بسم الله الرحمن الرحیم

درس: بلاکچین

استاد: محمدعلی مداح علی

دانشجو: امیرحسین رستمی

شماره دانشجویی: 96101635

تمرین: گزارش تمرین عملی سری 3

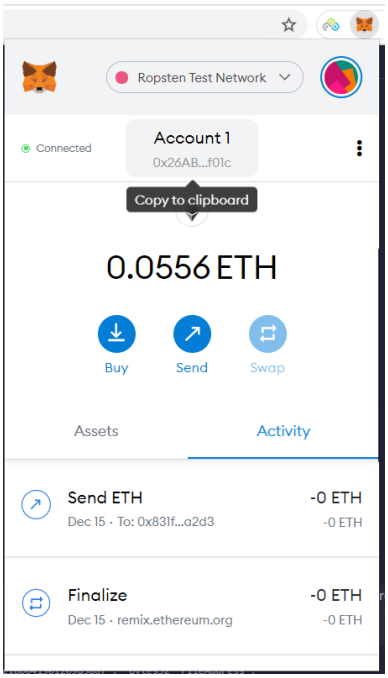
سوال اول:

به ترتیب زیر اعمال را انجام می دهیم:

1- ابتدا در Meta Mask یک کیف پول می سازیم.
   a. آدرس عمومی کیف پول بنده عبارت است از:

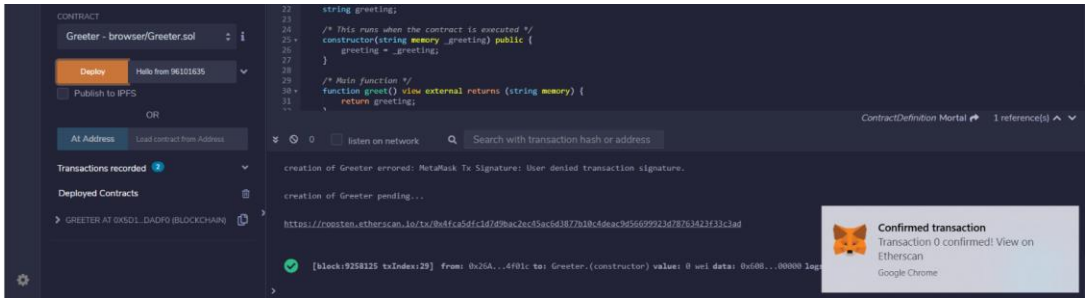
Etherium Account: 0x26AB4636Ea7945A074E98A36a2CEF5AC9684f01c

مشخصات اکانت بنده در Meta Mask :

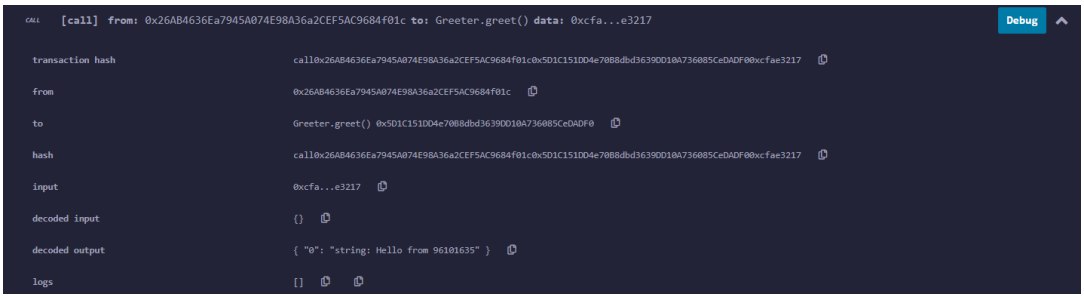


2- سپس به کمک Free Etherium providers مقداری اتریوم به حسابم واریز کردم.
   a. از این لینک جهت دریافت استفاده کردم.
   b. از لینک فوق 0.5 اتریوم می توان روزانه برداشت زد.
3- حال کد ضمیمه شده در سوال اول را داخل remix آپلود می کنیم.
4- کد را compile می کنیم.
5- کد کامپایل شده را در شبکه Ropsten Test Network ، deploy می کنیم.

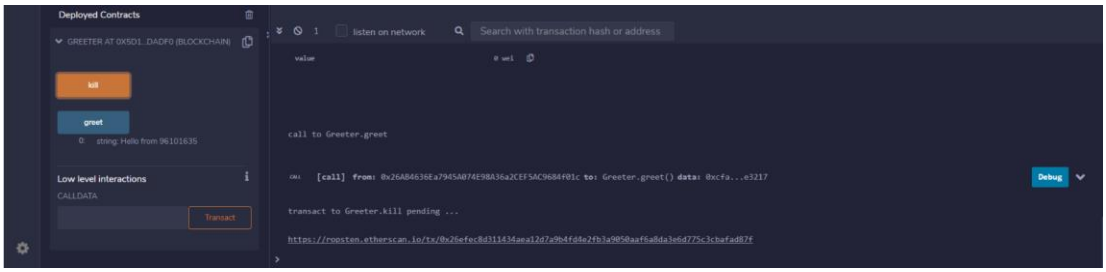
جهت مشاهده جزییاتِ تراکنشِ deploy شده به زیر مراجعه کنید.



