Nathan Roe
OOP with C++
EN.605.604 Section 81
3/13/2022

Module 5 Assignment: Probability

Design

For this assignment, the project requirements state that the program should provide calculations for Independent Probabilities using operators to find the following:

- & → Probability of A and B
- | → Probability of A and/or B
- ^ → Probability of A exclusiveOr B
- - → Probability of A but not B
- ~ → Probability of A not occuring

To do this, I will have a single class, IndependentProbability, that will represent a single probabilistic event, which will be constructed with a double. It will also need a function to get the probability of the event to receive the single private parameter that will store the probability.

Since Probabilities have to be contained in the bounds [0, 1], I will create an error if a developer attempts to construct an IndependentProbability event with an impossible probability.

Probabilities:

$$P(\sim A) = 1 - P(A)$$

$$P(A\&B) = P(A) \cdot P(B)$$

$$P(A|B) = 1 - P(\, P(\sim A)\, \&\, P(\sim B)\,)$$

$$P(A \wedge B) = 1 - P(\, P(\sim A)\, \&\, P(\sim B)\,) - P(A\&B) = P(A|B) - P(A\&B)$$

$$P(A - B) = P(A\, \&\, P(\sim B))$$

```cpp
#pragma once
#include <exception>
// @file
// @author Nathan Roe
// Object to represent an Independent Probabalistic Event
//
// Each IndependentProb stores an event probability created
// on construction, and can be accessed through getter.
// Event probabilities must be between 0.0 and 1.0, inclusive.
class IndependentProb
{
public:
    // Construct Probability Event with given probability.
    //
    // @param probability - likelyhood of event; must be
    //    between 0.0 and 1.0 inclusively, or will throw
    //    invalidProbability exception
    IndependentProb(double probability);

    // Get the probability of this event
    //
    // @return a double containing event probability
    double getProbability() const { return this->probability; };

    // Exception indicating an invalid probability
    // was provided.
    class invalidProbability : public std::exception
    {
    public:
        virtual const char *what() const throw()
        {
            return "Probability not between 0.0 and 1.0";
        }
    } invalidProbEx;

private:
    double probability = 0.0;
};

// NOT operator for finding P(NOT A)
IndependentProb operator~(const IndependentProb &probA);

// AND operator for finding P(A AND B)
IndependentProb operator&(const IndependentProb &probA,
                          const IndependentProb &probB);

// OR operator for finding P(A AND/OR B)
IndependentProb operator|(const IndependentProb &probA,
                          const IndependentProb &probB);

// EXCLUSIVE OR operator for finding P(A XOR B)
IndependentProb operator^(const IndependentProb &probA,
                          const IndependentProb &probB);

// EXCLUSIVE operator for finding P(A BUT NOT B)
IndependentProb operator-(const IndependentProb &probA,
                          const IndependentProb &probB);
```

```cpp
#include "independent-probability.hpp"

// Independent Probability object constructor; stores
// given probability.
IndependentProb::IndependentProb(double probability)
{
    if (probability >= 0.0 && probability <= 1.0)
    {
        this->probability = probability;
    }
    else
    {
        throw invalidProbEx;
    }
} // End constructor

// NOT operator for finding P(NOT A)
IndependentProb operator~(const IndependentProb &probA)
{
    return IndependentProb(1.0 - probA.getProbability());
}

// AND operator for finding P(A AND B)
IndependentProb operator&(const IndependentProb &probA, const IndependentProb &probB)
{
    return IndependentProb(probA.getProbability() * probB.getProbability());
}

// OR operator for finding P(A AND/OR B)
IndependentProb operator|(const IndependentProb &probA, const IndependentProb &probB)
{
    return IndependentProb(1.0 - ((~probA).getProbability() *
(~probB).getProbability()));
}

// EXCLUSIVE OR operator for finding P(A XOR B)
IndependentProb operator^(const IndependentProb &probA, const IndependentProb &probB)
{
    return IndependentProb((probA | probB).getProbability() - (probA &
probB).getProbability());
}

// EXCLUSIVE operator for finding P(A BUT NOT B)
IndependentProb operator-(const IndependentProb &probA, const IndependentProb &probB)
{
    return IndependentProb(probA.getProbability() * (~probB).getProbability());
}
```

```cpp
// @file
// @author Nathan Roe
//
// Accepts command line args for two double values
// (ints promoted to doubles) and calculates some
// probabilities based in the inputs. Alternately
// runs unit tests for the IndependentProb class

#include "IndependentProbability/independent-probability.hpp"
