**Course: ENSF 480 – Principles of Software Design  
Lab #:** Lab 2  
**Instructor Name:** Mahmood Moussavi **Student Name(s):** Stephen Ravelo, Aaron Lauang **Lab Section:** B02 **Date submitted:** September 20, 2025

# Exercise A

## dictionaryList.h

*/\**

*\* File Name: dictionaryList.h*

*\* Assignment: Lab 1 Exercise A*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#ifndef DICTIONARY\_H

#define DICTIONARY\_H

#include <iostream>

#include <string>

using *namespace* std;

*// class DictionaryList: GENERAL CONCEPTS*

*//*

*//    key/datum pairs are ordered.  The first pair is the pair with*

*//    the lowest key, the second pair is the pair with the second*

*//    lowest key, and so on.  This implies that you must be able to*

*//    compare two keys with the < operator.*

*//*

*//    Each DictionaryList object has a "cursor" that is either attached*

*//    to a particular key/datum pair or is in an "off-list" state, not*

*//    attached to any key/datum pair.  If a DictionaryList is empty, the*

*//    cursor is automatically in the "off-list" state.*

*// Edit these typedefs to change the key or datum types, if necessary.*

typedef *int* Key;

typedef string Datum;

*// THE NODE TYPE*

*//    In this exercise the node type is a class, that has a ctor.*

*//    Data members of Node are private, and class DictionaryList*

*//    is declared as a friend. For details on the friend keyword refer to your* *lecture notes.*

*class* DictionaryList;

ostream *&*operator<<(ostream *&os*, DictionaryList *&rhs*);

*class* Node

{

*friend* *class* DictionaryList;

*friend* ostream *&*operator<<(ostream *&os*, DictionaryList *&rhs*);

*private:*

  Key keyM;

  Datum datumM;

  Node \*nextM;

*// This ctor should be convenient in insert and copy operations.*

  Node(*const* Key *&keyA*, *const* Datum *&datumA*, Node *\*nextA*);

};

*class* DictionaryList

{

*friend* ostream *&*operator<<(ostream *&os*, DictionaryList *&rhs*);

*public:*

  DictionaryList();

  DictionaryList(*const* DictionaryList *&source*);

  DictionaryList *&*operator=(*const* DictionaryList *&rhs*);

  string *&*operator[](*const* *int* *index*);

  ~DictionaryList();

*int* size() *const*;

*// PROMISES: Returns number of keys in the table.*

*int* cursor\_ok() *const*;

*// PROMISES:*

*//   Returns 1 if the cursor is attached to a key/datum pair,*

*//   and 0 if the cursor is in the off-list state.*

*const* Key *&*cursor\_key() *const*;

*// REQUIRES: cursor\_ok()*

*// PROMISES: Returns key of key/datum pair to which cursor is attached.*

  Datum *&*cursor\_datum() *const*;

*// REQUIRES: cursor\_ok()*

*// PROMISES: Returns datum of key/datum pair to which cursor is attached.*

*void* insert(*const* Key *&keyA*, *const* Datum *&datumA*);

*// PROMISES:*

*//   If keyA matches a key in the table, the datum for that*

*//   key is set equal to datumA.*

*//   If keyA does not match an existing key, keyA and datumM are*

*//   used to create a new key/datum pair in the table.*

*//   In either case, the cursor goes to the off-list state.*

*void* remove(*const* Key *&keyA*);

*// PROMISES:*

*//   If keyA matches a key in the table, the corresponding*

*//   key/datum pair is removed from the table.*

*//   If keyA does not match an existing key, the table is unchanged.*

*//   In either case, the cursor goes to the off-list state.*

*void* find(*const* Key *&keyA*);

*// PROMISES:*

*//   If keyA matches a key in the table, the cursor is attached*

*//   to the corresponding key/datum pair.*

*//   If keyA does not match an existing key, the cursor is put in*

*//   the off-list state.*

*void* go\_to\_first();

*// PROMISES: If size() > 0, cursor is moved to the first key/datum pair*

*//   in the table.*

*void* step\_fwd();