6- حال تابع greet را call می کنیم و نتیجه به شرح زیر گردید:



7- در نهایت تابع kill را فرامی خوانیم و نتیجه به شرح زیر گردید:(جزییات تراکنش)



سوال دوم:

قطعه قطعه کد ضمیمه شده را آورده و توضیح می دهیم:

```
// who owns how many tokens
mapping(address => uint256) balances;
// account "A" allows account "B" to extract "X" amount
mapping(address => mapping(address => uint256)) internal allowed;
// total amount of money.
uint256 _totalSupply;
```

متغیرِ balances یک mapping است از اکانت ها به محتوای اتریومِ هرکدام.
متغیرِ allowed یک mapping است از اکانت ها به یک mapping دیگر که خود یک mapping است از اکانت های اجازه دار به مبلغ مجاز استفاده شان.
متغیر totalSupply_ بیانگر برآیند پول هاست.

```
constructor(uint256 total) public {
    _totalSupply = total;
    balances[msg.sender] = _totalSupply;
}

function totalSupply() public view returns (uint256) {
    return _totalSupply;
}

function balanceOf(address who) public view returns (uint256) {
    return balances[who];
}
```

Constructor یک مقدار total می گیرد که به معنی برآیند پول کل حساب هاست و در ابتدا تمامی total در حساب ایجاد کننده ی contract است.

تابع totalSupply خیلی ساده است و پولِ کل را(متغیر totalSupply_) را برمی گرداند.تابعِ balanceOf هم از آرایه balances خانه who را برمی گرداند.

```
function transfer(address to, uint256 value) public returns (bool) {
    // check to see if there is enough amount of money in sender's account.
    require(value <= balances[msg.sender]);
    // withdraw from sender's account
    balances[msg.sender] = balances[msg.sender].sub(value);
    // deposit to receiver's account
    balances[to] = balances[to].add(value);
    // broadcast event.
    emit Transfer(msg.sender, to, value);
    return true;
}

function allowance(address owner, address spender) public view returns (uint256) {
    return allowed[owner][spender];
}
```

تابعِ Transfer چنین عمل می کند:

1- ابتدا لازم می دارد که مقدار ارسالی(value) از محتوایِ اکانت کمترمساوی باشد.
2- از حساب ارسال کننده کم می کند.
3- به حساب دریافت کننده واریز می کند.

4- اعلامِ واقعه Transfer می کند.
5- و مقدار True را در صورت انجام همه مراحل قبل باز میگرداند.

عملکردِ تابع allowance هم واضح است و مقدار را ست می کند.

نکته:در solidity اگر محتوایِ یک خانه Boolean مقداردهی اولیه نشده باشد،این مقدار false درنظرگرفته می شود و نیز محتوایِ هر متغیر عددی نیز در صورت عدم مقداردهی اولیه برابر صفر خواهد بود.

```
function transferFrom(address from, address to, uint256 value) public returns (bool) {
    // check to see if there is enough amount of money in sender's account.
    require(value <= balances[from]);
    require(value <= allowed[from][msg.sender]);

    // withdraw
    balances[from] = balances[from].sub(value);
    allowed[from][msg.sender] = allowed[from][msg.sender].sub(value);
    // deposit
    balances[to] = balances[to].add(value);
    emit Transfer(from, to, value);
    return true;
}

function approve(address spender, uint256 value) public returns (bool) {
    allowed[msg.sender][spender] = value;
    emit Approval(msg.sender, spender, value);
    return true;
}
```

تابع transferFrom چنین عمل می کند:

1- الزامات:
   a. لازم می دارد که مقدارِ value از محتوای اکانت from کمتر مساوی باشد.
   b. هم چنین لازم می دارد که مقدار value از میزانِ قابل دسترسیِ ارسال کننده پیام از اکانتِ from نیز کمتر مساوی باشد.
2- از حساب from کم می کنیم.
3- از میزان قابل دستکاریِ ارسال کننده پیام از حساب from نیز کسر وجه می زنیم.
4- به حساب مقصد واریز می کنیم.
5- اعلامِ واقعه Transfer می کنیم.
6- و مقدار True را در صورت انجام همه مراحل قبل باز میگرداند.

تابع approve چنین عمل می کند:

1- اجازه دسترسی به میزان Value از حساب ارسال کننده پیغام به نفرِ spender می دهد.
2- اعلامِ واقعه Approval می کند.

سوال سوم:

قطعه قطعه کد ضمیمه شده را آورده و توضیح میدهم:

```
using SafeMath for uint256;
using AddressUtils for address;

bytes4 constant ERC721_RECEIVED = 0xf0b9e5ba;

mapping(uint256 => address) internal tokenOwner;
mapping (address => uint256) internal ownedTokensCount;
mapping (uint256 => address) internal tokenApprovals;

// address "A" allows address "B" to operate all A's assets
mapping (address => mapping (address => bool)) internal operatorApprovals;
```

در دو خط ابتدا اعلام می داریم که برای عملیات ریاضی روی uint256 از کلاسِ SafeMath استفاده کند و نیز در خط دوم نیز می گوییم که برای عملیات روی address از AddressUtils استفاده کند(این کلاس یک تابع isContract دارد که با فراخوانی اش روی یک آدرس با بررسی بزرگتر از صفر بودنِ آن اعلام می دارد که آیا این آدرس یک Contract است یا خیر).

متغیرِ tokenOwner یک tokenId میگیرد و مشخص می کند که مالک آن چه کسی است.
متغیرِ ownedTokensCount آدرس مالک میگیرد و مشخص می کند که این مالک چه تعداد tokenId دارد.
متغیرِ tokenApprovals مشخص می کند که یک tokenId به چه کسی Approve شده است.
متغیرِ operatorApprovals یک mapping است از اکانت ها به یک mapping دیگر که خود یک mapping است از اکانت های به مجاز بودنشان(جهت مصرف کل دارایِ اکانت اولیه) است.همانطور که میدانید وقتی یک متغیر از جنس Boolean در solidity مقدار دهی اولیه نشود،مقدار false به خود میگیرد.

```
modifier onlyOwnerOf(uint256 _tokenId) {
    require(ownerOf(_tokenId) == msg.sender);
    _;
}

modifier canTransfer(uint256 _tokenId) {
    require(isApprovedOrOwner(msg.sender, _tokenId));
    _;
}

function balanceOf(address _owner) public view returns (uint256) {
    require(_owner != address(0));
    return ownedTokensCount[_owner];
}

function ownerOf(uint256 _tokenId) public view returns (address) {
    address owner = tokenOwner[_tokenId];
    require(owner != address(0));
    return owner;
}

function exists(uint256 _tokenId) public view returns (bool) {
    address owner = tokenOwner[_tokenId];
    return owner != address(0);
}
```

Modifierِ اول،همانطور که از نامش پیداست چک می کند که دارنده ی این tokenId همان ارسال کننده پیغام است یا خیر که این چک به سادگی به صورت گفته شده انجام میگیرد.

