```
Script started on 2024-05-09 18:07:29-05:00 [TERM="xterm" TTY="/dev/pts/0" COLUMNS=
ee43254@ares:~$ pwd
/home/students/ee43254
ee43254@ares:~$ cat point.info
Name: Kyle Enkhzul

Class: CSC122-W01



Activity: Operate on this!

Level: 3, 2 (base program), 1 (overload operator [])



Description:



The program defines a Point class representing a 2D point with x and y

coordinates and provides methods for various operations such as input,

output, distance calculation, flipping, and shifting. Additionally, it

overloads operators for input, output, distance calculation, equality,

inequality, and midpoint calculation.

ee43254@ares:~$ show-code point.h


point.h:


```

```
 1  #ifndef POINT_CLASS_HEADER_INCLUDED
 2  #define POINT_CLASS_HEADER_INCLUDED
 3
 4  #include <iostream>
 5  #include <istream>
 6  #include <ostream>
 7
 8  // A 2D point class
 9  class Point
10  {
11      double x, // x coordinate of point
12             y; // y coordinate of point
13
14  public:
15      Point(void);
16      Point(double new_x, double new_y);
17
18      void Output(void);   // output this point
19      void Input(void);    // input this point
20      double distance(Point other);   // distance between this point and othe
21
22      double get_x(void) const { return x; }
23      double get_y(void) const { return y; }
24
25      void set_x(double new_x);
26      void set_y(double new_y);
27
28      Point flip_x(void);
29      Point flip_y(void);
30
31      Point shift_x(double move_by);
32      Point shift_y(double move_by);
33
34          double operator-(const Point& other) const;
35      friend std::istream& operator>>(std::istream& in, Point& p);
36      friend std::ostream& operator<<(std::ostream& out, const Point& p);
37      bool operator==(const Point& other) const;
38      bool operator!=(const Point& other) const;
39      Point operator/(const Point& other) const;
40          double& operator[](char coordinate);
41
42  };
43
44  #endif
```

```
ee43254@ares:~$ show-code point.cpp


point.cpp:


