Programming 2 - Assignment 3

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

May 11, 2022

1 Testing the Program

1.1 Testing Code - Main.cpp

1.2 testing - output

2 Problems

This section will briefly discuss the Problems that have occurred during programming.

2.1 virtual functions/Data types

Despite the best efforts, there is still a problem with the code. Specifically concerning the *Picture* class. While drawing a picture, the outputted polygons of type *RegularPolygon* do not contain a center point. In the following two sections, the code which was used to narrow the problem down and its output are presented. The code contains multiple *cout*s to determine where exactly the problem lies. We can see, that while adding polygons to the *LinkedList* of the picture, that holds pointer to polygons, the correct *clone()* function of the respective derived class is called. But when calling the *draw()* function, it always calls the virtual function defined in the *Polygon* class.

This Problem is probably caused by the type that those polygons hold during runtime of the program. Because the *LinkedListPointer* stores just pointer to polygons (*Polygon**) this could be caused by the declaration of the type of pointer inside there.

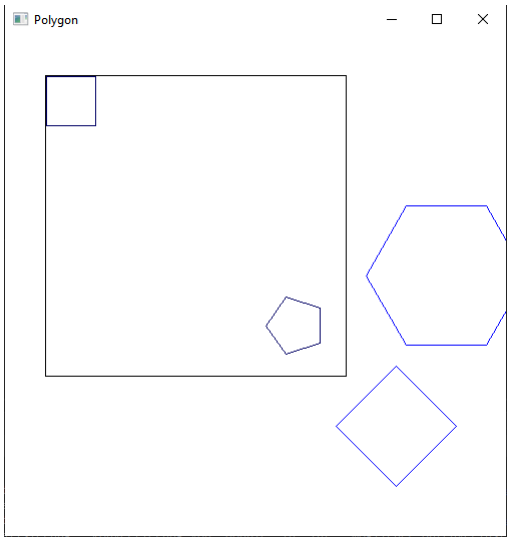
The testing of clone and draw, that we have already done suggests, that these functions are created correctly.

2.1.1 testing code

```
1  //*************************************************************
2  // "Main.cpp"
3  //
4  // is used to test the functions of the "Polygon" class and its derived
5  // classes. Further it also contains functions to test the class "Picture"
6  //
7  // created by Felix Dressler, 11.05.2022
8  //*************************************************************
9
10 #include"Polygon.h"
11 #include"Drawing.h"
12 #include"Picture.h"
13
14 #include<iostream>
15
16 using namespace std;
17
18 using namespace compsys;
19
20
21 int main() {
22   Polygon p(100);
23
24   beginDrawing(500, 500, "Polygon", 0xFFFFFF, false);
```

```
25
26    p.add(1, 1);
27    p.add(50, 1);
28    p.add(50, 50);
29    p.add(1, 50);
30
31    RegularPolygon q(250, 250, 30, 5, 0, 100);
32    Square r(350, 350, 60, 0, 0xFF);
33    Hexagon s(400,200,80,0,0x00FF);
34
35    Picture pic;
36
37    cout << "add p - Polygon" << endl;
38    pic.add(p);
39    cout << "add q - Regular" << endl;
40    pic.add(q);
41    cout << "add r - Regular" << endl;
42    pic.add(r);
43    cout << "add s - Regular" << endl;
44    pic.add(s);
45
46    pic.draw(40,40,300,300,1);
47
48    endDrawing();
49  }
```

2.1.2 output of the testing code




```
add p - Polygon
clone - Polygon
add q - Regular
clone - Regular Polygon
add r - Regular
clone - Regular Polygon
add s - Regular
clone - Regular Polygon
draw - Polygon
draw - Polygon
draw - Polygon
```

```
        draw - Polygon
        draw - Polygon
```