*// REQUIRES: cursor\_ok()*

*// PROMISES:*

*//   If cursor is at the last key/datum pair in the list, cursor*

*//   goes to the off-list state.*

*//   Otherwise the cursor moves forward from one pair to the next.*

*void* make\_empty();

*// PROMISES: size() == 0.*

*private:*

*int* sizeM;

  Node \*headM;

  Node \*cursorM;

*void* destroy();

*// Deallocate all nodes, set headM to zero.*

*void* copy(*const* DictionaryList *&source*);

*// Establishes \*this as a copy of source.  Cursor of \*this will*

*// point to the twin of whatever the source's cursor points to.*

};

#endif

## dictionaryList.cpp

*/\**

*\* File Name: dictionaryList.cpp*

*\* Assignment: Lab 1 Exercise A*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#include <assert.h>

#include <iostream>

#include <stdlib.h>

#include "dictionaryList.h"

using *namespace* std;

ostream *&*operator<<(ostream *&os*, DictionaryList *&rhs*)

{

  string s = "";

  Node\* p = *rhs*.headM;

  while (p != 0)

  {

    s += to\_string(p->keyM) + " " + p->datumM + "\n";

    p = p->nextM;

  }

  return cout << s;

}

Node::Node(*const* Key &*keyA*, *const* Datum &*datumA*, Node \**nextA*)

    : keyM(*keyA*), datumM(*datumA*), nextM(*nextA*)

{

}

DictionaryList::DictionaryList()

    : sizeM(0), headM(0), cursorM(0)

{

}

DictionaryList::DictionaryList(*const* DictionaryList &*source*)

{

  copy(*source*);

}

DictionaryList *&*DictionaryList::operator=(*const* DictionaryList *&rhs*)

{

  if (*this* != &*rhs*)

  {

    destroy();

    copy(*rhs*);

  }

  return \**this*;

}

string*&* DictionaryList::operator [](*const* *int* *index*)

{

  Node\* p = headM;

  for (*int* i = 0; i < *index*; i++)

  {

    p = p->nextM;

  }

  return p->datumM;

}

DictionaryList::~DictionaryList()

{

  destroy();

}

*int* DictionaryList::size() *const*

{

  return sizeM;

}

*int* DictionaryList::cursor\_ok() *const*

{

  return cursorM != 0;

}

*const* Key *&*DictionaryList::cursor\_key() *const*

{

  assert(cursor\_ok());

  return cursorM->keyM;

}

Datum *&*DictionaryList::cursor\_datum() *const*

{

  assert(cursor\_ok());

  return cursorM->datumM;

}

*void* DictionaryList::insert(*const* *int* *&keyA*, *const* string *&datumA*)

{

*// Add new node at head?*

  if (headM == 0 || *keyA* < headM->keyM)

  {

    headM = new Node(*keyA*, *datumA*, headM);

    sizeM++;

  }

*// Overwrite datum at head?*

  else if (*keyA* == headM->keyM)

    headM->datumM = *datumA*;

*// Have to search ...*

  else

  {

*// POINT ONE*

*// if key is found in list, just overwrite data;*

    for (Node \*p = headM; p != 0; p = p->nextM)

    {

      if (*keyA* == p->keyM)

      {

        p->datumM = *datumA*;

        return;

      }

    }

*// OK, find place to insert new node ...*

    Node \*p = headM->nextM;

    Node \*prev = headM;

    while (p != 0 && *keyA* > p->keyM)

    {

      prev = p;

      p = p->nextM;

    }

    prev->nextM = new Node(*keyA*, *datumA*, p);

    sizeM++;

  }

  cursorM = NULL;

}

*void* DictionaryList::remove(*const* *int* *&keyA*)

{

  if (headM == 0 || *keyA* < headM->keyM)

    return;

  Node \*doomed\_node = 0;

  if (*keyA* == headM->keyM)

  {

    doomed\_node = headM;

    headM = headM->nextM;

