**CS 37 Notes Part 5**

**overld1.cpp** It’s easier to write out cout << pupil than pupil.printData(); How do we make that happen?

This program only overloads the stream insertion and stream extraction operators. These two operators only are implemented using friend functions. All the other operators are implemented through class methods.

NOTE: insertion = output while exertion = input

OUTLINE

The main uses cin and cout to call the two friend functions.

**int** main()

{

    Student pupil;

    cout << "enter three test scores ";

    cin >> pupil;

    cout << endl << "the student information is " << endl;

    cout << pupil << endl;

    return 0;

}

The first thing to do in the class is to declare the operator<< and operator>> as friend functions. Note that both parameters are reference as value parameters are not used because a local copy needs to be made (takes additional time and memory) for value parameters.

**class** Student

{

**friend** ostream **&**operator<<(ostream**&**, Student**&**);

**friend** istream **&**operator>>(istream**&**, Student**&**);

|  |
| --- |
| The stream insertion operator<< friend function has two parameters: the output stream of type ostream and the object that is to be printed.  ostream **&**operator<<( ostream **&**output, Student **&**one)  {      output << "test 1 is " << one.test1 << endl;      output << "test 2 is " << one.test2 << endl;      output << "test 3 is " << one.test3 << endl;      output << "the average is " << one.avg << endl;      return output;*// enables cascading eg cout << a << b << c;*  }  *Note the object has a const declaration in front of it in the parameter list as the object being printed does not change value in the friend function.* (?????????? I think he meant there isn’t a const)  **class** Student  {  **friend** ostream **&**operator<<(ostream**&**, Student**&**);  **friend** istream **&**operator>>(istream**&**, Student**&**); |
| The stream extraction operator >> friend function also takes two parameters which is a reference to the input stream of type istream and the object whose values are to be entered by the user.  istream **&**operator >>( istream **&**input, Student **&**one)  {      input >> one.test1  >> one.test2 >> one.test3;      one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;      return input;*// enables cascading eg cin >> a >> b >> c;*  }  *Note that there is no const in this parameter list as the object is changing value to the values entered by the user.*  **class** Student  {  **friend** ostream **&**operator<<(ostream**&**, Student**&**);  **friend** istream **&**operator>>(istream**&**, Student**&**); |

The stream insertion friend function places all the values to be printed into the output stream.

istream **&**operator >>( istream **&**input, Student **&**one)

{

    input >> one.test1  >> one.test2 >> one.test3;

    one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;

    return input;*// enables cascading eg cin >> a >> b >> c;*

}

This program uses the variable name output for the output stream but any legal variable name can be used. *The last line must return the output stream, that is return output; This must be done to allow cascading in the main, eg cout << a << b << c;*

ostream **&**operator<<(ostream **&**output, Student **&**one)

{

    output << "test 1 is " << one.test1 << endl;

    output << "test 2 is " << one.test2 << endl;

    output << "test 3 is " << one.test3 << endl;

    output << "the average is " << one.avg << endl;

    return output;*// enables cascading eg cout << a << b << c;*

}

This cout statement will call the stream insertion friend function three times.

First object a is put into the output stream and the output stream is returned by the friend function. *The friend function is called again and object b is put into the output stream and the output stream is returned. The friend function is called a third time so that object c can be put into the output stream and the output stream is returned. (?????????????)*

ostream **&**operator<<(ostream **&**output, Student **&**one)

{

    output << "test 1 is " << one.test1 << endl;

    output << "test 2 is " << one.test2 << endl;

    output << "test 3 is " << one.test3 << endl;

    output << "the average is " << one.avg << endl;

    return output;*// enables cascading eg cout << a << b << c;*

}

The stream extraction friend function requires three steps:

1. Extract the data members entered by the user for the object. The friend function first extracts the three test scores from the input stream. The variable name used in this program for the input stream is input; but, any legal variable name can be used.

*// overloaded stream-extraction operator*

istream **&**operator >>(istream **&**input, Student **&**one)

{

    input >> one.test1  >> one.test2 >> one.test3;

    one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;

    return input;*// enables cascading eg cin >> a >> b >> c;*

}

1. Calculate the remaining data members not entered by the user. The friend function calculates the average data member for the object.