3 The Program

3.1 The Program - Polygon

3.1.1 Polygon.h

```
1  //*****************************************************************
2  // "Polygon.h"
3  //
4  // contains the classes "Plygon", "RegularPolygon", "Square" and
5  // "Hexagon", which all create polygons. The names of the classes
6  // are already self explanatory.
7  //
8  // created by Felix Dressler, 10.05.2022
9  //*****************************************************************
10
11 #pragma once
12
13 #include"LinkedList.h"
14
15 //*****************************************************************
16 // "Polygon.h"
17 //
18 // holds a linked list of points of a Polygon and the color of the
19 // outline of the polygon.
20 //*****************************************************************
21 class Polygon
22 {
23 private:
24     unsigned int color = 0xFFFF;
25     LinkedListArr points;
26
27 public:
28     // create polygon in denoted color (default black)
29     Polygon(unsigned int color = 0);
30
31     // copy constructor, copy assignment operator, destructor
32     Polygon(Polygon& p);
33     Polygon& operator=(Polygon& p);
34     virtual ~Polygon();
35
36     // create a heap-allocated copy of this polygon
37     virtual Polygon* clone();
38
39     // add point with relative coordinates (x,y) to polygon
40     void add(double x, double y);
41
42     // draws the polygon at absolute coordinates (x0,y0) scaled by factor f;
43     // thus every point (x,y) is drawn at position (x0+x*f, y0+y*f)
44     void draw(double x0 = 0, double y0 = 0, double f = 1);
45
46     //sets the color of the polygon
47     void setColor(unsigned int c){
48         this->color = c;
49     }
50
51     //gives back the color of the polygon
52     unsigned int getColor() {
```

```
53          return color;
54      }
55
56      //gives back the number of points in the polygon
57      int pNum() {
58          return points.length();
59      }
60
61      //gives back the specific coordinate x (n=0) or y (n=1) for a point of Polygon
62      int getCoord(int i, int n) {
63          return points.get(i, n);
64      }
65  };
66
67
68  //****************************************************************
69  // "Polygon.h"
70  //
71  // constructs a special case of a polygon, a regular polygon.
72  // It additionally holds an angle a, x and y middle-point coordinates
73  // and a radius.
74  //****************************************************************
75  class RegularPolygon : public Polygon
76  {
77  private:
78      double a;
79      double x;
80      double y;
81      double r;
82  public:
83      //constructor
84      RegularPolygon(double x, double y, double r, int n,
85          double a = 0, unsigned int c = 0);
86
87      //destructor
88      ~RegularPolygon();
89
90      //draws a regular polygon
91      void draw(double x0 = 0, double y0 = 0, double f = 1);
92
93      //clones a regular polygon
94      RegularPolygon* clone();
95  };
96
97
98  //****************************************************************
99  // "Polygon.h" and "Hexagon"
100 //
101 // constructs two special cases of a regular polygon by fixing
102 // the number of points to 4/6.
103 //****************************************************************
104 class Square : public RegularPolygon
105 {
106 public:
107     //constructs a square
108     Square(double x, double y, double r,
109         double a = 0, unsigned int c = 0);
110 };
111
112 class Hexagon : public RegularPolygon
```

```
{
public:
    //constructs a hexagon
    Hexagon(double x, double y, double r,
        double a = 0, unsigned int c = 0);
};
```

