// SPDX-License-Identifier: MIT pragma solidity ^0.6.2; 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) { return sub(a, b, "SafeMath: subtraction overflow"); } /\*\* \* @dev Returns the subtraction of two unsigned integers, reverting with custom message on \* overflow (when the result is negative). \* \* Counterpart to Solidity's `-` operator. \* \* Requirements: \* \* - Subtraction cannot overflow. \*/ function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) { require(b <= a, errorMessage); 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) { return div(a, b, "SafeMath: division by zero"); } /\*\* \* @dev Returns the integer division of two unsigned integers. Reverts with custom message 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, string memory errorMessage) internal pure returns (uint256) { require(b > 0, errorMessage); 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) { return mod(a, b, "SafeMath: modulo by zero"); } /\*\* \* @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo), \* Reverts with custom message 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, string memory errorMessage) internal pure returns (uint256) { require(b != 0, errorMessage); return a % b; } }