**Operator Overloading**

**Functions defined as not members of the class.**

The operator overloading functions will be implemented in the Testing Program only since they are not members of the class.

Because they are not members of the class, the number of parameters must be the same as the number of operators needed for the operation. If you are doing overloading the + addition then as you know the addition needs two operators(values) therefore the number of parameters in your function must be two as well.

There as some cases I use const and there are other cases that I don’t use const. Please remember that const prevent us to change the data members (we won’t be able to use set functions).

I don’t use const in the pre and post increment because as you know the idea of pre and post increments is to change the information of the data members. The same applies to the >> (istream).

Because functions are not members of the class then the data members can’t be accessed directly but data members must be accessed through functions (get, set, etc)

|  |  |  |
| --- | --- | --- |
| Headed file | Header implementation | Testing Program |
| point.h | point.cpp | **Test.cpp** (contains main) |
| #pragma once  class point  {  int x;  int y;  public:  point();  point(int);  point(int, int);  int getX() const;  int getY() const;  void setX(int);  void setY(int);  }; | #include <iostream>  #include "point.h"  point::point() :x(0), y(0) {}  point::point(int newX) {  setX(newX);  setY(0);  }  point::point(int newX, int newY) {  setX(newX);  setY(newY);  }  int point::getX() const { return x; }  int point::getY() const { return y; }  void point::setX(int newX) {  x = newX;  }  void point::setY(int newY) {  y = newY;  } | Implementation of the main and the operator overloading functions is done outside of the class because they are not members |

**Test.cpp**

#include <iostream>

#include "point.h"

using namespace std;

//functions defined as not members of the class

//the number of parameters will be the same

//number of operands used in the operation

//so addition needs to operands therefore

//the overloading function will need two parameters

//addition

//a+b

point operator+(const point&, const point&);

//subtraction

//a-b

point operator-(const point&, const point&);

//unary operator

//-a

point operator-(const point&);

//comparison

//a<b

bool operator<(const point&, const point&);

//pre-increment

//b=++a;

point operator++(point&);

//post-increment

//b=a++;

point operator++(point&,int);

//output stream

//cout<<a;

ostream& operator<<(ostream&, const point&);

//input stream

//cin>>a;

istream& operator>>(istream&, point&);

int main()

{

point p1,p2;

cout << "enter x and y for point 1:";

cin >> p1;

cout << "enter x and y for point 2:";

cin >> p2;

cout << p1 << p2<< endl;

cout <<"addition: " << p1 + p2 << endl;

point result = p1 - p2;

cout << "subtraction: " << result << endl;

result = -p1;

cout << "unary operator: " << result << endl;

if (p1 < p2)

cout << "p1 is less than p2";

else

cout << "p1 is greater or equals to p2";

cout << endl;

return 0;

}

//addition

//a+b

point operator+(const point& left, const point& right) {

return point(left.getX()+right.getX(),left.getY()+right.getY());

}

//subtraction

//a-b

point operator-(const point& left, const point& right) {

return point(left.getX() - right.getX(), left.getY() - right.getY());

}

//unary operator

//-a

point operator-(const point& opt) {

return point(-opt.getX(),-opt.getY());

}

//comparison

//a<b

bool operator<(const point& left, const point& right) {

return left.getX() < right.getX() && left.getY() < right.getY();

}

//pre-increment

//const is not being used here because we want to change the

//values of the data members in other words we need to

//access the set functions

//b=++a;

point operator++(point& p) {

p.setX(p.getX() + 1);

p.setY(p.getY() + 1);

return point(p.getX(),p.getY());

}

//post-increment

//const is not being used here because we want to change the

//values of the data members in other words we need to

//access the set functions

//b=a++;

point operator++(point& p, int) {

//the value return to be should be the value of

//a before the incremnt so

//first create a temp to store values of a

point temp;

temp.setX(p.getX());

temp.setY(p.getY());

//second increment the values of a

p.setX(p.getX() + 1);

p.setY(p.getY() + 1);

//return the temp object which has the original values of a

return temp;

}

//output stream

//cout<<a;

ostream& operator<<(ostream& out, const point& p) {

out << "(" << p.getX() << "," << p.getY() << ")\n";

return out;

}

//input stream

//const is not used here because we intend to change

//the values of the data members in other words

//we must access the set functions

//cin>>a;

istream& operator >> (istream& input, point& p) {

int newX, newY;

input >> newX >> newY;

p.setX(newX);

p.setY(newY);

return input;

}

**Functions defined as members of the class.**

The operator overloading functions will be implemented in the header implementation file and not in the testing file.