3.1.2 Polygon.cpp

```
//*****************************************************************
// "Polygon.cpp"
//
// contains the methods of the class "Polygon", "RegularPolygon",
// "Square" and "Hexagon".
//
// created by Felix Dressler,
//*****************************************************************

#include"Polygon.h"
#include"LinkedList.h"
#include"Drawing.h"

#define _USE_MATH_DEFINES

#include<cmath>
#include<iostream>

using namespace std;
using namespace compsys;

//constructor for Polygon
Polygon::Polygon(unsigned int color) {
    this->color = color;
    this->points;
}

//copy constructor for polygon
Polygon::Polygon(Polygon& p) {
    this->color = p.color;
    int length = p.points.length();

    for (int i = 0; i < length; i++) {
        this->add(p.points.get(i, 0), p.points.get(i, 1));
    }
}

//copy assignment operator for Polygon
Polygon& Polygon::operator=(Polygon& p) {
    this->color = p.color;
    int length = p.points.length();

    for(int i = 0; i < length; i++) {
        this->add(p.points.get(i, 0), p.points.get(i, 1));
    }


    return *this;
}
```

```
50
51  //destructor for Polygon (in this case there is nothing to be deallocated)
52  Polygon::~Polygon() {
53
54  }
55
56  //*****************************************************************
57  // "clone()"
58  //
59  // virtual function clone, that creates a heap allocated clone
60  // of a polygon.
61  //*****************************************************************
62  Polygon* Polygon::clone() {
63      //cout << "clone - Polygon" << endl;
64      Polygon* p = new Polygon(this->color);
65
66      int length = this->points.length();
67
68      for (int i = 0; i < length; i++) {
69          p->add(this->points.get(i,0), this->points.get(i, 1));
70      }
71
72      return p;
73  }
74
75  //*****************************************************************
76  // "add"
77  //
78  // is a method of the "Polygon" class. It adds a point to a polygon.
79  //*****************************************************************
80  void Polygon::add(double x, double y) {
81      int e[2] = { x,y };
82      points.insert(e);
83  }
84
85  //*****************************************************************
86  // "draw()"
87  //
88  // is a method of Polygon. It draws a Polygon with the left upper
89  // point of the surrounding square at (x0,y0) by using the drawPolygon()
90  // function defined in "Drawing.cpp".
91  //*****************************************************************
92  void Polygon::draw(double x0, double y0, double f) {
93      //cout << "draw - Polygon" << endl;
94      int length = points.length();
95      int* tempx = new int[length];
96      int* tempy = new int[length];
97
98      for (int i = 0; i < length; i++) {
99          tempx[i] = x0 + points.get(i,0)*f;
100         tempy[i] = y0 + points.get(i, 1)*f;
101     }
102
103     drawPolygon(length, tempx, tempy, color);
104
105     delete[] tempx;
106     delete[] tempy;
107 }
108
```

```
109 //
        ********************************************************************************************************
110
111 //***************************************************************
112 // constructor of "RegularPolygon"
113 //
114 // constructs a regular polygon with center point (x,y), radius from
115 // the center to the points r, number of points n, angle of the
116 // first point to the horizontal axis a and color c.
117 //***************************************************************
118 RegularPolygon::RegularPolygon(double x, double y,
119     double r, int n, double a, unsigned int c):Polygon(c) {
120
121     for (double i = 0; i < n; i ++) {
122         this->add(x - (cos(a + i * 2 * M_PI / n) * r), y - (sin(a + i * 2 * M_PI / n) * r
            ));
123     }
124     this->x = x;
125     this->y = y;
126     this->a = a;
127     this->r = r;
128     setColor(c);
129 }
130
131
132 //***************************************************************
133 // "draw"
134 //
135 // the "draw" function of Polygon is a virtual function, this is the
136 // respective function for "RegularPolygon". It first calls the normal
137 // "Polygon::draw()" and then additionally draws the center point
138 // of the regular polygon.
139 //***************************************************************
140 void RegularPolygon::draw(double x0, double y0, double f) {
141     //cout << "draw - Regular Polygon" << endl;
142     Polygon::draw(x0,y0,f);
143     drawPoint(x0 + x, y0 + y, getColor());
144 }
145
146 //destructor for Polygon (in this case there is nothing to be deallocated)
147 RegularPolygon::~RegularPolygon() {
148
149 }
150
151
152 //***************************************************************
153 // "clone()"
154 //
155 // the "clone" function of Polygon is a virtual function, this is the
156 // respective function for "RegularPolygon". It constructs a new
157 // regular polygon with the same values.
158 //***************************************************************
159 RegularPolygon* RegularPolygon::clone() {
160     //cout << "clone - Regular Polygon" << endl;
161     RegularPolygon* p = new RegularPolygon(*this);
162
163     return p;
164 }
165
```

```
166 //
        ********************************************************************************************
167
168 //constructor for Square - constructs a regular polygon with a fixed number of points (4)
169 Square::Square(double x, double y,
170     double r, double a, unsigned int c):RegularPolygon(x,y,r,4,a,c) {
171
172 }
173
174 //
        ********************************************************************************************
175
176 //constructor for Square - constructs a regular polygon with a fixed number of points (6)
177 Hexagon::Hexagon(double x, double y,
178     double r, double a, unsigned int c) :RegularPolygon(x, y, r, 6, a, c) {
179
180 }
```