پیش از پرداختن به Modifier ِ دوم ابتدا سه تابع دیگر را توضیح میدهیم.

تابعِ balanceOf ابتدا چک می کند که آدرس متقاضی صفر نباشد(یعنی این متغیر مقدار دهی اولیه شده باشد) و سپس محتوایِ tokenId آن را برمیگرداند.

تابعِ ownerOf ابتدا مقدارِ صاحبِ tokenId داده شده را استخراج می کند و سپس لازم می دارد که این مقدار address(0) نباشد و درصورت ناصفر بودن آدرس صاحب را برمیگرداند.(همانطورکه می دانید اگر متغیری از جنس آدرس اگر مقداردهی اولیه نشود در solidity برابر Address(0) خواهد بود).

تابعِ exists که جهت بررسی وجودِ tokenId است چنین عمل می کند:
ابتدا صاحب آن را استخراج می کند،سپس چک می کند که آیا این آدرس وجود دارد(یعنی در آرایه مقداردهی اولیه شده است) یا خیر و این چک از طریق بررسیِ برابر با address(0) انجام میگیرد.

حال برویم سراغِ Modifier دوم: این modifier چک می کند که ارسال کننده درخواست آیا قابلیت بهره برداری از tokenId ذکرشده را دارد یا خیر.این modifier از تابعی استفاده می کند که لازم است ابتدا پیاده سازیِ آن را بیاوریم:

```
function isApprovedOrOwner(address _spender,uint256 _tokenId) internal view returns (bool){
    address owner = ownerOf(_tokenId);
    return (_spender == owner || getApproved(_tokenId) == _spender || isApprovedForAll(owner, _spender));
}
```

این تابع همانطورکه از نامش پیداست بررسی می کند که آیا spender توانایی استفاده از tokenId را دارد و این توانایی استفاده به صورت حالت های زیر قابل انجام است:

1- Spender صاحب tokenId باشد.
2- Spender اجازه استفاده از tokenId به خاطرِ تخصیص Approve داشته باشد.
3- Spender اجازه مصرف تمامی دارایی های صاحب اصلی tokenId را داشته باشد.

هر کدام از سه شرط فوق به صورت logical OR در انتهای بررسی و نتیجه حاصله بازگردانی می شود و modifier هم صرفا لازم میدارد که خروجی این تابع برای tokenId داده شده و شخصِ درخواست دهنده True باشد.

```
function approve(address _to, uint256 _tokenId) public {
    address owner = ownerOf(_tokenId);
    require(_to != owner);
    require(msg.sender == owner || isApprovedForAll(owner, msg.sender));

    if (_to != address(0)) { /* getApproved(_tokenId) == address(0)  || */
      tokenApprovals[_tokenId] = _to;
      emit Approval(owner, _to, _tokenId);
    }
}

function getApproved(uint256 _tokenId) public view returns (address) {
    return tokenApprovals[_tokenId];
}

function setApprovalForAll(address _to, bool _approved) public {
    require(_to != msg.sender);
    operatorApprovals[msg.sender][_to] = _approved;
    emit ApprovalForAll(msg.sender, _to, _approved);
}
```

تابع Approve دارای عملکردی مشابه سوال قبل است و این چنین عمل می کند:

1- ابتدا صاحب tokenId را استخراج می کند و لازم می دارد که با مقصد یکی نباشد(چون این کار عبث بوده و بی معنی است).
2- لازم می دارد که انجام دهنده درخواست یا صاحب اصلی tokenId باشد یا اجازه مصرف دارایِ صاحبِ tokenId را داشته باشد.
3- چک می کند که to_ مخالف آدرس صفر باشد.(توجه کنید که تابع Approve قابلیت override کردن شخصِ Approve شده را دارد،اگر می خواد این امکان وجود نداشته باشد کافیست شرطی که comment شده و جلو if قرار گرفته است را به داخل if انتقال داده و از comment خارج کنید).
4- در صورت برقرار بودن موارد ذکر شده، Approval انجام میگیرد.
5- اعلامِ واقعه Approval می کند.

تابع getApproved به سادگی شخص Approved شده برای tokenId داده شده برگردانده می شود.

تابع setApprovalForAll هم به سادگی پس از احراز عدم برابریِ to_ و درخواست دهنده،اپراتوریِ Approval را تنظیم می کند و در نهایت اعلامِ واقعه ApprovalForAll می کند.

```
function clearApproval(address _owner, uint256 _tokenId) internal {
    require(ownerOf(_tokenId) == _owner);
    if (tokenApprovals[_tokenId] != address(0)) {
      tokenApprovals[_tokenId] = address(0);
      emit Approval(_owner, address(0), _tokenId);
    }
}

function addTokenTo(address _to, uint256 _tokenId) internal {
    require(tokenOwner[_tokenId] == address(0));
    tokenOwner[_tokenId] = _to;
    ownedTokensCount[_to] = ownedTokensCount[_to].add(1);
}

function removeTokenFrom(address _from, uint256 _tokenId) internal {
    require(ownerOf(_tokenId) == _from);
    ownedTokensCount[_from] = ownedTokensCount[_from].sub(1);
    tokenOwner[_tokenId] = address(0);
}
```

تابع clearApproval چنین عمل می کند:

1- ابتدا لازم میدارد که صاحب این tokenId همین شخص owner باشد.
2- دستور if چک می کند که این tokenId قبلا به کسی approve شده باشد.
   a. اگر قبلا به کسی داده شده باشد این Approval را پاک می کند.
   b. اگر قبلا به کسی داده نشده باشد کار انجام شده است ☺.

