CS2 PRAC Exam - circle class array

Write a statement that defines an array of five objects of the Circle class:

class Circle

{

private:

double radius;

public:

Circle()

{ radius = 0.0; }

Circle(double r)

{ radius = r; }

void setRadius(double r)

{ radius = r; }

double getRadius()

{ return radius; }

double getArea()

{ return 3.14159 \* radius \* radius; }

};

Pass the following arguments to the elements’ constructor: 12, 7, 9, 14, and 8.

|  |  |
| --- | --- |
| **Answer Key** |  |
| const int SIZE = 25;  Circle collection[SIZE] = {12, 7, 9, 14, 8 }; | |

Stack

This program uses a stack to determine whether a string entered at the keyboard has balanced parenthesis, ( ). The string is balanced when each right parenthesis occurring in the string is matched with a preceding left parenthesis. Your task is to **complete the** **code that** uses the stack as it **checks to see if a string has balanced parenthesis** (in the box or boxes). **Use the program's given constructs and variables only.**

#include <iostream>

#include <string>

#include <stack> // STL stack

using namespace std;

bool isBalanced(string, int); // Prototype

int main()

{

string str;

// Tell user what program does

cout << "This program checks a string to see "

<< "if its parentheses are properly \n"

<< "balanced.";

// Get String from user

cout << "\nType in a string with some parenthesis:\n";

getline(cin, str);

// Check the string and report

if (isBalanced(str, str.length()))

cout << "\nThe string has balanced parentheses.\n\n";

else

cout << "\nThe string does not have balanced parentheses.\n\n";

return 0;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Checks to see if a string has balanced parenthesis. \*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool isBalanced(string str, int size)

{

bool status;

stack<char> charStack;

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

{

switch(str[k])

{

case '(' :

// Put left paren on stack

charStack.push(str[k]);

break;

case ')' :

// Determine if the right parenthesis has a preceding

// left parenthesis on the stack.

if (charStack.empty())

status = false;

else

charStack.pop();

break;

}

}

if (charStack.empty())

status = true;

else

status = false;

return status;

}

U2 Arrays of objects

Assume a class Window with accessor method getWidth that accepts no parameters and returns an integer. Assume further an array of 3 Window elements named winarr, has been declared and initialized . Write a sequence of statements that prints out the width of the widest window in the array .

int maxIndex = 0;

for (int i = 1; i < 3; i++)

if (winarr[i].getWidth() > winarr[maxIndex].getWidth()) maxIndex = i;

cout << winarr[maxIndex].getWidth() << endl;

OR

if(winarr[0].getWidth() > winarr[1].getWidth())

{

if(winarr[0].getWidth() > winarr[2].getWidth())

cout << winarr[0].getWidth();

else

cout << winarr[2].getWidth();

}

else

{

if(winarr[1].getWidth() > winarr[2].getWidth())

cout << winarr[1].getWidth();

else

cout << winarr[2].getWidth();

}

OR

if(winarr[0].getWidth()>winarr[1].getWidth() &&

winarr[0].getWidth()>winarr[2].getWidth())

cout<<winarr[0].getWidth();

else if(winarr[1].getWidth()>winarr[0].getWidth() &&

winarr[1].getWidth()>winarr[2].getWidth())

cout<<winarr[1].getWidth();

else if(winarr[2].getWidth()>winarr[0].getWidth() &&

winarr[2].getWidth()>winarr[1].getWidth())

cout<<winarr[2].getWidth();

else if(winarr[0].getWidth()==winarr[0].getWidth() &&

winarr[1].getWidth()==winarr[1].getWidth())

cout<<winarr[2].getWidth();

OR

int maxIndex = 0;

for (int i = 1; i < 3; i++)

if (winarr[i].getWidth() > winarr[maxIndex].getWidth()) maxIndex = i;

cout << winarr[maxIndex].getWidth() << endl;

OR

int x = winarr[0].getWidth();

int y = winarr[1].getWidth();

int z = winarr[2].getWidth();

if (x > y && x > z) {

cout<< x;

} else if (y > z) {

cout<<y;

} else {

cout<<z;

}

OR

int x = winarr[0].getWidth();

int y = winarr[1].getWidth();

int z = winarr[2].getWidth();

if(x > y && x > z){

cout << x;

}

else if(y > x && y > z){

cout << y;

}

else{

cout << z;

}

OR

int max\_width = winarr[0].getWidth();

max\_width = max(max\_width, winarr[1].getWidth());

max\_width = max(max\_width, winarr[2].getWidth());

cout<<max\_width;

Function template palindrome

Write a function template palindrome that takes a vector parameter and returns true or false according to whether the vector does or does not read the same forward as backward (e.g., a vector containing 1, 2, 3, 2, 1 is a palindrome, but a vector containing 1, 2, 3, 4 is not).

|  |  |
| --- | --- |
| **Answer Key** |  |
| template<typename Iterator> palindrome(Iterator begin, Iterator end)  Complete solution not provided/written | |

Square class overloading

Finish the following operator overloading functions

|  |  |
| --- | --- |
| **Answer Key** |  |
| class Sqaure {  private:  double side;  public:  Square(double side = 0){  This->side = side;  }  double getSide() const{  return side;  }  double getArea() const{  return side\*side;  }  bool operator>(const Square &right){  if (side > right.side){  return true;  }else{  Return false;  }  //or  //return side>right.side?true:false;  }  bool operator==(const Square &right){  if (side == right.side){  return true;  }else{  Return false;  }  //or  //return side==right.side?true:false;  } | |

Square class template

Convert the code of following Square class to a template class so it can take int, double data type for its side.

Class Square{

private:

double side;

public:

Square(double side=0);

void setSide(double side);

double getSide();

double getArea();

};

Square::Square(double side=0){

this->side = side;

}

void Square::setSide(double side){

this->side = side;

}

double Square::getSide() {return side;}

double Square::getArea() {return side\*side;}

***// Code here…***

template <class T>

class Square{

private:

T side;

public:

Square(T side=0);

void setSide(T side);

T getSide();

T getArea();

};

template <class T>

Square<T>::Square(T side){

this->side = side;

}

template <class T>

void Square<T>::setSide(T side){

this->side = side;

}

template <class T>

T Square<T>::getSide() {return side;}

template <class T>

T Square<T>::getArea() {return side\*side;}