IMPLEMENTING THE POURING PUZZLE

Transforming a problem solution into a functional program

Delroy A. Brinkerhoff

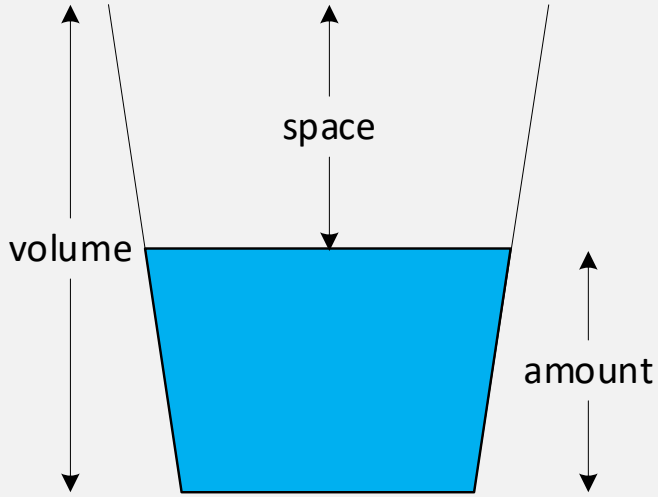
AUTHENTICALLY DEMONSTRATES

- `static` or class features
  - `static` class variable
  - `static` getter function
- Noncommutative binary member function
  - Commutative, a ☺ b = b ☺ a vs. noncommutative, a ☺ b ≠ b ☺ a
  - `a.pour(b)` ≠ `b.pour(a)`
  - Changes both arguments

Glass VARIABLES

```
class Glass
{
    private:
        static int pours;

        int volume;
        int amount;
};
```



Glass FUNCTIONS

```
class Glass
{
    public:
        Glass(int a_volume, int a_amount)
            : volume(a_volume), amount(a_amount) {}
        int getVolume() { return volume; }
        int getAmount() { return amount; }
        void display() { cout << amount << " / " << volume << endl; }
        static int getPours() { return pours; }

        void pour(Glass& source);
        //void pour(Glass* source);
};
```

POURING WATER FROM ONE GLASS TO ANOTHER

```
int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;

    int transfer = min(space, source.amount);
    amount += transfer;
    source.amount -= transfer;
}
```

INSTANTIATING THE `Glass` OBJECTS

```
int main()
{
    Glass    glasses[3] { Glass(3,0), Glass(5,0), Glass(8,8) };
    //Glass* glasses[] { new Glass(3,0), new Glass(5,0), new Glass(8,8) };

                                . . .

    return 0;
}
```

IMPLEMENTING THE GAME RULES: ENDING THE GAME

```
while (glasses[1].getAmount() != 4 && glasses[2].getAmount() != 4)
//while (glasses[1]->getAmount() != 4 && glasses[2]->getAmount() != 4)
{
                                    . . .
}
```

DISPLAYING THE CURRENT STATE

```
for (int i = 0; i < 3; i++)
{
    cout << "Glass " << i+1 << ": ";
    glasses[i].display();
    //glasses[i]->display();
}
```

CHOOSING A GLASS

```
int destination;

cout << "Pour TO glass: <1, 2, or 3; or enter 4 to quit>: ";
cin >> destination;

if (destination == 4)
     exit(0);
```

VALIDATING USER INPUT AND POURING THE WATER

```
if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
    glasses[destination - 1].pour(glasses[source - 1]);
    //glasses[destination - 1]->pour(glasses[source - 1]);
else
    cout << "0 < destination <= 3 AND 0 < source <= 3" << endl;
```

PRINTING THE "SCORE:" CALLING A STATIC FUNCTION

```
cout << "\n\nYou solved the puzzle in " <<
    Glass::getPours() << " pours" << endl;
```