3.2 The Program - Picture

3.2.1 Picture.h

```
1  //*************************************************************
2  // "Picture.h"
3  //
4  // contains the definition for the class "Picture", which holds polygons
5  // in a linked list and is able to draw those in combination with a frame.
6  //
7  // created by Felix Dresser, 11.05.2022
8  //*************************************************************
9
10 #pragma once
11
12 #include"LinkedListPointer.h"
13 #include"Polygon.h"
14
15 class Picture
16 {
17 private:
18     LinkedListPointer Polygons;
19     unsigned int h;
20     unsigned int w;
21 public:
22     Picture();
23     Picture(Picture& p);
24     Picture& operator=(Picture& p);
25     ~Picture();
26     void add(Polygon& p);
27     void draw(double x, double y, double w, double h, double f = 1.0);
28 };
```

3.2.2 Picture.cpp

```
1  //*************************************************************
2  // "Picture.cpp"
3  //
4  // contains the methods of the class "Picture".
5  //
6  // created by Felix Dressler,
7  //*************************************************************
8
9  #include"Polygon.h"
10 #include"LinkedListPointer.h"
11 #include"Drawing.h"
12 #include"Picture.h"
13
14 #define _USE_MATH_DEFINES
15
16 #include<iostream>
17 #include<cmath>
18
19 using namespace std;
20
21 //constructor for Picture
22 Picture::Picture() {
23     h = 200;
24     w = 200;
25 }
26
27 //copy constructor for Picture
28 Picture::Picture(Picture& p) {
29     this->h = p.h;
30     this->w = p.w;
31
32     int length = p.Polygons.length();
33     for (int i = 0; i < length; i++) {
34         this->Polygons.insert(p.Polygons.get(i));
35     }
36
37 }
38
39 //copy assignment operator for Picture
40 Picture& Picture::operator=(Picture& p) {
41     this->h = p.h;
42     this->w = p.w;
43
44     int length = p.Polygons.length();
45     for (int i = 0; i < length; i++) {
46         this->Polygons.insert(p.Polygons.get(i));
47     }
48
49     return *this;
50 }
51
52 //destructor for Picture
53 // destructs all heap allocated clones that were created and
54 // added into the Linked List that Picture holds.
55 Picture::~Picture() {
56     int length = Polygons.length();
57
58     for (int i = 0; i < length; i++) {
59         delete Polygons.get(i);
```

```
60      }
61  }
62
63  //*****************************************************************
64  // "add()"
65  //
66  // adds a Polygon to the linked list of picture by cloning the
67  // respective polygon. This creates a heap allocated copy of the
68  // polygon, which has to be deleted by the destructor of Picture.
69  //*****************************************************************
70  void Picture::add(Polygon &p) {
71      this->Polygons.insert(p.clone());
72  }
73
74  //*****************************************************************
75  // "draw()"
76  //
77  // first draws a frame of width w and heigth h at position (x,y) for
78  // its upper left corner.
79  // It then draws the polygons that are held in the linked list at
80  // their respective position to (x,y) and resized by a factor f.
81  // (clipping allowed)
82  //*****************************************************************
83  void Picture::draw(double x, double y, double w, double h, double f) {
84      Polygon frame(0);
85      frame.add(0, 0);
86      frame.add(0, h);
87      frame.add(w, h);
88      frame.add(w, 0);
89      frame.draw(x,y,f);
90
91      int length = Polygons.length();
92
93      for (int i = 0; i < length; i++) {
94          Polygons.get(i)->draw(x,y,f);
95      }
96  }
```