*// overloaded stream-extraction operator*

istream **&**operator >>(istream **&**input, Student **&**one)

{

    input >> one.test1  >> one.test2 >> one.test3;

    one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;

    return input;*// enables cascading eg cin >> a >> b >> c;*

}

1. Make sure to return the input stream at the end of the friend function. That is, return input;

*// overloaded stream-extraction operator*

istream **&**operator >>(istream **&**input, Student **&**one)

{

    input >> one.test1  >> one.test2 >> one.test3;

    one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;

    return input;*// enables cascading eg cin >> a >> b >> c;*

}

This allows for cascading in the main, eg cin >> a >> b >> c; This cascading works similarly to the stream insertion friend function as each time the friend function is called, the input stream is returned.

    return output;*// enables cascading eg cout << a << b << c;*

or

    return input;*// enables cascading eg cin >> a >> b >> c;*

|  |
| --- |
| #include<iostream>  using **namespace** std;  *// overloaded operators are used to make the main section*  *// more consistent and easier to write*  *// eg easier to write cout << pupil; than pupil.printData();*  **class** Student  {  **friend** ostream **&**operator<<(ostream**&**, Student**&**);*// output*  **friend** istream **&**operator>>(istream**&**, Student**&**);*// input*    **private:**  **int** test1, test2, test3;  **float** avg;  };    *// overloaded stream-insertion operator*  ostream **&**operator<<(ostream **&**output, Student **&**one)  {      output << "test 1 is " << one.test1 << endl;      output << "test 2 is " << one.test2 << endl;      output << "test 3 is " << one.test3 << endl;      output << "the average is " << one.avg << endl;      return output;*// enables cascading eg cout << a << b << c;*  }  *// overloaded stream-extraction operator*  istream **&**operator >>(istream **&**input, Student **&**one)  {      input >> one.test1  >> one.test2 >> one.test3;      one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;      return input;*// enables cascading eg cin >> a >> b >> c;*  }  **int** main()  {      Student pupil;      cout << "enter three test scores ";      cin >> pupil;      cout << endl << "the student information is " << endl;      cout << pupil << endl;      return 0;  }  */\**  *enter three test scores 10 50 20*  *the student information is*  *test 1 is 10*  *test 2 is 50*  *test 3 is 20*  *the average is 26.6667*  *Press any key to continue*  *\*/* |

**overld2a.cpp / overld2b.cpp** What if we do it with +=, ++preincrement, and postincrement++? instead of >> and <<? This program overloads the += , ++ preincrement, ++ postincrement and has the same stream insertion and stream extraction friend functions as overld1.cpp.

The overld2a.cpp main demonstrates ++s; and s++ where only one task is done per line of code. Overld2b.cpp demonstrates lines of code like cout << ++s; and cout << s++; where more than one task is done per line of code. overld2a.cpp and overld2b.cpp have identical classes and friend functions; only the mains are different. *Please remember that in the line cout << ++s; 1 is added to each test score and then the object is printed (preincrement). cout >> s++; first prints the object as is and then adds 1 to each test score (postincrement)*

*overworld2a.cpp*

The += operator will assign extra credit to each of the three test scores.

    pupil+=5;*//assign 5 extra credit points to each test*

    cout << endl << "after extra credit the scores are " << endl;

    cout << pupil << endl;

The two ++ operators will add 1 point to each of the three tests.

    ++pupil;*//add 1 to each test - preincrement*

    cout << endl << "after preincrementing the scores are " << endl;

    cout << pupil << endl;

    pupil++;*//add 1 to each test - postincrement*

    cout << endl << "after postincrementing the scores are " << endl;

    cout << pupil << endl;

The class for both programs begins with the friend declarations for the stream insertion and stream extraction operators. This means that we will still be using cin >> pupil and cout << pupil

**class** Student

{

**friend** ostream **&**operator<<( ostream**&**, **const** Student**&**);

**friend** istream **&**operator>>( istream**&**, Student**&**);

The public section then declares the three operator overloading methods. Note that the prototype for the preincrement is Student &operators++(); as this method will modify the object and then return \*this;

**public:**

    Student **&**operator+=(**int**);*// modify the object*

    Student **&**operator++();*// preincrement*

    Student operator++(**int**);*// postincrement - dummy int parameter*

The prototype for the postincrement which must be different than the preincrement prototype is Student operator++(int); Also note the (int), this is called a dummy parameter as there is no int being passed. The int just tells C++ that this method is for the postincrement.

Note there is no & as this method will NOT return \*this. **Please remember that any function that returns \*this; requires & in the prototype. += and ++ preincrement will return \*this but ++ postincrement will not return \*this.**

*The predecrements work exactly the same with the following prototypes (hypothetical)*

    Student **&**operator--(**int**);*// postincrement*

    Student operator--(**int**);*// preincrement*

*The stream insertion and stream extraction friend functions are the same as overld1.cpp because the operators -=, \*=, /=, %= are similar to +=. The – predecrement and --postdecrement are similar to ++ preincrement and ++ postincrement.*

OUTLINE

The operator += method takes one parameter which is the amount of extra credit to add to each of the three test scores.