*// POINT TWO*

  }

  else

  {

    Node \*before = headM;

    Node \*maybe\_doomed = headM->nextM;

    while (maybe\_doomed != 0 && *keyA* > maybe\_doomed->keyM)

    {

      before = maybe\_doomed;

      maybe\_doomed = maybe\_doomed->nextM;

    }

    if (maybe\_doomed != 0 && maybe\_doomed->keyM == *keyA*)

    {

      doomed\_node = maybe\_doomed;

      before->nextM = maybe\_doomed->nextM;

    }

  }

  if (doomed\_node == cursorM)

    cursorM = 0;

  delete doomed\_node; *// Does nothing if doomed\_node == 0.*

  sizeM--;

}

*void* DictionaryList::go\_to\_first()

{

  cursorM = headM;

}

*void* DictionaryList::step\_fwd()

{

  assert(cursor\_ok());

  cursorM = cursorM->nextM;

}

*void* DictionaryList::make\_empty()

{

  destroy();

  sizeM = 0;

  cursorM = 0;

}

*void* DictionaryList::find(*const* Key *&keyA*)

{

  for (Node \*p = headM; p != 0; p = p->nextM)

    if (*keyA* == p->keyM)

    {

      cout << "'" << *keyA* << "' was found with datum value " << p->datumM.c\_str() << ".\n";

      cursorM = p;

      return;

    }

  cout << "'" << *keyA* << "' was not found.\n";

  cursorM = 0;

}

*void* DictionaryList::destroy()

{

  Node \*p = headM;

  Node \*prev;

  while (p != 0)

  {

    prev = p;

    p = p->nextM;

    delete prev;

  }

  headM = 0;

  sizeM = 0;

}

*void* DictionaryList::copy(*const* DictionaryList *&source*)

{

  if (*source*.headM == 0)

  {

    headM = 0;

    return;

  }

  headM = new Node(*source*.headM->keyM, *source*.headM->datumM, NULL);

  Node \*newest\_node = headM;

*const* Node \*source\_node = *source*.headM;

  if (source\_node == *source*.cursorM)

    cursorM = newest\_node;

  while (true)

  {

    source\_node = source\_node->nextM;

    if (source\_node == 0)

      break;

    newest\_node->nextM = new Node(source\_node->keyM, source\_node->datumM, NULL);

    if (source\_node == *source*.cursorM)

      cursorM = newest\_node->nextM;

    newest\_node = newest\_node->nextM;

  }

  sizeM = *source*.sizeM;

}

## Output

A screenshot of a computer program

AI-generated content may be incorrect.

# Exercise B

## point.h

*/\**

*\* File Name: point.h*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#ifndef POINT\_H

#define POINT\_H

*class* Point

{

*public:*

    Point(*double* *x*, *double* *y*);

*double* getX() *const*;

*void* setX(*const* *double* *x*);

*double* getY() *const*;

*void* setY(*const* *double* *y*);

*int* getID() *const*;

*void* display() *const*;

*static* *int* counter();

*double* distance (Point *&other*);

*static* *double* distance(Point *&p1*,Point *&p2*);

*private:*

*double* xM;

*double* yM;

*const* *int* ID;

*static* *int* count;

};

#endif

## point.cpp

*/\**

*\* File Name: point.cpp*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#include <iostream>

#include <iomanip>

#include <math.h>

using *namespace* std;

#include "point.h"

*int* Point::count = 0;

Point::Point(*double* *x*, *double* *y*) : xM(*x*), yM(*y*), ID((++count) + 1000) {}

*double* Point::getX() *const* { return xM; }

*double* Point::getY() *const* { return yM; }

*int* Point::getID() *const* { return ID; }

*int* Point::counter() { return count; }

*void* Point::setX(*const* *double* *x*) { xM = *x*; }

*void* Point::setY(*const* *double* *y*) { yM = *y*; }

*void* Point::display() *const*

{

    cout << fixed;

    cout << setprecision(2);

    cout << xM << endl;

    cout << yM << endl;