#include <iostream>

using namespace std;

// Run all probability operator calcs for given events A and B
void runProbabilities(const IndependentProb *a, const IndependentProb *b)
{
    // Run Probability Calcs
    cout << "P(~A) = " << (~*a).getProbability() << endl;
    cout << "P(~B) = " << (~*b).getProbability() << endl;
    cout << "P(A & B) = " << (*a & *b).getProbability() << endl;
    cout << "P(A | B) = " << (*a | *b).getProbability() << endl;
    cout << "P(A ^ B) = " << (*a ^ *b).getProbability() << endl;
    cout << "P(A - B) = " << (*a - *b).getProbability() << endl;
    cout << "P(B - A) = " << (*b - *a).getProbability() << endl;
}

// Function main begins probability evaluation loop
int main(int argc, char *argv[])
{
    // If Input args provided, run calcs on args
    if (argc == 3)
    {
        IndependentProb *a;
        IndependentProb *b;

        // Create IndependentProbability
        try
        {
            double x = stod(argv[1]);
            a = new IndependentProb(x);
            cout << "P(A) = " << a->getProbability() << endl;
        }
        // Catch inputs that can't be converted to numbers
        catch (const invalid_argument &e)
        {
            cout << e.what() << ": Could not interpret <" << argv[1] << "> as a
double." << endl;
            return EXIT_FAILURE;
        }
        // Catch invalid probabilities
        catch (const IndependentProb::invalidProbability &e)
        {
            cout << e.what() << ": " << argv[1] << endl;
            return EXIT_FAILURE;
        }

        // Create IndependentProbability
```

```
57          try
58          {
59              double y = stod(argv[2]);
60              b = new IndependentProb(y);
61              cout << "P(B) = " << b->getProbability() << endl;
62          }
63          // Catch inputs that can't be converted to numbers
64          catch (const invalid_argument &e)
65          {
66              cout << e.what() << ": Could not interpret <" << argv[2] << "> as a
    double." << endl;
67              return EXIT_FAILURE;
68          }
69          // Catch invalid probabilities
70          catch (const IndependentProb::invalidProbability &e)
71          {
72              cout << e.what() << ": " << argv[2] << endl;
73              return EXIT_FAILURE;
74          }
75          runProbabilities(a, b);
76      }
77      // If args are not provided, run default cases
78      else
79      {
80          cout << "Running Probability Tests\n" << endl;
81
82          int numTests = 8;
83          double probabilities[numTests][2] = {{0, 1},
84                                               {1, 0},
85                                               {0.25, 0.75},
86                                               {0.5, 0.5},
87                                               {0, 0},
88                                               {1, 1},
89                                               {1.1, 0.2},
90                                               {0.2, 1.1}};
91
92          // Run each of the probability tests
93          for (int idx = 0; idx < numTests; ++idx)
94          {
95              try
96              {
97                  IndependentProb *a = new IndependentProb(probabilities[idx][0]);
98                  IndependentProb *b = new IndependentProb(probabilities[idx][1]);
99                  cout << "P(A) = " << a->getProbability() << endl;
100                 cout << "P(B) = " << b->getProbability() << endl;
101                 runProbabilities(a, b);
102                 cout << "\n" << endl;
103             }
104             catch (const IndependentProb::invalidProbability &e)
105             {
106                 cout << e.what() << ": " << probabilities[idx][0]
107                      << "  " << probabilities[idx][1] << "\n" << endl;
108             }
109         }
110     }
111     return EXIT_SUCCESS;
112 } // End function main
```

```
Running Probability Tests

P(A) = 0
P(B) = 1
P(~A) = 1
P(~B) = 0
P(A & B) = 0
P(A | B) = 1
P(A ^ B) = 1
P(A - B) = 0
P(B - A) = 1


P(A) = 1
P(B) = 0
P(~A) = 0
P(~B) = 1
P(A & B) = 0
P(A | B) = 1
P(A ^ B) = 1
P(A - B) = 1
P(B - A) = 0


P(A) = 0.25
P(B) = 0.75
P(~A) = 0.75
P(~B) = 0.25
P(A & B) = 0.1875
P(A | B) = 0.8125
P(A ^ B) = 0.625
P(A - B) = 0.0625
P(B - A) = 0.5625


P(A) = 0.5
P(B) = 0.5
P(~A) = 0.5
P(~B) = 0.5
P(A & B) = 0.25
P(A | B) = 0.75
P(A ^ B) = 0.5
P(A - B) = 0.25
P(B - A) = 0.25


P(A) = 0
P(B) = 0
P(~A) = 1
P(~B) = 1
P(A & B) = 0
P(A | B) = 0
P(A ^ B) = 0
P(A - B) = 0
P(B - A) = 0


```

```
P(A) = 1
P(B) = 1
P(~A) = 0
P(~B) = 0
P(A & B) = 1
P(A | B) = 1
P(A ^ B) = 0
P(A - B) = 0
P(B - A) = 0


Probability not between 0.0 and 1.0: 1.1  0.2

Probability not between 0.0 and 1.0: 0.2  1.1
```