    Student **&**operator+=(**int**);*// modify the object*

If in the main, s += 5; is done, the 5 automatically gets passed to the extraCredit parameter.

    pupil+=5;*//assign 5 extra credit points to each test*

Which calls

Student **&**Student::operator+=(**int** extraCredit)

{

    test1+= extraCredit;

    test2+= extraCredit;

    test3+= extraCredit;

    avg = (test1 + test2 + test3) / (**float**) 3;

    return \*this;

}

*The method adds to each test score and recalculates the average.* Lastly, return \*this is done as the object has changed value and needs to be returned to the main.

The operator ++ preincrement method adds one to each test score, recalculates the average, and lastly returns \*this. If in the main we do, cout <<++s; object s needs to be returned so it can be printed.

    ++pupil;*//add 1 to each test - preincrement*

    cout << endl << "after preincrementing the scores are " << endl;

    cout << pupil << endl;

Which calls ++method

Student **&**Student::operator ++()*// preincrement*

{

    test1++;

    test2++;

    test3++;

    avg = (test1 + test2 + test3) / (**float**) 3;

    return \*this;

}

The operator ++ postincrement method first makes a copy of the object since the unchanged object is the one that needs to be returned if we do cout << s++; in the main.

*// POSTINCREMENT*

Student Student::operator ++(**int**)*// postincrement - note dummy int parameter has no name*

{

    Student temp = \*this;

    test1++;

    test2++;

    test3++;

    avg = (test1 + test2 + test3) / (**float**) 3;

    return temp;*// return non-incremented value*

}

The original value of s (pupil) must be printed yet the object must change value too. After the copy of the object is made, the original object is updated and the average is recalculated.

*// POSTINCREMENT*

Student Student::operator ++(**int**)*// postincrement - note dummy int parameter has no name*

{

    Student temp = \*this;

    test1++;

    test2++;

    test3++;

    avg = (test1 + test2 + test3) / (**float**) 3;

    return temp;*// return non-incremented value*

}

Lastly, the unmodified object is returned so that the unmodified object is printed. (?????????)

*// POSTINCREMENT*

Student Student::operator ++(**int**)*// postincrement - note dummy int parameter has no name*

{

    Student temp = \*this;

    test1++;

    test2++;

    test3++;

    avg = (test1 + test2 + test3) / (**float**) 3;

    return temp;*// return non-incremented value*

}

|  |
| --- |
| #include<iostream>  using **namespace** std;  **class** Student  {  **friend** ostream **&**operator<<( ostream**&**, **const** Student**&**);  **friend** istream **&**operator>>( istream**&**, Student**&**);  **public:**      Student **&**operator+=(**int**);*// modify the object*  *//increments*      Student **&**operator++();*// preincrement*      Student operator++(**int**);*// postincrement - dummy int parameter*  *// specifices postincrement*  **private:**  **int** test1, test2, test3;  **float** avg;  };  *// EXTRA CREDIT*  Student **&**Student::operator+=(**int** extraCredit)  {      test1+= extraCredit;      test2+= extraCredit;      test3+= extraCredit;      avg = (test1 + test2 + test3) / (**float**) 3;      return \*this;  }    *// PREINCREMENT*  Student **&**Student::operator ++()  {      test1++;      test2++;      test3++;      avg = (test1 + test2 + test3) / (**float**) 3;      return \*this;  }  *// POSTINCREMENT*  Student Student::operator ++(**int**)*// postincrement - note dummy int parameter has no name*  {      Student temp = \*this;      test1++;      test2++;      test3++;      avg = (test1 + test2 + test3) / (**float**) 3;      return temp;*// return non-incremented value*  }  *// INPUT*  *// overloaded stream-extraction operator*  istream **&**operator>>( istream **&**input, Student **&**one)  {      input >> one.test1  >> one.test2 >> one.test3;      one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;      return input;*// enables cascading eg cin >> a >> b >> c;*  }  *// OUTPUT*  *// overloaded stream-insertion operator*  ostream **&**operator<<( ostream **&**output, **const** Student **&**one)  {      output << "test 1 is " << one.test1 << "   ";      output << "test 2 is " << one.test2 << "   ";      output << "test 3 is " << one.test3 << endl;      output << "the average is " << one.avg << endl;      return output;*// enables cascading eg cout << a << b << c;*  }  **int** main()  {      Student pupil;      cout << "enter three test scores ";      cin >> pupil;      cout << endl << "the student information is " << endl;      cout << pupil << endl;      pupil+=5;*//assign 5 extra credit points to each test*      cout << endl << "after extra credit the scores are " << endl;      cout << pupil << endl;      ++pupil;*//add 1 to each test - preincrement*      cout << endl << "after preincrementing the scores are " << endl;      cout << pupil << endl;      pupil++;*//add 1 to each test - postincrement*      cout << endl << "after postincrementing the scores are " << endl;      cout << pupil << endl;      return 0;  }  */\**  *enter three test scores 10 30 50*  *the student information is*  *test 1 is 10   test 2 is 30   test 3 is 50*  *the average is 30*  *after extra credit the scores are*  *test 1 is 15   test 2 is 35   test 3 is 55*  *the average is 35*  *after preincrementing the scores are*  *test 1 is 16   test 2 is 36   test 3 is 56*  *the average is 36*  *after postincrementing the scores are*  *test 1 is 17   test 2 is 37   test 3 is 57*  *the average is 37*  *Press any key to continue*  *\*/* |