3.3 The Program - Linked Lists

3.3.1 LinkedListArr.h

```
1   //*****************************************************************
2   // "LinkedList.h"
3   //
4   // contains the class "LinkedList", that implements a linked list
5   // based on the linked list that was presented in the lecture slides.
6   //
7   // created by Felix Dressler,
8   //*****************************************************************
9
10  #pragma once
11
12  class LinkedListArr
13  {
14    class Node;
15  private:
```

```
16   Node* head;
17   int number; //starts with 0 for the first element
18 public:
19   LinkedListArr();
20   ~LinkedListArr();
21
22   int length() const;
23   LinkedListArr& insert(int* e);
24   int get(int i, int n) const;
25 };
```

3.3.2 LinkedListArr.cpp

```
1  //*****************************************************************
2  // "LinkedList.h"
3  //
4  // contains the methods of the class "LinkedList" based on the
5  // functionality as presented in the lecture slides.
6  //
7  // created by Felix Dressler,
8  //*****************************************************************
9
10 #include "LinkedList.h"
11
12
13 class LinkedListArr::Node {
14   friend class LinkedListArr;
15 private:
16   int* value; Node* next;
17   Node(int* v, Node* n) {
18     next = n;
19     value = new int[2];
20     value[0] = v[0];
21     value[1] = v[1];
22   }
23 };
24
25 LinkedListArr::LinkedListArr() {
26   head = 0;
27   number = 0;
28 }
29
30 LinkedListArr::~LinkedListArr() {
31   Node* node = head;
32   while (node != 0) {
33     Node* node0 = node->next;
34     delete node;
35     node = node0;
36   }
37 }
38
39 int LinkedListArr::length() const {
40   return number;
41 }
42 LinkedListArr& LinkedListArr::insert(int* e) {
43   Node* node = new Node(e, head);
44   head = node;
45   number = number + 1;
```

```
46   return *this;
47 }
48 int LinkedListArr::get(int i, int n) const {
49   Node* node = head;
50   for (int j = 0; j < number - i - 1; j++)
51     node = node->next;
52   return node->value[n];
53 }
```

3.3.3 LinkedListPointer.h

```
1  //****************************************************************
2  // "LinkedListPointer.h"
3  //
4  // contains the class "LinkedListPointer", that implements a linked list
5  // based on the linked list that was presented in the lecture slides.
6  //
7  // created by Felix Dressler,
8  //****************************************************************
9
10 #pragma once
11
12 #include"Polygon.h"
13
14 class LinkedListPointer
15 {
16     class PointerNode;
17 private:
18     PointerNode* head;
19     int number; //starts with 0 for the first element
20 public:
21     LinkedListPointer();
22     ~LinkedListPointer();
23
24     int length() const;
25     LinkedListPointer& insert(Polygon* e);
26     Polygon* get(int i) const;
27 };
```

3.3.4 LinkedListPointer.cpp

```
1  //****************************************************************
2  // "LinkedListPointer.h"
3  //
4  // contains the methods of the class "LinkedListPointer" based on the
5  // functionality as presented in the lecture slides.
6  //
7  // created by Felix Dressler, 11.05.2022
8  //****************************************************************
9
10 #include"LinkedListPointer.h"
11 #include"Polygon.h"
12
13 #include<iostream>
14
15 using namespace std;
```

```
16
17 class LinkedListPointer::PointerNode {
18     friend class LinkedListPointer;
19 private:
20     Polygon* value; PointerNode* next;
21     PointerNode(Polygon* v, PointerNode* n) {
22         next = n;
23         value = v;
24     }
25 };
26
27 LinkedListPointer::LinkedListPointer() {
28     head = 0;
29     number = 0;
30 }
31
32 LinkedListPointer::~LinkedListPointer() {
33     PointerNode* node = head;
34     while (node != 0) {
35         PointerNode* node0 = node->next;
36         delete node;
37         node = node0;
38     }
39 }
40
41 int LinkedListPointer::length() const {
42     return number;
43 }
44 LinkedListPointer& LinkedListPointer::insert(Polygon* e) {
45     PointerNode* node = new PointerNode(e, head);
46     head = node;
47     number = number + 1;
48     return *this;
49 }
50 Polygon* LinkedListPointer::get(int i) const {
51     PointerNode* node = head;
52     for (int j = 0; j < number - i - 1; j++)
53         node = node->next;
54     return node->value;
55 }
```