pragma solidity ^0.5.0;

/\*\*

\* @dev Wrappers over Solidity's arithmetic operations with added overflow

\* checks.

\*

\* Arithmetic operations in Solidity wrap on overflow. This can easily result

\* in bugs, because programmers usually assume that an overflow raises an

\* error, which is the standard behavior in high level programming languages.

\* `SafeMath` restores this intuition by reverting the transaction when an

\* operation overflows.

\*

\* Using this library instead of the unchecked operations eliminates an entire

\* class of bugs, so it's recommended to use it always.

\*/

library SafeMath {

/\*\*

\* @dev Returns the addition of two unsigned integers, reverting on

\* overflow.

\*

\* Counterpart to Solidity's `+` operator.

\*

\* Requirements:

\* - Addition cannot overflow.

\*/

function add(uint256 a, uint256 b) internal pure returns (uint256) {

uint256 c = a + b;

require(c >= a, "SafeMath: addition overflow");

return c;

}

/\*\*

\* @dev Returns the subtraction of two unsigned integers, reverting on

\* overflow (when the result is negative).

\*

\* Counterpart to Solidity's `-` operator.

\*

\* Requirements:

\* - Subtraction cannot overflow.

\*/

function sub(uint256 a, uint256 b) internal pure returns (uint256) {

require(b <= a, "SafeMath: subtraction overflow");

uint256 c = a - b;

return c;

}

/\*\*

\* @dev Returns the multiplication of two unsigned integers, reverting on

\* overflow.

\*

\* Counterpart to Solidity's `\*` operator.

\*

\* Requirements:

\* - Multiplication cannot overflow.

\*/

function mul(uint256 a, uint256 b) internal pure returns (uint256) {

// Gas optimization: this is cheaper than requiring 'a' not being zero, but the

// benefit is lost if 'b' is also tested.

// See: <https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522>

if (a == 0) {

return 0;

}

uint256 c = a \* b;

require(c / a == b, "SafeMath: multiplication overflow");

return c;

}

/\*\*

\* @dev Returns the integer division of two unsigned integers. Reverts on

\* division by zero. The result is rounded towards zero.

\*

\* Counterpart to Solidity's `/` operator. Note: this function uses a

\* `revert` opcode (which leaves remaining gas untouched) while Solidity

\* uses an invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\* - The divisor cannot be zero.

\*/

function div(uint256 a, uint256 b) internal pure returns (uint256) {

// Solidity only automatically asserts when dividing by 0

require(b > 0, "SafeMath: division by zero");

uint256 c = a / b;

// assert(a == b \* c + a % b); // There is no case in which this doesn't hold

return c;

}

/\*\*

\* @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),

\* Reverts when dividing by zero.

\*

\* Counterpart to Solidity's `%` operator. This function uses a `revert`

\* opcode (which leaves remaining gas untouched) while Solidity uses an

\* invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\* - The divisor cannot be zero.

\*/

function mod(uint256 a, uint256 b) internal pure returns (uint256) {

require(b != 0, "SafeMath: modulo by zero");

return a % b;

}

}