}

*double* Point::distance(Point *&other*) { return sqrt(pow(*other*.xM - xM, 2) + pow(*other*.yM - yM, 2)); }

*double* Point::distance(Point *&p1*, Point *&p2*) { return sqrt(pow(*p2*.xM - *p1*.xM, 2) + pow(*p2*.yM - *p1*.yM, 2)); }

## shape.h

*/\**

*\* File Name: shape.h*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#ifndef SHAPE\_H

#define SHAPE\_H

#include "point.h"

*class* Shape

{

*public:*

    Shape(*double* *x*, *double* *y*, *const* *char* *\*shapeName*);

    Shape(*const* Shape *&source*);

    Shape *&*operator=(*const* Shape *&rhs*);

    ~Shape();

*const* Point *&*getOrigin() *const*;

*char* *\**getName() *const*;

*virtual* *void* display() *const*;

*double* distance(Shape *&other*);

*static* *double* distance(Shape *&s1*, Shape *&s2*);

*void* move(*double* *dx*, *double* *dy*);

*private:*

    Point originM;

*char* \*shapeNameM;

};

#endif

## shape.cpp

*/\**

*\* File Name: shape.cpp*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#include <iostream>

#include <iomanip>

#include <math.h>

#include <cstring>

using *namespace* std;

#include "shape.h"

Shape::Shape(*double* *x*, *double* *y*, *const* *char* \**shapeName*)

    : originM(Point(*x*, *y*)), shapeNameM(new *char*[strlen(*shapeName*) + 1])

{

    strcpy(shapeNameM, *shapeName*);

}

Shape::Shape(*const* Shape &*source*)

    : originM(Point(*source*.getOrigin().getX(), *source*.getOrigin().getY())), shapeNameM(new *char*[strlen(*source*.shapeNameM) + 1])

{

    strcpy(shapeNameM, *source*.shapeNameM);

}

Shape *&*Shape::operator=(*const* Shape *&rhs*)

{

    if (*this* != &*rhs*)

    {

        delete[] shapeNameM;

        shapeNameM = new *char*[strlen(*rhs*.shapeNameM) + 1];

        originM.setX(*rhs*.originM.getX());

        originM.setY(*rhs*.originM.getY());

    }

    return \**this*;

}

Shape::~Shape() { delete[] shapeNameM; }

*const* Point *&*Shape::getOrigin() *const* { return originM; }

*char* *\**Shape::getName() *const* { return shapeNameM; }

*void* Shape::display() *const*

{

    cout << fixed << setprecision(2);

    cout << "Shape Name: " << shapeNameM << endl

         << "X-coordinate: " << originM.getX() << endl

         << "Y-coordinate: " << originM.getY() << endl;

}

*double* Shape::distance(Shape *&other*)

{

    return sqrt(pow(*other*.getOrigin().getX() - originM.getX(), 2) + pow(*other*.getOrigin().getY() - originM.getY(), 2));

}

*double* Shape::distance(Shape *&s1*, Shape *&s2*)

{

    return sqrt(pow(*s2*.getOrigin().getX() - *s1*.getOrigin().getX(), 2) + pow(*s2*.getOrigin().getY() - *s1*.getOrigin().getY(), 2));

}

*void* Shape::move(*double* *dx*, *double* *dy*)

{

    originM.setX(originM.getX() + *dx*);

    originM.setY(originM.getY() + *dy*);

}

## square.h

*/\**

*\* File Name: square.h*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#ifndef SQUARE\_H

#define SQUARE\_H

#include "shape.h"

*class* Square : *public* Shape

{

*public:*

    Square(*double* *x*, *double* *y*, *double* *side\_a*, *const* *char* *\*shapeName*);

    Square(*const* Square *&source*);

    Square *&*operator=(*const* Square *&rhs*);

*double* get\_side\_a() *const*;

*void* set\_side\_a(*const* *double* *side\_a*);

*virtual* *void* display() *const*;

*double* area() *const*;

*double* perimeter() *const*;

