|  |
| --- |
| Lab 05 Operator Overloading and Exception Handling |
|  |
| Exercise 1: Overloading Operator (40 min) |
| What operator(s) must be overloaded for the Rational class based on the following main function?  Complete the definition of the Rational class below to produce the 3 expected outputs.  int main() {  Rational r1, r2;  cout << "Input a rational number (num den): ";  cin >> r1;  cout << "Input a rational number (num den): ";  cin >> r2;  cout << r1 << " + " << r2 << " = " << r1 + r2 << endl;  cout << r1 << " - " << r2 << " = " << r1 - r2 << endl;  if (r1 < r2)  cout << r1 << " is smaller than " << r2 << endl;  else if (r1 == r2)  cout << r1 << " is equal to " << r2 << endl;  else  cout << r1 << " is larger than " << r2 << endl;  }  Expected output by run 1:  Input a rational number (num den): 1 2  Input a rational number (num den): 2 3  1/2 + 2/3 = 7/6  1/2 - 2/3 = -1/6  1/2 is smaller than 2/3  Expected output by run 2:  Input a rational number (num den): 1 2  Input a rational number (num den): 1 2  1/2 + 1/2 = 4/4  1/2 - 1/2 = 0/4  1/2 is equal to 1/2  Expected output by run 3:  Input a rational number (num den): 2 3  Input a rational number (num den): 1 2  2/3 + 1/2 = 7/6  2/3 - 1/2 = 1/6  2/3 is larger than 1/2 |
| Exercise 2: Overloading Operator (30 min) |
| What operator(s) must be overloaded for the Point class based on the following main function?  Complete the definition of the Point class below to produce the expected output.  int main() {  vector<Point> v;  Point p;  do {  cout << "Input a Point (0 0 to end): ";  cin >> p;  if (p == Point(0,0)) break;  v.push\_back (p);  } while (true);  cout << "Sort points by x then y:\n";  sort (v.begin(), v.end(), SortByXY());  for (int i = 0; i < v.size(); i++)  cout << v[i] << " ";  cout << endl;  cout << "Sort points by y then x:\n";  // Add one line of code to sort.  sort (v.begin(), v.end(), SortByYX());  for (int i = 0; i < v.size(); i++)  cout << v[i] << " ";  cout << endl;  }  Expected output:  Input a Point (0 0 to end): 2 3  Input a Point (0 0 to end): 3 2  Input a Point (0 0 to end): 1 1  Input a Point (0 0 to end): 2 1  Input a Point (0 0 to end): 1 2  Input a Point (0 0 to end): 0 0  Sort points by x then y:  (1,1) (1,2) (2,1) (2,3) (3,2)  Sort points by y then x:  (1,1) (2,1) (1,2) (3,2) (2,3) |

|  |
| --- |
| Exercise 3: Exception Handling (20 min) |
| Convert the following program to use exception handling.  #include <iostream>  #include <sstream>  #include <string>  using namespace std;  int main() {  string s; // input as string.  int n; // input as int.  cout << "Enter a positive integer or 0: ";  cin >> s;  if (!(istringstream (s) >> n))  cout << "Error: You input string \"" << s << "\".\n"  << "Input must be a positive integer or 0 only.\n";  else if (n < 0)  cout << "Error: You input negative integer " << n << ".\n"  << "Input must be a positive integer or 0 only.\n";  else {  long long result = 1;  for (int i = 1; i <= n; i++)  result \*= i;  cout << "Factorial(" << n << ") = " << result << endl;  }  cout << "Good bye.\n";  } |

|  |
| --- |
| Exercise 4: Exception Handling (20 min) |
| Complete the code for the catch blocks based on the expected output. You should have 4 catch blocks.  #include <iostream>  #include <string>  using namespace std;  int main () {  for (int i = 1; i < 7; i++) {  try {  if (i == 1) throw 1; // int  if (i == 2) throw 3.3; // double  if (i == 3) throw 'c'; // char  if (i == 4) throw string("str"); // string  if (i == 5) throw "const char[]"; // array of char  cout << "No exception.\n";  }  }  }  Expected output:  Unknown exception.  double exception: 3.3.  Unknown exception.  string exception: str.  const char[] exception: const char[].  No exception. |

|  |
| --- |
| Take Home Exercises |
| * 1. Design a Job class with three data fields—Job number, time in hours to complete the Job, and per-hour rate charged for the Job. Include overloaded extraction (<<) and insertion (>>) operators that get and display a Job’s values. Include overloaded + and – operators that return integers that indicate the total time for two Jobs, and indicate the difference in time between two Jobs, respectively. Write a driver program demonstrating that all the functions work correctly.   2. Design a class to hold a JobBid. Each JobBid contains a bid number and a quoted price. Each JobBid also contains overloaded extraction and insertion operators. Include an overloaded < (less than) function. A JobBid is considered lower than another JobBid when the quoted price is lower. Write a driver program that declares an array/vector of four JobBid objects. Find and display the lowest JobBid.   3. Design a SoccerPlayer class that includes three integer fields: a player’s jersey number, number of goals, and number of assists. Overload extraction and insertion operators for the class. Include an overloaded > (more than) function for the class. One SoccerPlayer is considered greater than another if the sum of goals plus assists is greater. Create an array of 11 SoccerPlayers, then use the > operator to find the player who has the greatest total of goals plus assists.   4. Create a class named RealEstate that has data members to hold the price of a house, the number of bedrooms, and the number of baths. Member functions include overloaded insertion and extraction operators. Write a driver program that instantiates a RealEstate object, allows the user to enter data, and displays the data members entered. The program should display an appropriate thrown error message if non-numeric or negative values are entered for any of the data members.   5. Create an Inventory class with data members for stock number, quantity, and price, and overloaded data entry and output operators. The data entry operator function should throw: * An error message, if the stock number is negative or higher than 999 * The quantity, if it is less than 0 * The price, if it is over $100.00   Then perform the following tasks:   1. Write a driver program that instantiates an array of five Inventory objects, and accepts data for each. Display an appropriate error message for each exception situation. When an exception is detected, replace all data fields with zeroes. At the end of the program, display data fields for all five objects. 2. Write a driver program that instantiates an array of five Inventory objects and accepts data for each. If an exception is thrown because of an invalid stock number, force the user to reenter the stock number for the Inventory object. If the quantity is invalid, do nothing. If the price is in error, then set the price to 99.99. (Add a setPrice() function to the Inventory class to accomplish this.) At the end of the program, display data fields for all five objects. 3. Write a driver program that instantiates an array of five Inventory objects and accepts data for each. If an exception is thrown because of an invalid stock number, force the stock number to 999; otherwise, do nothing. (Add any needed functions to the Inventory class.) At the end of the program, display data fields for all five objects. 4. Write a driver program that instantiates an array of five Inventory objects and accepts data for each. If any exception is thrown, stop accepting data. Display as many objects as were entered correctly. |