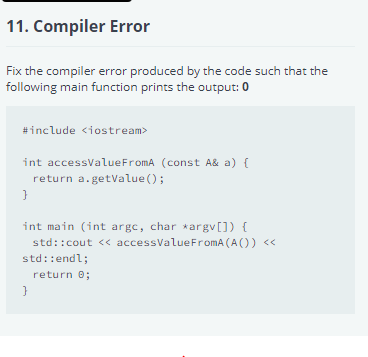
  }

  return 0;

}



Answer :C-> 0 1 8 27 64



#include <fstream>

struct A {

int getValue () {

return 0;

}

};

int accessValueFromA (const A& a) {

return a.getValue();

}

int main (int argc, char \*argv[]) {

std::ofstream fout(getenv("OUTPUT\_PATH"));

fout << accessValueFromA(A()) << std::endl;

return 0;}

answer:

#include <fstream>

struct A {

int getValue () const{

return 0;

}

};

int accessValueFromA (const A& a) {

return a.getValue();

}

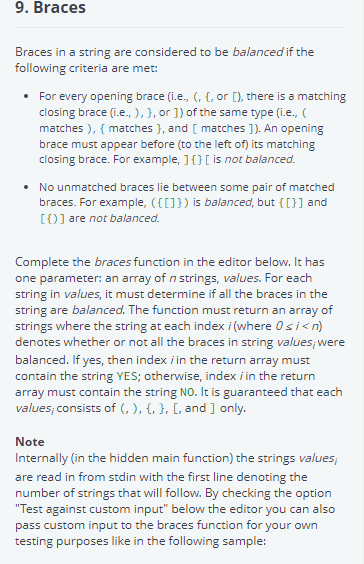
int main (int argc, char \*argv[]) {

std::ofstream fout(getenv("OUTPUT\_PATH"));

fout << accessValueFromA(A()) << std::endl;

return 0;

}



#include <bits/stdc++.h>

using namespace std;

/\*

 \* Complete the function below.

 \*/

vector < string > braces(vector < string > values) {

  //TODO

}

int main() {

    ofstream fout(getenv("OUTPUT\_PATH"));

    vector < string > res;

    int \_values\_size = 0;

    cin >> \_values\_size;

    cin.ignore (std::numeric\_limits<std::streamsize>::max(), '\n');

    vector<string> \_values;

    string \_values\_item;

    for(int \_values\_i=0; \_values\_i<\_values\_size; \_values\_i++) {

        getline(cin, \_values\_item);

        \_values.push\_back(\_values\_item);

    }

    res = braces(\_values);

    for(int res\_i=0; res\_i < res.size(); res\_i++) {

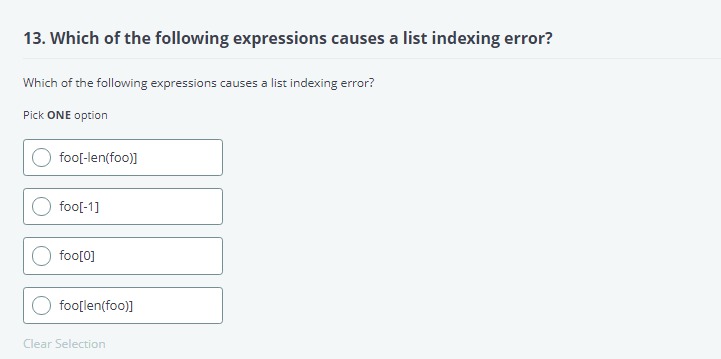
        fout << res[res\_i] << endl;;

    }

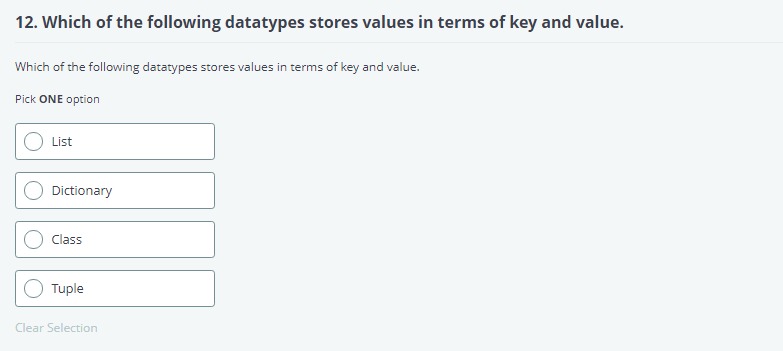
    fout.close();

    return 0;

}



Answer: d



Answer: b