As members of the class, the number of parameters must be 1 less than the number of operators needed for the operation. If you are doing overloading the + addition then as you know the addition needs two operators(values) therefore the number of parameters in your function will be only one since the left operator is the one calling the operator and passing the missing object.

C = A + B; //A is calling the + function which means that the data members of A can be accessed directly

//without the need of a dot operator so what is missing is B (left operand) which is the only

//value passed to the function

The implementation needs the name of the class in front for example **point::**

|  |  |  |
| --- | --- | --- |
| Headed file | Header implementation | Testing Program |
| point.h | point.cpp | **Test.cpp** (contains main) |
| #pragma once  class point  {  int x;  int y;  public:  point();  point(int);  point(int, int);  int getX() const;  int getY() const;  void setX(int);  void setY(int);  //addition  //a+b  point operator+(const point&);  //subtraction  //a-b  point operator-(const point&);  //unary operator  //-a  point operator-();  //comparison  //a<b  bool operator<(const point&);  //pre-increment  //b=++a;  point operator++();  //post-increment  //b=a++;  point operator++(int);  }; | #include <iostream>  #include "point.h"  point::point() :x(0), y(0) {}  point::point(int newX) {  setX(newX);  setY(0);  }  point::point(int newX, int newY) {  setX(newX);  setY(newY);  }  int point::getX() const { return x; }  int point::getY() const { return y; }  void point::setX(int newX) {  x = newX;  }  void point::setY(int newY) {  y = newY;  }  //addition  //a+b  point point::operator+(const point& right) {  return point(x + right.x, y + right.y);  }  //subtraction  //a-b  point point::operator-(const point& right) {  return point(x - right.x, y - right.y);  }  //unary operator  //-a  point point::operator-() {  return point(-x, -y);  }  //comparison  //a<b  bool point::operator<(const point& right) {  return x < right.x && y < right.y;  }  /\*  pre-increment  const is not being used here because we want to change the values of the data members in other words we need to access the set functions  \*/  //b=++a;  point point::operator++() {  x++;  y++;  return point(x,y);  }  //post-increment  /\*const is not being used here because we want to change the values of the data members in other words we need to access the set functions\*/  //b=a++;  point point::operator++(int) {  //first store value of a in a temp object  point temp;  temp.x = x;  temp.y = y;  //second increment the values of a  x++;  y++;  //return temp which has original values of a  return temp;  } | Implementation of the main  The << and >> can’t be members of the class so they are implemented in the test program |

**Test.cpp**

#include <iostream>

#include "point.h"

using namespace std;

//output stream

//cout<<a;

ostream& operator<<(ostream&, const point&);

//input stream

//const is not used here because we intend to change

//the values of the data members in other words

//we must access the set functions

//cin>>a;

istream& operator >> (istream&, point&);

int main()

{

point p1,p2;

cout << "enter x and y por point 1:";

cin >> p1;

cout << "enter x and y por point 2:";

cin >> p2;

cout << p1 << p2<< endl;

cout <<"addition: " << p1 + p2 << endl;

point result = p1 - p2;

cout << "subtraction: " << result << endl;

result = -p1;

cout << "unary operator: " << result << endl;

if (p1 < p2)

cout << "p1 is less than p2";

else

cout << "p1 is greater or equals to p2";

cout << endl;

return 0;

}

//output stream

//cout<<a;

ostream& operator<<(ostream& out, const point& p) {

out << "(" << p.getX() << ","

<< p.getY() << ")\n";

return out;

}

//input stream

//const is not used here because we intend to change

//the values of the data members in other words

//we must access the set functions

//cin>>a;

istream& operator >> (istream& input, point& p) {

int newX, newY;

input >> newX >> newY;

p.setX(newX);

p.setY(newY);

return input;

}

**Friend functions.**

Friend functions are not members of the class which means that the numbers of the parameters is the same as the number of operands of the operation (same as not member functions)

The difference is that the friend functions even though are not members of the class they will be able to access the data members directly instead through functions. You don’t need **point::** before the function.