*Overworld2b.cpp -* Main body comparison

Insertion and print for both programs are the same:

cout << "enter three test scores ";

cin >> pupil;

cout << endl << "the student information is " << endl;

cout << pupil << endl;

Preincrement comparison:

    ++pupil;*//add 1 to each test - preincrement*

    cout << endl << "after preincrementing the scores are " << endl;

    cout << pupil << endl;

versus:

*// test the preincrement*

    cout << "after preincrement " << endl << ++pupil << endl;

    cout << "the student information is " << endl << pupil << endl;

Postincrement comparison:

    pupil++;*//add 1 to each test - postincrement*

    cout << endl << "after postincrementing the scores are " << endl;

    cout << pupil << endl;

versus:

*// test the postincrement*

    cout << "after postincrement " << endl << pupil++ << endl;

    cout << "the student information is " << endl << pupil << endl;

|  |
| --- |
| *// same as last program but with different main*  *// to test pre and postincrements such as*  *// cout << ++pupil << endl;*  #include<iostream>  using **namespace** std;  **class** Student  {  **friend** ostream **&**operator<<( ostream**&**, **const** Student**&**);  **friend** istream **&**operator>>( istream**&**, Student**&**);  **public:**      Student **&**operator+=(**int**);*// modify the object*      Student **&**operator++();*// preincrement*      Student operator++(**int**);*// postincrement - dummy int parameter*  *// specifices postincrement*  **private:**  **int** test1, test2, test3;  **float** avg;  };  Student **&**Student::operator+=(**int** extraCredit)  {      test1+= extraCredit;      test2+= extraCredit;      test3+= extraCredit;      avg = (test1 + test2 + test3) / (**float**) 3;      return \*this;  }    Student **&**Student::operator ++()*// preincrement*  {      test1++;      test2++;      test3++;      avg = (test1 + test2 + test3) / (**float**) 3;      return \*this;  }  Student Student::operator ++(**int**)*// postincrement - note dummy*  *// int parameter has no name*  {      Student temp = \*this;      test1++;      test2++;      test3++;      avg = (test1 + test2 + test3) / (**float**) 3;      return temp;*// return non-incremented value*  }  *// overloaded stream-insertion operator*  ostream **&**operator<<( ostream **&**output, **const** Student **&**one)  {      output << "test 1 is " << one.test1 << "   ";      output << "test 2 is " << one.test2 << "   ";      output << "test 3 is " << one.test3 << endl;      output << "the average is " << one.avg << endl;      return output;*// enables cascading eg cout << a << b << c;*  }  *// overloaded stream-extraction operator*  istream **&**operator>>( istream **&**input, Student **&**one)  {      input >> one.test1  >> one.test2 >> one.test3;      one.avg = (one.test1 + one.test2 + one.test3) / (**float**) 3;      return input;*// enables cascading eg cin >> a >> b >> c;*  }  **int** main()  {      Student pupil;      cout << "enter three test scores ";      cin >> pupil;      cout << endl << "the student information is " << endl;      cout << pupil << endl;  *// test the preincrement*      cout << "after preincrement " << endl << ++pupil << endl;      cout << "the student information is " << endl << pupil << endl;  *// test the postincrement*      cout << "after postincrement " << endl << pupil++ << endl;      cout << "the student information is " << endl << pupil << endl;      return 0;  }  */\**  *enter three test scores 20 40 60*  *the student information is*  *test 1 is 20   test 2 is 40   test 3 is 60*  *the average is 40*  *after preincrement*  *test 1 is 21   test 2 is 41   test 3 is 61*  *the average is 41*  *the student information is*  *test 1 is 21   test 2 is 41   test 3 is 61*  *the average is 41*  *after postincrement*  *test 1 is 21   test 2 is 41   test 3 is 61*  *the average is 41*  *the student information is*  *test 1 is 22   test 2 is 42   test 3 is 62*  *the average is 42*  *Press any key to continue*  *\*/* |

**overld3.cpp** What if we do the same thing for + > and >> (without stream extraction)?

The program begins with the friend declaration for the stream insertion friend function.