```

```
 1  #include "point.h"
 2
 3  #include <iostream>
 4  #include <cmath>
 5
 6  using namespace std;
 7
 8  // read standard 2D point notation (x,y) -- ignore
 9  // window dressing
10  void Point::Input(void)
11  {
12      char dummy;
13      cin >> dummy >> x >> dummy >> y >> dummy;
14      return;
15  }
16
17  // output standard 2D point notation (x,y)
18  void Point::Output(void)
19  {
20      cout << '(' << x << ", " << y << ')';
21      return;
22  }
```

```cpp

// calculate distance between two 2D points --
// the one that called us and the argument
double Point::distance(Point other)
{
    return sqrt(pow(other.x-x, 2.0) +
                pow(other.y-y, 2.0));
}

// set coordinates to programmer-specified values
void Point::set_x(double new_x)
{
    x = new_x;        // no error checking since anything is legal
    return;
}

// set coordinates to programmer-specified values
void Point::set_y(double new_y)
{
    y = new_y;        // no error checking since anything is legal
    return;
}

// construct Point by default -- no values specified
Point::Point(void)
        : x(0.0), y(0.0) {}

// construct Point given initial x,y values
Point::Point(double new_x, double new_y)
        : x(new_x), y(new_y) {}

// creates a point flipped about the x axis from us
Point Point::flip_x(void)
{
    return Point(x,-y);
}

// creates a point flipped about the y axis from us
Point Point::flip_y(void)
{
    return Point(-x,y);
}

// creates a point shifted along the x axis from us
Point Point::shift_x(double move_by)
{
    return Point(x+move_by,y);
}

// creates a point shifted along the y axis from us
Point Point::shift_y(double move_by)
{
    return Point(x,y+move_by);
}


// Overload operator- for distance between two points
double Point::operator-(const Point& other) const
{
        return sqrt(pow(other.get_x() - get_x(), 2.0) +
                pow(other.get_y() - get_y(), 2.0));
}

// Overload operator>> for input
istream& operator>>(istream& in, Point& p)
{
    char dummy;
    in >> dummy >> p.x >> dummy >> p.y >> dummy;
    return in;
}

// Overload operator<< for output
ostream& operator<<(ostream& out, const Point& p)
{
    out << '(' << p.x << ", " << p.y << ')';
    return out;
}

// Overload operator== for equality
bool Point::operator==(const Point& other) const
{
// Included a tolerance because the operator== was throwing warnings of
// floating-point conversion. This meant that if the comparison was too
// large it could lead to problems so this is the only fix that I found.
    const double range = 1e-8;
    return (abs(x - other.x) < range) && (abs(y - other.y) < range);
}

// Overload operator!= for inequality
bool Point::operator!=(const Point& other) const
{
    return !(*this == other);
}

// Overload operator/ for midpoint
Point Point::operator/(const Point& other) const
{
    return Point((x + other.x) / 2.0, (y + other.y) / 2.0);
}

// Overloading operator for []
double& Point::operator[](char coordinate)
{
    if (coordinate == 'x' || coordinate == 'X')
    {
        return x;
    }
    else if (coordinate == 'y' || coordinate == 'Y')
    {
```

```
        return y;
    }
    else
    {
        cerr << "Invalid coordinate specified";
    }
        static double dummy = 0.0; // Dummy value to return in case of erro
    return dummy;
}

// Driver program
int main()
{
    Point p1(1.0, 2.0);
    Point p2(4.0, 6.0);

        cout << "Generating two points...\n";
        cout << "Point 1 of: " << p1 << "\n";
        cout << "Point 2 of: " << p2 << "\n";

    // Usage of overloaded operators
    double distance1 = p1 - p2;
    cout << "The distance between p1 and p2 are: " << distance1
                << " units\n" << endl;

    cout << "Enter a point: ";
    cin >> p1;
    cout << "You entered: " << p1 << endl;
        cout << "Setting Point 1 to " << p1 << endl;

        double distance2 = p1 - p2;
    cout << "The new distance between p1 and p2 are: " << distance2
                << " units\n" << endl;

    cout << "\nAre p1 and p2 equal? "  << endl;
        if(p1==p2) {
                cout << "Yes, p1 and p2 are equal.\n" << endl;
        }
        else {
                cout << "No, p1 and p2 are not equal.\n" << endl;
        }

    cout << "Are p1 and p2 not equal? " << endl;
        if(p1!=p2) {
                cout << "Yes, p1 and p2 are not equal.\n" << endl;
        }
        else {
                cout << "No, p1 and p2 are equal.\n" << endl;
        }

    Point midpoint = p1 / p2;
    cout << "The midpoint between p1 and p2 is: " << midpoint << endl;

        Point p3(3.2, 9.8);
        cout << "\nGenerating a third point of: " << p3 << endl;
        double my_x, my_y;
        my_x = p3['x'];
        my_y = p3['y'];
        cout << "\nThe x-value of Point 3 is " << my_x << endl;
        cout << "The y-value of Point 3 is " << my_y << endl;

    return 0;
}
```

```
ee43254@ares:~$ CPP point
point.cpp***


ee43254@ares:~$ ./point.out
Generating two points...
Point 1 of: (1, 2)
Point 2 of: (4, 6)
The distance between p1 and p2 are: 5 units

Enter a point: (10, 5)
You entered: (10, 5)
Setting Point 1 to (10, 5)
The new distance between p1 and p2 are: 6.08276 units


Are p1 and p2 equal?
No, p1 and p2 are not equal.

Are p1 and p2 not equal?
Yes, p1 and p2 are not equal.

The midpoint between p1 and p2 is: (7, 5.5)

Generating a third point of: (3.2, 9.8)

The x-value of Point 3 is 3.2
The y-value of Point 3 is 9.8
ee43254@ares:~$ ./point.out
Generating two points...
Point 1 of: (1, 2)
Point 2 of: (4, 6)
The distance between p1 and p2 are: 5 units

Enter a point: (4, 6)
You entered: (4, 6)
Setting Point 1 to (4, 6)
The new distance between p1 and p2 are: 0 units


Are p1 and p2 equal?
Yes, p1 and p2 are equal.

Are p1 and p2 not equal?
No, p1 and p2 are equal.
```

```
The midpoint between p1 and p2 is: (4, 6)

Generating a third point of: (3.2, 9.8)

The x-value of Point 3 is 3.2
The y-value of Point 3 is 9.8
ee43254@ares:~$ ./point.out
Generating two points...
Point 1 of: (1, 2)
Point 2 of: (4, 6)
The distance between p1 and p2 are: 5 units

Enter a point: (2, 3)
You entered: (2, 3)
Setting Point 1 to (2, 3)
The new distance between p1 and p2 are: 3.60555 units


Are p1 and p2 equal?
No, p1 and p2 are not equal.

Are p1 and p2 not equal?
Yes, p1 and p2 are not equal.

The midpoint between p1 and p2 is: (3, 4.5)

Generating a third point of: (3.2, 9.8)

The x-value of Point 3 is 3.2
The y-value of Point 3 is 9.8
ee43254@ares:~$ cat point.tpq
1. Which operators are members and which are non-members? Do any have to

be members?



Generally, unary operators are often implemented as member functions,

while binary operators may be implemented as either member functions or

non-member functions. Some operators, like assignment (=), subscript ([]),

and function call (()) must be member functions. However, many operators,

especially binary ones, can be implemented as non-member functions or as

friend functions.


2. Which operators should be const? What other methods might well be made const?

In general, what is the rule which determines if a method should be made const?


Member functions that do not modify the state of the object they are called on

should be marked as const. This includes functions like accessors, operators

that only retrieve data, and any other methods that don't modify the object's

internal state. The rule for determining if a method should be made const is

whether or not it modifies the object's state. If it does not modify the state,

it should be const.


3. What type do equality and inequality return? Input? Output? Assignment?


Equality and inequality operators (== and !=) typically return a bool type.

Input, Output, and Assignment are all void.


4. Do you agree with your friend's decision to use operator/ for midpoint?

Why/Why not?


Using the / operator for the midpoint calculation might not be the best

choice in terms of clarity and readability. While it's not inherently wrong,

it might lead to confusion or ambiguity about the purpose of the operator.

It's often better to choose operators that intuitively represent the operation

being performed, and using a separate method like midpoint() might be clearer.


5. Why didn't you overload operators for less than, greater than, etc.?


Overloading comparison operators like less than (<) and greater than (>)

can lead to ambiguities and unexpected behavior, especially in cases where
```