تابع addTokenTo چنین عمل می کند:

1- ابتدا لازم میدارد که این tokenId بی صاحب باشد.
2- صاحبِ این tokenId را برابر _to قرار می دهد.
3- به تعداد دارایی هایِ توکنِ _to یک عدد اضافه می کند.

تابع removeTokenFrom چنین عمل می کند:

1- ابتدا لازم میدارد که صاحب tokenId همان _from باشد.
2- از تعداد دارایی هایِ توکنِ _from یک واحد کم می کند.
3- صاحب tokenId را برابر address(0) می کند به اصطلاح tokenId را بی صاحب می کند☺.

```
function isApprovedForAll(address _owner,address _operator) public view returns (bool) {
    return operatorApprovals[_owner][_operator];
}

function transferFrom(address _from,address _to,uint256 _tokenId) public canTransfer(_tokenId){

    require(isApprovedOrOwner(msg.sender, _tokenId)); //"ERC721: transfer caller is not owner nor approved");
    require(_from != address(0));
    require(_to != address(0));

    clearApproval(_from, _tokenId);
    removeTokenFrom(_from, _tokenId);
    addTokenTo(_to, _tokenId);

    emit Transfer(_from, _to, _tokenId);
}

function safeTransferFrom(address _from,address _to,uint256 _tokenId) public canTransfer(_tokenId){
    safeTransferFrom(_from, _to, _tokenId, "");
}
```

تابع isApprovedForAll، واجدیت را استخراج کرده و بازمیگرداند(توجه کنید که همانطور که پیشتر گفتیم،یک خانه Boolean اگر مقداردهی نشده باشد،مقدار false را برمیگرداند).

تابع transferFrom چنین عمل می کند:

1- ابتدا لازم می دارد که شخص درخواست دهنده اجازه انجام تراکنش داشته باشد(یا Approved باشد یا owner)
2- سپس لازم می دارد که _from و _to ناصفر باشند(مشابه همان دلایلی که پیشتر ذکر شده).
3- صاحبانِ واسطه ایِ توکن را(درصورت وجود)حذف می کند(کسانی که Approve شده اند که مصرف کنند).
4- توکن مصرفی را از لیست ممالکِ _from (صاحب اصلی) حذف می کند.
5- توکن مصرفی را به ممالک _to اضافه می کند.
6- اعلامِ واقعه Transfer می کند.

آن نسخه ای از تابع safeTransferFrom که data را نمی گیرد همانند فراخوانیِ نسخه ای است که data را میگیرد منتها با مقدارِ "" = data.

پیش از پرداختن به ادامه کد،ابتدا لازم است تا نکته زیر را ذکر کنیم:

When using the `safeTransferFrom` function to send ERC721 tokens to a contract address, it will fail unless the receiving contract properly implements the `ERC721TokenReceiver` interface. (See the ERC721 Standard for details).

Any implementation of ERC721TokenReceiver will have the `onERC721Received` function and will return `bytes4(keccak256("onERC721Received(address,uint256,bytes)"))`.

همانطور که می دانید خروجی تابعِ ذکر شده به ازای هر بارفراخوانی اش یک مقدار ثابت است و لذا یک optimization انجام می دهیم(جهت کاهش مصرف gas) و مقدار آن را محاسبه کرده و به صورت یک متغیر داخل ERC721Receiver قرار می دهیم.(این مقدار برابرِ 0xf0b9e5ba است).لذا داریم که پیاده سازیِ ERC721Receiver به شکل زیر می گردد:

```
contract ERC721Receiver {

    bytes4 constant ERC721_RECEIVED = 0xf0b9e5ba;
    function onERC721Received(address _from, uint256 _tokenId, bytes memory _data) public returns(bytes4);
}
```

حال قطعه کد زیر را درنظر بگیرید:

```
function checkAndCallSafeTransfer(address _from,address _to,uint256 _tokenId,bytes memory _data) internal returns (bool){
    if (!_to.isContract()) {
      return true;
    }
    bytes4 retval = ERC721Receiver(_to).onERC721Received(_from, _tokenId, _data);
    return (retval == ERC721_RECEIVED);
}
```

این تابع ابتدا چک می کند که _to یک contract نباشد و اگر نباشد true بازمیگرداند،حال اگر contract باشد چک می کند که پروتکل ذکر شده را رعایت کرده باشد یعنی returnValue با ERC721_RECEIVED برابر باشد. (مقدار ERC721_RECEIVED همان 0xf0b9e5ba است که جهت optimization محاسبه شده است).

```
function safeTransferFrom(address _from,address _to,uint256 _tokenId,bytes memory _data) public canTransfer(_tokenId){
    transferFrom(_from, _to, _tokenId);
    require(checkAndCallSafeTransfer(_from, _to, _tokenId, _data));
}
```

هرکدام از دو تابع استفاده شده داخل تابع فوق پیشتر معرفی و تشریح شده اند و لذا عملکرد تابع فوق واضح است.