|  |  |  |
| --- | --- | --- |
| Headed file | Header implementation | Testing Program |
| point.h | point.cpp | **Test.cpp** (contains main) |
| #pragma once  #include <iostream>  using namespace std;  class point  {  int x;  int y;  public:  point();  point(int);  point(int, int);  int getX() const;  int getY() const;  void setX(int);  void setY(int);  //addition  //a+b  friend point operator+(const point&,const point&);  //subtraction  //a-b  friend point operator-(const point&, const point&);  //unary operator  //-a  friend point operator-(const point&);  //comparison  //a<b  friend bool operator<(const point&, const point&);  //pre-increment  //b=++a;  friend point operator++(point&);  //post-increment  //b=a++;  friend point operator++(point&,int);  //output stream  //cout<<a;  friend ostream& operator<<(ostream&, const point&);  //input stream  //const is not used here because we intend to change  //the values of the data members in other words  //we must access the set functions  //cin>>a;  friend istream& operator>>(istream&, point&);  }; | #include <iostream>  #include "point.h"  point::point() :x(0), y(0) {}  point::point(int newX) {  setX(newX);  setY(0);  }  point::point(int newX, int newY) {  setX(newX);  setY(newY);  }  int point::getX() const { return x; }  int point::getY() const { return y; }  void point::setX(int newX) {  x = newX;  }  void point::setY(int newY) {  y = newY;  }  //addition  //a+b  point operator+(const point& left, const point& right) {  return point(left.x + right.x, left.y + right.y);  }  //subtraction  //a-b  point operator-(const point& left, const point& right) {  return point(left.x - right.x, left.y - right.y);  }  //unary operator  //-a  point operator-(const point& p) {  return point(-p.x, -p.y);  }  //comparison  //a<b  bool operator<(const point& left, const point& right) {  return left.x < right.x && left.y < right.y;  }  //pre-increment  //const is not being used here because we want to change the  //values of the data members in other words we need to  //access the set functions  //b=++a;  point operator++(point& p) {  p.x++;  p.y++;  return point(p.x,p.y);  }  //post-increment  //const is not being used here because we want to change the  //values of the data members in other words we need to  //access the set functions  //b=a++;  point operator++(point& p,int) {  //the value return to be should be the value of  //a before the incremnt so  //first create a temp to store values of a  point temp;  temp.x = p.x;  temp.y = p.y;  //second increment the values of a  p.x++;  p.y++;  //return the temp object which has the original values of a  return temp;  }  //output stream  //cout<<a;  ostream& operator<<(ostream& out, const point& p) {  out << "(" << p.x << ","  << p.y << ")\n";  return out;  }  //input stream  //const is not used here because we intend to change  //the values of the data members in other words  //we must access the set functions  //cin>>a;  istream& operator >> (istream& input, point& p) {  input >> p.x >> p.y;  return input;  } | Implementation of the main only |

**Test.cpp**

#include <iostream>

#include "point.h"

using namespace std;

int main()

{

point p1,p2;

cout << "enter x and y por point 1:";

cin >> p1;

cout << "enter x and y por point 2:";

cin >> p2;

cout << p1 << p2<< endl;

cout <<"addition: " << p1 + p2 << endl;

point result = p1 - p2;

cout << "subtraction: " << result << endl;

result = -p1;

cout << "unary operator: " << result << endl;

if (p1 < p2)

cout << "p1 is less than p2";

else

cout << "p1 is greater or equals to p2";

cout << endl;

return 0;

}

**Using Pointers**

**Creating a dynamic array**

#include <iostream>

using namespace std;

void fillArray(int a[],int sizeA)

{

for (int i = 0; i < sizeA; i++)

a[i] = i + 1;

}

void print(int a[],int sizeA)

{

for (int i = 0; i < sizeA; i++)

cout << a[i] << endl;

}

int main()

{

//asking user to enter size of array

int sizeA;

cout << "enter size of array: ";

cin >> sizeA;

//creation of dynamic array in heap

int \*intArr = new int[sizeA];

fillArray(intArr,sizeA);

print(intArr,sizeA);

//delete memory allocated for the array in the heap

delete[] intArr;

//delete the dangling pointer

intArr = NULL;

return 0;

}