there are multiple valid ways to compare objects.

6. Your friend wanted to overload operators for the flip and shift methods, too (~ and += respectively). Why did you talk them out of it? Why wasn't this a good idea?

Overloading operators like ~ and += for flip and shift methods might not be a good idea because these operators are not commonly associated with those operations. Using operators in unconventional ways can lead to confusion and make the code less readable and maintainable. It's better to stick to standard conventions and use descriptive method names for these operations.

7. Just because you've added operators, should you necessarily remove the old methods that did these jobs?

Not necessarily. While operators can provide improve syntax and improve readability in some cases, the old methods may still have their uses, especially if they are well-named and clearly convey the intended operation. It's generally a good idea to keep both the operators and the methods if they serve distinct purposes or if there's a need to maintain compatibility with existing code.

MORE TPQS

1. Should the programmer be able to do: p['X'] = 2.0; to change the X-coordinate of a Point object p?

Allowing direct access to the x-coordinate using the subscript operator [] might not be a good idea as it could lead to confusion and potential misuse.

2. If you were going to allow such behavior, how would you do it?

Instead, you could provide separate member functions to set the x and y coordinates, such as set_x() and set_y(). These functions would ensure proper encapsulation and allow for better control over the modification of object state.ee43254@ares:~$ exit
exit

Script done on 2024-05-09 18:09:17-05:00 [COMMAND_EXIT_CODE="0"]