سوال چهارم:

ابتدا Auction.sol را توضیح می دهیم:

```
contract Auction {

    struct Description {

        address payable admin;
        uint startBlock; // considered as the block.number during construction.
        address payable winnerAddress;
        uint winnerBid;
    }

    Description public description;

    modifier onlyAdmin() {
        require(msg.sender == description.admin);
        _;
    }

    event auctionStarted();
    event auctionFinished(address winnerAddress, uint winnerBid);

    function activateAuction() public;
    function finalize() public;

}
```

Contract ِ حراج،یک متغیری دارد به نام description که حاوی اطلاعات زیر است:

1- مالکِ حراج(admin)
2- شماره بلاک حینِ هنگامِ ساختِ Auction.
3- آدرس برنده
4- مقدار پیشنهادی برنده.

Auction یک modifier داردکه به سادگی چک می کند که آیا فراخواننده همان admin است یا خیر.

Auction دو عدد رخداد دارد که یکی شروع شدنِ حراج را اعلام می کندو دیگری اتمام آن را اعلام می دارد.

دو تابع دارد که یکی فعال سازِ حراج بوده و دیگری تمام کننده حراج است.

حال می رویم سراغ Bid.sol:

```
contract Bid {
    /// @notice This function generates the nonce and the hash needed in the Vickrey Auction.
    /// @param _bidValue is the desired bid.
    function generate(uint _bidValue) public view returns(uint, bytes32, bytes32) {

        uint value = _bidValue;
        bytes32 nonce = keccak256(abi.encodePacked(block.timestamp));
        bytes32 calculatedHash;
        calculatedHash = keccak256(abi.encodePacked(value,nonce));
        return (value, nonce, calculatedHash);
    }
}
```

تابع generate یک مقدارِ پیشنهادی میگیرد و چنین عمل می کند:

1- ابتدا یک nonce ِ که برابر است با hash گرفتن از block.timestamp ،محاسبه می کند.

توجه کنید که این nonce با توجه به نحوه محاسبه اش حالت رندوم دارد.
هم چنین توجه کنید که به جای استفاده از SHA-3 از نسخه به روز شده ی آن که keccake256 است استفاده میکنیم.(SHA-3 یک تابعِ deprecated است).

2- مقدار value را کنار nonce گذاشته(الحاق می کنیم(concat)) و سپس از حاصل hash میگیریم و در اصل ارایه دهنده در مقامِ commit این مقدار را در حراج اعلام میکند(با توجه به نحوه محاسبه آن نمی توان به value دسترسی پیدا کرد و به سادگی با دادن nonce و value هم می توان مالکِ آن بودن را احراز کرد).

حال برویم سراغ CustomAuction.sol:

```
contract CustomAuction is Auction {

    enum Phase { Pending, Commitment, Opening, Finished }

    Phase public phase;

    uint startPhaseBlock;
    uint startOpeningBlock;

    uint commitment_len;
    uint opening_len;

    address payable lowestBidder;
    uint lowestBid;
    uint ReserveFund;

    bool firstOpen = true;

    struct Bid {
        bytes32 FileAddress;
        uint value;
        bytes32 calculatedHash;
        bytes32 nonce;
        bool exists;
    }

    mapping(address => Bid) bids;

    // address of accounts admin choose in Opening phase
    mapping(address => bool) chooses;
```

یک enum داریم به نام Phase که جهت بیانِ فاز های مختلف حراج بوده و متغیر phase را داریم که برابر فازِ حراج بوده که چهار حالت مختلف زیر را می تواند داشته باشد:

1- Pending
2- Commitment
3- Opening
4- Finished

متغیر startPhaseBlock را داریم که بیانگر شماره بلاک در هنگامی است که activateAuction() فراخوانی می شود.

متغیر startOpeningBlock را داریم که بیانگر شماره بلاک در هنگامی است که startOpening() فراخوانی شود.

متغیرهای commitment_len و opening_len به ترتیب بیانگر طول دوره commit و دوره open اند.

LowestBidder بیانگر آدرس برنده است که طبیعتا کمترین پیشنهاد را دارد و متغیر lowestBid برابر است با value متناظر با lowestBidder.

متغیر ReserveFund برابر است با مبلغ رزرویِ صاحب حراج برای حراج است.

متغیر firstOpen جهت تعیین اولین مرحله فراخوانی تابع startOpening() است.

از struct bid جهت اعلام bid ها به حراج استفاده می شود که 5 فیلد دارد:

1- آدرس فایل آپلودی(جهت بررسی رزومه)
2- مبلغ پیشنهادی
3- مقدار Hash اعلامی
4- مقدار nonce متنظار با hash بالا و مبلغ پیشنهادی
5- یک فیلد Boolean جهت تشخیص مقداردهی شدنِ bid.
   a. همانطور که می دانید مقادیر Boolean در هنگامی که مقدار دهی اولیه در solidity نشده باشند برابر false درنظرگرفته می شوند.

Bids یک mapping از آدرس ها به bid های مدعی شان است و chooses یک mapping از آدرس ها به بودنشان در دسته انتخاب مالک حراج پس از مرحله commit است.

```
event openingStarted();

constructor(address payable _admin,uint _commitment_len,uint _opening_len) public payable {

    require( _admin > address(0) );
    // adding thresholding.
    commitment_len = _commitment_len;
    opening_len = _opening_len;
    phase = Phase.Pending;

    description.startBlock = block.number;

    description.admin = _admin;
    ReserveFund = msg.value;

}

/// @dev This modifier allow to invoke the function olny during the Commitment phase.
modifier duringCommitment {
    require(phase == Phase.Commitment);
    _;
}

/// @dev This modifier allow to invoke the function olny during the Opening phase.
modifier duringOpening {
    require(phase == Phase.Opening);
    _;
}

function getReserveFund() public view returns (uint256) {
    return ReserveFund;
}
```

رخدادِ openingStarted جهت ابلاغ شروع شدن فازِ opening می باشد.