*private:*

*double* side\_aM;

};

#endif

## square.cpp

*/\**

*\* File Name: square.cpp*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#include <iostream>

#include <iomanip>

#include <math.h>

#include <cstring>

using *namespace* std;

#include "square.h"

Square::Square(*double* *x*, *double* *y*, *double* *side\_a*, *const* *char* \**shapeName*)

    : Shape(*x*, *y*, *shapeName*), side\_aM(*side\_a*)

{

}

Square::Square(*const* Square &*source*)

    : Shape(*source*.getOrigin().getX(), *source*.getOrigin().getY(), *source*.getName()), side\_aM(*source*.side\_aM)

{

}

Square *&*Square::operator=(*const* Square *&rhs*)

{

    if (*this* != &*rhs*)

    {

        Shape::operator=(*rhs*);

        set\_side\_a(*rhs*.side\_aM);

    }

    return \**this*;

}

*double* Square::get\_side\_a() *const* { return side\_aM; }

*void* Square::set\_side\_a(*double* *side\_a*) { side\_aM = *side\_a*; }

*double* Square::area() *const* { return side\_aM \* side\_aM; }

*double* Square::perimeter() *const* { return side\_aM \* 4; }

*void* Square::display() *const*

{

    cout << fixed << setprecision(2);

    cout << "Shape Name: " << getName() << endl

         << "X-coordinate: " << getOrigin().getX() << endl

         << "Y-coordinate: " << getOrigin().getY() << endl

         << "Side a: " << side\_aM << endl

         << "Area: " << area() << endl

         << "Perimeter: " << perimeter() << endl;

}

## rectangle.h

*/\**

*\* File Name: rectangle.h*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include "square.h"

*class* Rectangle : *public* Square

{

*public:*

    Rectangle(*double* *x*, *double* *y*, *double* *side\_a*, *double* *side\_b*, *const* *char* *\*shapeName*);\

    Rectangle(*const* Rectangle *&source*);

    Rectangle *&*operator=(*const* Rectangle *&rhs*);

*double* get\_side\_b() *const*;

*void* set\_side\_b(*const* *double* *side\_b*);

*void* display() *const*;

*double* area() *const*;

*double* perimeter() *const*;

*private:*

*double* side\_bM;

};

#endif

## rectangle.cpp

*/\**

*\* File Name: rectangle.cpp*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#include <iostream>

#include <iomanip>

#include <math.h>

#include <cstring>

using *namespace* std;

#include "rectangle.h"

Rectangle::Rectangle(*double* *x*, *double* *y*, *double* *side\_a*, *double* *side\_b*, *const* *char* \**shapeName*)

    : Square(*x*, *y*, *side\_a*, *shapeName*), side\_bM(*side\_b*)

{

}

Rectangle::Rectangle(*const* Rectangle &*source*)

    : Square(*source*.getOrigin().getX(), *source*.getOrigin().getY(), *source*.get\_side\_a(), *source*.getName()), side\_bM(*source*.side\_bM)

{

}

Rectangle *&*Rectangle::operator=(*const* Rectangle *&rhs*)

{

    if (*this* != &*rhs*)

    {

        Square::operator=(*rhs*);

        set\_side\_b(*rhs*.side\_bM);

    }

    return \**this*;

}

*double* Rectangle::get\_side\_b() *const* { return side\_bM; }

*void* Rectangle::set\_side\_b(*double* *side\_b*) { side\_bM = *side\_b*; }

*double* Rectangle::area() *const* { return get\_side\_a() \* side\_bM; }

*double* Rectangle::perimeter() *const* { return (get\_side\_a() \* 2) + (side\_bM \* 2); }

*void* Rectangle::display() *const*

{

    cout << fixed << setprecision(2);

    cout << "Shape Name: " << getName() << endl

         << "X-coordinate: " << getOrigin().getX() << endl

         << "Y-coordinate: " << getOrigin().getY() << endl

         << "Side a: " << get\_side\_a() << endl

         << "Side b: " << side\_bM << endl

         << "Area: " << area() << endl

         << "Perimeter: " << perimeter() << endl;

