safemath.sol

```
pragma solidity ^0.4.25;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is
greater than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

/**
 * @title SafeMath32
 * @dev SafeMath library implemented for uint32
 */
library SafeMath32 {

  function mul(uint32 a, uint32 b) internal pure returns (uint32) {
    if (a == 0) {
      return 0;
    }
    uint32 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint32 a, uint32 b) internal pure returns (uint32) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint32 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint32 a, uint32 b) internal pure returns (uint32) {
    assert(b <= a);
    return a - b;
  }

  function add(uint32 a, uint32 b) internal pure returns (uint32) {
    uint32 c = a + b;
    assert(c >= a);
    return c;
  }
}

/**
 * @title SafeMath16
 * @dev SafeMath library implemented for uint16
 */
library SafeMath16 {
  function mul(uint16 a, uint16 b) internal pure returns (uint16) {
    if (a == 0) {
      return 0;
    }
    uint16 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint16 a, uint16 b) internal pure returns (uint16) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint16 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint16 a, uint16 b) internal pure returns (uint16) {
    assert(b <= a);
    return a - b;
  }

  function add(uint16 a, uint16 b) internal pure returns (uint16) {
    uint16 c = a + b;
    assert(c >= a);
    return c;
  }
}
```