در constructor حراج به ترتیب موارد زیر انجام می شود:

1- چک کردن اینکه آدرس admin ناصفر باشد.(به همان دلیل همیشگی ☺)
2- تنظیم کردن طول دوره های commitment و opening.
3- تنظیم کردن فازِ حراج برابر با pending(چون هنوز start نشده است).
4- تعیین کردن description.startBlock برابر با شماره بلاک در این مرحله.
5- تنظیم کردن admin حراج.
6- تنظیم کردن reserveFund از روی msg.value.واضحا مبلغ پرداختی admin حین ساخت حراج به عنوان reserverFund درنظرگرفته می گردد.

Modifier های duringX احراز می کند که آیا فاز حراج برابر باX است یا خیر و به سادگی با یک require انجام می گیرد.

تابع getReservedFund() مقدار reserveFund را برمیگرداند.

```
function getFile( address add ) public view returns ( bytes32 fileAdd) {
    // check if there is a bid with this address
    require(bids[add].exists);
    return bids[add].FileAddress;
}

/// @notice This function will activate the auction.
function activateAuction() public onlyAdmin {
    require(phase == Phase.Pending);
    phase = Phase.Commitment;
    startPhaseBlock = block.number;
    emit auctionStarted();
}
```

تابع getFile، آدرس فایل را در صورت وجود داشتن bid ای با آدرسِ صاحب داده شده برمیگرداند.

تابع activateAuction چنین عمل می کند:

1- ابتدا چک می کند که آیا حراج در مرحله pending قرار دارد یا نه.(جهت پرهیز از فراخوانیِ آن در زمان های دیگر).
2- سپس فاز حراج را با Commitment قرار می دهد.
3- مقدار startPhaseBlock را برابر با شماره بلاک قرار می دهد.
4- رخداد واقعه auctionStarted را اعلام می دارد.

```
function bid(bytes32 _bidHash, bytes32 _FileAddress) public duringCommitment payable {

    require( (block.number - startPhaseBlock) <= commitment_len );

    // The bidder should not be able to send another bid.
    //require(!bids[msg.sender].exists);

    Bid memory bid_create = Bid(
        _FileAddress,
        0,
        _bidHash,
        0,
        true
    );

    bids[msg.sender] = bid_create;

}
```

تابع فوق جهت ذخیره bid های اعلام شده توسط شرکت کنندگان در حراج است و چنین عمل می کند:

1- ابتدا چک می کند که مجاز است یا خیر.
2- یک عدد bid با آدرس فایل و Hash داده شده می سازد.
3- Bid ساخته شده را در mapping عه bids تزریق میکند.

```
///@notice This function activate the Opening phase
function startOpening(address add1, address add2, address add3) public onlyAdmin {

    require(firstOpen == true);
    require(bids[add1].exists);
    require(bids[add2].exists);
    require(bids[add3].exists);

    firstOpen = false;

    chooses[add1] = true;
    chooses[add2] = true;
    chooses[add3] = true;

    startOpeningBlock = block.number;
    phase = Phase.Opening;

    emit openingStarted();

}
```

تابع فوق جهت فعال سازی فازِ opening بوده و چنین عمل می کند:

1- ابتدا چک می کند که پیشتر وارد این عمل را انجام نداده باشد.
2- لازم می دارد که bid هایی با آدرس های داده شده وجود داشته باشد.
   a. اگر آدرس ها اشتباه داده باشد این امکان را به صاحب حراج می دهد که مجدد آدرس بدهد(دقت کنید که مقدار firstOpen را پس از صحیح بودن سه آدرس داده شده به false تغییر می دهیم).
3- مقدار متغیر firstOpen را برابر false قرار میدهد تا از ورود مجدد به این تابع جلوگیری کند.
4- در آرایه chooses مقدار سه آدرس فوق را true می کند.
5- startOpeningBlock را برابر با شماره بلاک قرار میدهد.
6- فاز حراج را Open تنظیم می کند.
7- رخداد واقعه openingStarted را اعلام می کند.

```
function open( uint value, bytes32 _nonce) public duringOpening {

    if((block.number - startOpeningBlock) <= opening_len ){
        // is this fund possible :)?
        require(value <= ReserveFund);

        // Control the correctness of the bid
        Bid memory bid_open = bids[msg.sender];
        bytes32 thisNonceHash =  keccak256(abi.encodePacked(value,_nonce));
        require(thisNonceHash == bid_open.calculatedHash );

        // Update the bid status
        bid_open.value = value;
        bid_open.nonce = _nonce;

        // existance among chooses
        if(/* existance check */ chooses[msg.sender] ){
            if( lowestBid!= 0 ){
                if(lowestBid >= value){
                    lowestBid = value;
                    lowestBidder = msg.sender;
                }
            }else{
                lowestBid = value;
                lowestBidder = msg.sender;
            }
        }

    }else{
            // semi auto finalize :) .
            phase = Phase.Finished;
            finalize();
    }

}
```

تابع open جهت دادن امکان به شرکت کننده ها جهت احراز و بازگشایی bid هایشان است و چنین عمل می کند:

1- ابتدا چک می کند که مهلت فازِ opening تموم نشده باشد و اگر:
   a. تمام شده باشد:
      i. فاز حراج را برابر Finished تنظیم می کند.
      ii. تابع finalize را فرا می خواند.
   b. تمام نشده باشد:
      i. چک می کند که مقدار value کمترمساویِ حداکثر پول قابل پرداخت توسط صاحب حراج باشد.
      ii. Bid ادعا شده را استخراج می کند.
      iii. چک می کند که آیا bid استخراج شده با nonce و value داده شده تطابق دارد یا خیر.
      iv. در صورت تطابق مقادیر value و nonce را در bid احراز شده قرار می دهد.
      v. سپس بررسی می کند که آیا این bid در میان انتخاب شده هایِ صاحب حراج قرار دارد.
      vi. در صورت قرار داشتن در میان chooses چنین عمل می کند:
         1. اگر lowestBid مقدار دهی نشده باشد،این را به جای آن قرار می دهد.
         2. اگر lowestBid پیشتر مقداردهی شده بود،مقایسه می کند واگر کمتر از آن بود به جایش قرار میگیرد و lowestBidder هم برابر صاحب این bid می شود.

```
///@notice This function finalize and close the contract.
function finalize() public adminOrAutomaticTimeOutFinalize {

    // emergency Admin Finalizing...
    if(phase != Phase.Finished){
        phase = Phase.Finished;
    }

    description.winnerAddress = lowestBidder;
    description.winnerBid = lowestBid;
    lowestBidder.transfer(lowestBid);
    description.admin.transfer(ReserveFund - lowestBid);
    emit auctionFinished(lowestBidder,lowestBid);
    selfdestruct(description.admin);

}


modifier adminOrAutomaticTimeOutFinalize() {
    if(msg.sender == description.admin){
        require((block.number - startOpeningBlock) >= opening_len );
    }else{
        require(phase == Phase.Finished);
    }
    _;
}
```

ابتدا modifier را توضیح می دهیم،این modifier چک می کند که:

1- اگر فرستنده صاحبِ Auction باشد:
   a. در این صورت جهت finalize کردن نیاز است که مدت زمانِ opening بگذرد.(به این دلیل این شرط بررسی می شود که مثلا صاحب حراج با یکی از chooses ها تبانی علیه سایرین نکند).
2- اگر فرستنده نباشد:
   a. در این صورت باید حتما فازِ حراج به finished تبدیل شده باشد.

تابعِ finalize ابتدا لازم دارد که modifier ذکر شده را ارضا کرده باشد.سپس داریم که:

1- اگر فاز حراج متفاوت با finish باشد به سطحِ Finish ارتقا یابد.
2- lowestBidder به عنوان برنده تنظیم شود.
3- lowestBid هم به عنوان بهترین bid تنظیم گردد.
4- حال مبلغِ lowestBid از حساب Admin به lowestBidder واریز می گردد.
5- مابقی من التفاوت بین reserveFund و lowestBid به حساب صاحبِ حراج واریز می گردد.
6- رخداد auctionFinalized اعلام می گردد.
7- Contract به صورت خودکار نابود می شود(فراخوانیِ selfDestruct)

سوال پنجم:

1- ابتدا 5 عدد Bid به وجود می آوردم و سپس تابع generate هر کدام از آن هارا فرامی خوانیم:

هنگام deploy کردن لینک تایید شده در remix console داده می شود و پس از فراخوانیِ تابع generate نیز خروجی Json ای در بخش output به ما می دهد که همان tuple ِ روبه رو می دهد:

$$output = (Value, Nonce, Hash)$$

بنده این دو خروجی را کنار هم داده و برای هرکدام از5 پیشنهاد(Bid contract)،در زیر آورده ام.

Bid1:
https://ropsten.etherscan.io/tx/0x01f7d0440b55c6214968af1faf9c1f513c8215026d9dc482f3d648521fe44bac
```
{
    "0": "uint256: 1",
    "1": "bytes32: 0x418be6fc193a484ee29a5b1018d725f19dfd2b59f61d905ec2718828d1193470",
    "2": "bytes32: 0xd7b1c79e7d8998163afd907e77e460c2645be421cb9e66e18b041901209d388f"
}
```

Bid2:
https://ropsten.etherscan.io/tx/0xfffe35a268d8abe14160979681e5ba12a78771338a587fbcc18f95c37bde2f5c
```
{
    "0": "uint256: 2",
    "1": "bytes32: 0x4fd3c164bdba28f05f453d04bbc662e8e010048293007e5a41257652acd1ae84",
    "2": "bytes32: 0xfb0aeccb8e306cc0ea7b56d9c495f48d270d5eac34d67e0e5cd1d2b80493f836"
}
```

Bid3:
https://ropsten.etherscan.io/tx/0x5631759ac6793f38185c7acdec41b25b0e0e43c6200a47e6d2b8b2fcefd2e4fe
```
{
    "0": "uint256: 3",
    "1": "bytes32: 0x4fac6a754b99a25c3edbbc648185aca0296e4dccf92e1ed744ab31ea6bd2ed46",
    "2": "bytes32: 0x5b1620ee16f047c6e7d369d52487344affbb20eb2e9e70e5d878d97a3c70f3ec"
}
```

Bid4:
https://ropsten.etherscan.io/tx/0xb7c25e12c2adce816d911129833e072df527df98b1958245caba40783e492815
```
{
    "0": "uint256: 4",
    "1": "bytes32: 0xbfc8b6bb92c3ef0630839e0b168552326ed7797b84d77dd75a81fdc59b682011",
    "2": "bytes32: 0x564fda71e72f30e89415dda4f332c30d82eb59426d4889f0813c0b8d811fc609"
}
```

Bid5:
https://ropsten.etherscan.io/tx/0x3df5937b492cd143282e837dee1bc31e6ff53bca080d88eda88afaecaac35bf4
```
{
    "0": "uint256: 5",
    "1": "bytes32: 0x1e52704df584729ff6c3f14395adf4e608359a11dda712f5eab72f528cb0a8a1",
    "2": "bytes32: 0xf7e02e7a639f6dd34e09b5a0f09b6510f39f557fe34613da2a99434644fcd1bb"
}
```

همانطور که مشاهده می کنید مقدار پیشنهادیِ کاربران برای Bid برابر است با 1 تا 5.