}

## graphicsWorld.h

*/\**

*\* File Name: graphicsWorld.h*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#ifndef GRAPHICS\_WORLD\_H

#define GRAPHICS\_WORLD\_H

#include "rectangle.h"

*class* GraphicsWorld

{

*public:*

*static* *void* run();

};

#endif

## graphicsWorld.cpp

*/\**

*\* File Name: graphicsWorld.cpp*

*\* Assignment: Lab 1 Exercise B*

*\* Lab Section: B02*

*\* Completed by: Stephen Ravelo, Aaron Lauang*

*\* Submission Date: September 20, 2025*

*\*/*

#include <iostream>

using *namespace* std;

#include "graphicsWorld.h"

*void* GraphicsWorld::run()

{

#if 1 *// Change 0 to 1 to test Point*

    Point m(6, 8);

    Point n(6, 8);

    n.setX(9);

    cout << "\nExpected to dispaly the distance between m and n is: 3";

    cout << "\nThe distance between m and n is: " << m.distance(n);

    cout << "\nExpected second version of the distance function also print: 3";

    cout << "\nThe distance between m and n is again: "

         << Point::distance(m, n);

#endif *// end of block to test Point*

#if 1 *// Change 0 to 1 to test Square*

    cout << "\n\nTesting Functions in class Square:" << endl;

    Square s(5, 7, 12, "SQUARE - S");

    s.display();

#endif *// end of block to test Square*

#if 1 *// Change 0 to 1 to test Rectangle*

    cout << "\nTesting Functions in class Rectangle:" << endl;

    Rectangle a(5, 7, 12, 15, "RECTANGLE A");

    a.display();

    Rectangle b(16, 7, 8, 9, "RECTANGLE B");

    b.display();

*double* d = a.distance(b);

    cout << "\nDistance between square a, and b is: " << d << endl;

    Rectangle rec1 = a;

    rec1.display();

    cout << "\nTesting assignment operator in class Rectangle:" << endl;

    Rectangle rec2(3, 4, 11, 7, "RECTANGLE rec2");

    rec2.display();

    rec2 = a;

    a.set\_side\_b(200);

    a.set\_side\_a(100);

    cout << "\nExpected to display the following values for objec rec2: " << endl;

    cout << "Rectangle Name: RECTANGLE A\n"

         << "X-coordinate: 5\n"

         << "Y-coordinate: 7\n"

         << "Side a: 12\n"

         << "Side b: 15\n"

         << "Area: 180\n"

         << "Perimeter: 54\n";

    cout << "\nIf it doesn't there is a problem with your assignment operator.\n"

         << endl;

    rec2.display();

    cout << "\nTesting copy constructor in class Rectangle:" << endl;

    Rectangle rec3(a);

    rec3.display();

    a.set\_side\_b(300);

    a.set\_side\_a(400);

    cout << "\nExpected to display the following values for objec rec2: " << endl;

    cout << "Rectangle Name: RECTANGLE A\n"

         << "X-coordinate: 5\n"

         << "Y-coordinate: 7\n"

         << "Side a: 100\n"

         << "Side b: 200\n"

         << "Area: 20000\n"

         << "Perimeter: 600\n";

    cout << "\nIf it doesn't there is a problem with your assignment operator.\n"

         << endl;

    rec3.display();

#endif *// end of block to test Rectangle*

#if 1 *// Change 0 to 1 to test using array of pointer and polymorphism*

    cout << "\nTesting array of pointers and polymorphism:" << endl;

    Shape \*sh[4];

    sh[0] = &s;

    sh[1] = &b;

    sh[2] = &rec1;

    sh[3] = &rec3;

    sh[0]->display();

    sh[1]->display();

    sh[2]->display();

    sh[3]->display();

#endif *// end of block to test array of pointer and polymorphism*

}

*int* main(*void*)

{

    GraphicsWorld::run();

    return 0;

}

## Output

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.