The public section first has a prototype for the constructor

**public:**

    Student(**int** = 0, **int** = 0, **int** = 0);

followed by the operator+ and operator> methods.

    Student operator+(**const** Student **&**rhs) **const**

    {

        return Student(test1 + rhs.test1, test2 + rhs.test2, test3 + rhs.test3);

    }

**bool** operator> (**const** Student **&**rhs) **const**

    {

        return test1 > rhs.test1;

    }

These methods are short so were implemented in the class. It is fine to implement short methods in the declaration area.

The operator+ method requires one Student object parameter:

    Student operator+(**const** Student **&**rhs) **const**

As the method is being implemented in the declaration part of the class, the student:: is not needed. I included it in the code in case you do want to pull out the method and write it outside of the class. The const in the parameter list means the parameter does not change and the const at the end of the line means \*this does not change value in the method.

In the main, we can do s3 = s1 + s2;

The first object after = becomes the this pointer, s1 in this case.

The second object becomes the value of the parameter in the method, s2 in this case.

The parameter in the method is called rhs which means right hand side. Any legal variable name can be used as the parameter. The object returned goes into object s3 in the main. The return statement calculates each of the totals for the three test scores.

The operator> also requires one parameter.

In the main, we can do if(s2 > s1)

The first object becomes the this pointer, s2 in this case.

The second object becomes the value of the parameter rhs, s1 in this case.

This method returns a bool data type which is short for Boolean, that is true or false.

The method returns true if test1 of the first object is > test1 of the second object.

The stream instruction friend function is unchanged from the previous programs.

All the other relational operators, ==, != >=, <, <= are similar to >

All the other mathematical operators, -, \*, /, % are similar to +

|  |
| --- |
| #include<iostream>  using **namespace** std;  *// overloading binary and relational operators*  *// binary operators have two operands such as + - \* /*  *// relational operators see how two operands relate such as*  *// == != < <= > >=*  **class** Student  {  **friend** ostream **&**operator<<( ostream**&**, **const** Student**&**);  **public:**      Student(**int** = 0, **int** = 0, **int** = 0);      Student operator+(**const** Student **&**rhs) **const**      {          return Student(test1 + rhs.test1, test2 + rhs.test2, test3 + rhs.test3);      }  **bool** operator> (**const** Student **&**rhs) **const**      {          return test1 > rhs.test1;      }    **private:**  **int** test1, test2, test3;  };  *// constructor*  Student::Student(**int** a, **int** b, **int** c)          : test1(a), test2(b), test3(c)  {}    *// overloaded stream-insertion operator*  ostream **&**operator<<( ostream **&**output, **const** Student **&**one)  {      output << "test 1 is " << one.test1 << "   ";      output << "test 2 is " << one.test2 << "   ";      output << "test 3 is " << one.test3 << endl;      return output;*// enables cascading eg cout << a << b << c;*  }  *// stream-extraction operator not overloaded in this program*  **int** main()  {      Student s1(10,20,30), s2(1,2,3), s3;      cout <<"student 1 data is: " << s1 << endl;      cout <<"student 2 data is: " << s2 << endl;      s3 = s1 + s2;      cout <<"student 3 data is: " << s3 << endl;      if (s2 > s1)          cout << "student 2 has a higher test 1 score " << endl;      else          cout << "student 2 does not have a higher test 1 score " << endl;      return 0;  }  */\**  *student 1 data is: test 1 is 10   test 2 is 20   test 3 is 30*  *student 2 data is: test 1 is 1   test 2 is 2   test 3 is 3*  *student 3 data is: test 1 is 11   test 2 is 22   test 3 is 33*  *student 2 does not have a higher test 1 score*  *Press any key to continue*  *\*/* |

**overld4.cpp** – optional program.

This program is the beginning of a string library so for example, main programmers can do name1 = name2; instead of strcpy(name1, name2);

Similarly the relational operators could be overloaded so that the main programmer would not need to use the strcmp function.