حال contract ِ حراج را ایجاد می کنم.توجه کنید که admin را همان اکانت خودم قرار میدهم.

```
Auction:
https://ropsten.etherscan.io/tx/0x5a4a37a5836a1d50dbc846fa0fe74859725f7605056a8d6586eae18169640733
{
    "address _admin": "0x26AB4636Ea7945A074E98A36a2CEF5AC9684f01c",
    "uint256 _commitment_len": {
        "type": "BigNumber",
        "hex": "0x2710"
    },
    "uint256 _opening_len": {
        "type": "BigNumber",
        "hex": "0x2710"
    }
}
```

محتوای Json ذکر شده در بالا از بخشِ output خروجیِ console remix استخراج کرده ام.

حال تراکنش را Activate می کنیم:

```
Activating:
https://ropsten.etherscan.io/tx/0x3307a2625ff252065f9d464683f6fea4aba6b4c701ac1c0bda4a64ee80a3686d
[
    {
        "from": "0x1A694d3cfCC9Ef1D0cd72accE4F9e725A9869A37",
        "topic": "0xee2679bc2382e067cc3dfbda872852b5b1702b359c1fd6ce272c32dae5bacabf",
        "event": "auctionStarted",
        "args": {}
    }
]
```

همانطورکه ملاحظه می کنید event ِ بازگشایی حراج همانطورکه در کد نوشته شده بود اعلام گردیده است.

حال وارد مرحله Commit شده ایم و باید افراد اعلامِ Bid به حراج بکنند:

یک نمونه مثال از اعلام Bid را ذکر می کنیم:

همانطورکه در صورت سوال 4 اعلام گردیده بود :

Now, developers must upload their proposal on IPFS submit it's address in addition to a commitment to their bid, not bid itself. Specifically, this

همانطورکه می دانید جهت فرخوانیِ تابع bid از contract نیاز است تا دو ورودی _bidHash و _bidAddress به آن بدهیم و با توجه به گفته سوال،آدرس تراکنشِ آپلودِ contract ِ پیشنهاد(bid) را به عنوان _bidAddress می دهیم.مقدار _bidHash هم واضحا **سومین** خروجیِ تابعِ generate است.

$$generate\ function\ output = (Value, Nonce, Hash)$$

حال ورودی های تابع Bid را برای ارایه دهنده bid1 به شکل زیر پر کرده و خروجی remix به شرح زیر می گردد:

```
calling Bid function for first Bid:
    _bidHash = 0xd7b1c79e7d8998163afd907e77e460c2645be421cb9e66e18b041901209d388f;
    _bidAddress = 0x01f7d0440b55c6214968af1faf9c1f513c8215026d9dc482f3d648521fe44bac;

remix Console:
https://ropsten.etherscan.io/tx/0x6ac26b701359deeb44104613a933459b01b264b15afec925da9b22876fc9a563
```

مشابها برای سایر bid ها هم به همین طریق عمل می کنیم.حال که bid هارا پرکردیم می رویم سراغ مرحله بعد که در این مرحله admin فازِ را از Commitment به Opening ارتقا می دهد(با فراخوانیِ تابع()startOpening).

توجه کنید که نیاز است در این مرحله آدرس های مقبولش را اعلام کند.

خروجی remix console پس از فراخوانیِ تابعِ Opening به شرح زیر می گردد:

```
Opening:
https://ropsten.etherscan.io/tx/0x8d48c3e8f6e7d25b12e2892b6e0f5f9f671c0eb54b8482f5d2b3f90642fa9913
```

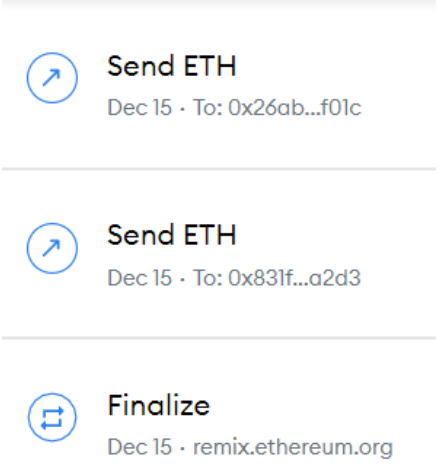
پس از فراخوانیِ این تابع،در console remix در بخش "event" مشاهده می کنید که ()openingStarted اعلام شده است.

حال نیاز است تا کاربران تابع Open را فرابخوانند و مقادیر Value و nonce خود را به حراج اعلام کنند(جهت احراز اصالت bid ِ اعلامی).

در نهایت admin با اعلام تابع Finalize به حراج خاتمه می دهد و واریز ها هم انجام می گیرد.

```
Finalize:
https://ropsten.etherscan.io/tx/0xbcdbef1c1ea16ca4ad66b2a893dbe1fbd30d4b361491380924eefed41a2f961d
```

همانطور که مشاهده می کنید پرداخت پس از اعمال Finalize انجام گردیده است.



همانطور که مشاهده می کنید پس از فراخونیِ تابعِ Finalize مقداری پول به حساب برنده واریز میگردد(تراکنش وسط) و باقی پول هم به حساب اصلی حراج(تراکنش بالا) واریز میگردد.(مشاهده می کنید که شماره کارت تراکنش بالا با شماره کارت بنده 0x26AB4636Ea7945A074E98A36a2CEF5AC9684f01c برابر است).

علی ذلک در خروجی remix console رخدادِ emit auctionFinished(lowestBidder,lowestBid) نیز اعلام شده است.

سوال ششم:

علاوه بر توضیحات سوال،نیاز است تا چند مورد زیر را هم اشاره کنم:

```
// Check if Ganache is running on port 7545
// Change it otherwise
const web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:7545"));

//Copy contract Address Here
const contractAddress = "0x565A8233cc3c37bd0D6148fd50469ca44775245b";

```

همچنین مقدار abi رو از tab موجود در بخش remix compiler کپی کرده و در کد جاوااسکریپت موجود paste کرده ام.

در نهایت جدول رای ها به صورت زیر گردید:(توجه من از هر Wallet Address یک رای ارسال کردم).