CS2013 - Programación III

Práctica Calificada #3 (PC3)

2023 - 2

Profesor: Rubén Rivas

Heap – 10 puntos

Una entidad financiera desea desarrollar un simulador (**investment_simulator**) con el fin de gestionar sus inversiones, la entidad invertira en instrumentos financieros (**instrument**) descritos a traves del capital (**equity**), tasa de retorno (**rate**) y una categoria (**category**) que es definida por el riesgo (B=bajo, M=medio y A=alto).

Se solicita desarrollar un simulador **investment_simulator** que ejecute **k** operaciones de forma consecutiva, de modo que en cada operación recupera el **capital** y reinvertira la **ganancia** que es equivalente al **capital** * **rate**, el simulador deberar retornar la suma de todas las recuperaciones obtenidas de las **k** operaciones y la suma total de todos los capitales luego de ejecutar las k operaciones.

Casos de uso #1

```
investment_simulator<double> i_simulator;
i_simulator.add(10, 0.5, 'A');
i_simulator.add(50, 0.5, 'A');
i_simulator.add(60, 0.5, 'M');
i_simulator.add(30, 0.5, 'B');
i_simulator.add(20, 0.5, 'B');
i_simulator.add(40, 0.5, 'A');
auto result = i_simulator(8);
cout << std::setprecision(2) << std::fixed
     << result.first << " " << result.second;
```

Casos de uso #2

```
investment_simulator<float> i_simulator;
i_simulator.add(10, 0.5, 'A');
i_simulator.add(50, 0.5, 'A');
i_simulator.add(60, 0.5, 'M');
i_simulator.add(30, 0.5, 'B');
i_simulator.add(20, 0.5, 'B');
i_simulator.add(40, 0.5, 'A');
i_simulator.add(70, 0.5, 'M');
auto result = i_simulator(8);
cout << std::setprecision(2) << std::fixed
     << result.first << " " << result.second;
```

Casos de uso #3

```
   investment_simulator<float> i_simulator;
   i_simulator.add(360, 0.25, 'A');
   auto result = i_simulator(12);
   cout << std::setprecision(2) << std::fixed
        << result.first  << " " << result.second;
```

Casos de uso #4

```
   investment_simulator<long double> i_simulator;
   i_simulator.add(36, 0.25, 'A');
   i_simulator.add(36, 0.25, 'A');
   i_simulator.add(36, 0.25, 'A');
   auto result = i_simulator(15);
   cout << std::setprecision(2) << std::fixed << result.first << " " <<
 result.second;
```

hash – 10 puntos

Dada una matriz de **N x N**, Inicialmente, con cada celda vacía. Y dada **K** actualizaciones de las celdas de la Matriz, donde cada actualización representada por (**r**,**c**) actualizara toda la fila **r** y toda la columna **c**.

Desarrollar una clase template (functor) **count_empty_cells** que permita calcular y retornar el número de celdas vacias en la matriz al finalizar las actualizaciones.

Tip: Utilizar 2 tablas hash, una para filas y otra para columnas

Caso de uso #1

```
    count_empty_cells<20> counter;
    counter.add(0,0);
    counter.add(19,19);
    std::cout << counter() << std::endl;
```

Caso de uso #2

```
    count_empty_cells<15> counter;
    counter.add(0,0);
    counter.add(1,0);
    counter.add(0,3);
    counter.add(4,3);
    counter.add(19,19);
    std::cout << counter() << std::endl;
```

Caso de uso #3

```
    count_empty_cells<15> counter {
       {0,  0},
       {1,  0},
       {0,  3},
       {4,  3},
       {11,  2},
       {5,  11},
       {19, 19},
    };
    std::cout << counter() << std::endl;
```

Caso de uso #4

```
   std::vector<pair<size_t, size_t>> v = {
       {0,  0},
       {1,  0},
       {0,  3},
       {4,  3},
       {11,  2},
       {5,  11},
       {19, 19},
   };
   count_empty_cells<45, std::vector> counter(v);
   std::cout << counter() << std::endl;
```

Barranco, 1 de diciembre 2023.