MỤC LỤC

[I./SMART CONTRACT 1](#_Toc515811164)

[1.Giới thiệu về Smart contract 1](#_Toc515811165)

[2.Ứng dụng của Smart contract 2](#_Toc515811166)

[3.Cách thức Smart Contract Etherem hoạt động 3](#_Toc515811167)

[II.GIỚI THIỆU VỀ ETHERUM 4](#_Toc515811168)

[1.Tiền mã hóa Ethereum là gì? 4](#_Toc515811169)

[2.Những công cụ làm nên đặc điểm riêng của Ethereum 6](#_Toc515811170)

[3.Nguyên lý hoạt động của đồng Ethereum 8](#_Toc515811171)

[III/ TOKEN 11](#_Toc515811172)

[IV. SOLIDITY 13](#_Toc515811173)

[1.Giới thiệu Solidity 13](#_Toc515811174)

[2. Cấu trúc Smart Contract 15](#_Toc515811175)

[3. Các loại dữ liệu 17](#_Toc515811176)

[a) Value Types 17](#_Toc515811177)

[b) Reference Types 22](#_Toc515811178)

[c) Units and Globally Available Variables 27](#_Toc515811179)

[d) Expressions and Control Structures 30](#_Toc515811180)

[4.Contracts 34](#_Toc515811181)

[a) Giới thiệu 34](#_Toc515811182)

[b)Tính chất 38](#_Toc515811183)

I./SMART CONTRACT

**1.Giới thiệu về Smart contract**

Hiểu theo cách đơn giản nghĩa là hợp đồng thông minh. Một hợp đồng thông minh khác với hợp đồng thông thường ở chỗ hợp đồng thông minh có thể được xác lập và thỏa thuận giữa con người với con người, máy móc với con người, hoặc máy móc với máy móc thay vì chỉ con người với con người như hợp đồng thông thường.

Khái niệm Smart contract được đề xuất lần đầu tiên năm 1996 bởi Nick Szabo. Ông định nghĩa một Smart contract là một giao thức máy tính tạo ra để số hóa, kiểm chứng và thực thi các thỏa thuận và nghĩa vụ được quy định trong một hợp đồng. Smart contract cho phép thực thi các giao dịch một cách đáng tin cậy mà không cần một bên thứ ba làm chứng. Các giao dịch đó có thể theo dấu và không thể đảo ngược được.

Ngày nay, với sự phát triển của công nghệ [blockchain](https://www.payvnn.com/blockchain/), smart contract / hợp đồng thông minhchủ yếu được sử dụng cho các mục đích tính toán hoạt động trên blockchain hoặc sổ cái phân tán.

Nổi bật nhất ta có thể thấy đó là [Ethereum](https://www.payvnn.com/eth-la-gi/), đồng tiền mã hóa với một framework cho smart contract được sử dụng rộng rãi nhất hiện nay.

Những ưu điểm của Smart contract

Ưu điểm rõ ràng nhất của **hợp đồng thông minh** mà ta có thể nhận thấy ngay đó là sự can thiệp của máy tính giúp cho mọi quy trình được **tự động hóa**, không cần sự can thiệp của bên thứ 3.

Tiết kiệm chi phí là ưu điểm tiếp theo, thay vì phải trả phí cho bên làm chứng thì bạn chỉ cần trả một khoản phí rất nhỏ cho mạng lưới blockchain.

Độ tin tưởng cao: với smart contract và blockchain mã hóa, không một bên nào có thể can thiệp vào quá trình thực thi cũng như thỏa thuận của hợp đồng một khi đã được hoàn tất.

Minh bạch, rõ ràng: mọi giao dịch đều được ghi nhận trên blockchain, có thể truy dấu ra nguồn gốc và không thể đảo nghịch giao dịch.

Nhanh, gọn và tiện lợi: với sự trợ giúp của blockchain, một hợp đồng thông minh có thể thiết lập và thực thi chỉ trong vài giây. Hợp đồng cũng có thể cùng lúc thiết lập với nhiều người khác nhau và dùng đi dùng lại nhiều lần, rất tiện lợi.

Linh hoạt: hệ thống máy tính tự động sắp xếp và thực thi chính xác những quy định trong hợp đồng thông minh. Từng trường hợp có thể được xử lý linh hoạt và hiệu quả cho người sử dụng

2.Ứng dụng của Smart contract

Để hiểu thêm về cách thức hoạt động của smart contract, ta hãy đi vào vài [ứng dụng](https://www.payvnn.com/dapp/) thực tiễn.

Smart contract kết hợp với blockchain có thể ứng dụng trong rất nhiều lĩnh vực, từ tài chính, bảo hiểm, ngân hàng cho đến nhỏ hơn như giải trí, bình chọn… Với mỗi lĩnh vực ta lại sử dụng một loại hợp đồng thông minh khác nhau, ví dụ hợp đồng vay tiền, hợp đồng đóng tiền định kỳ, hợp đồng mua bán, hợp đồng chuyển nhượng…

Bạn có thể hiểu đơn giản là khi một lĩnh vực được áp dụng hợp đồng thông minh, mọi quyết định hoặc giao dịch thuận theo hợp đồng đều được xử lý tự động khi thỏa điều kiện đã đưa ra. Ví dụ:

* Hợp đồng vay tiền có lãi suất: bạn vay một số tiền có lãi suất và ký hợp đồng này, khi đến một ngày nhất định, hợp đồng thông minh sẽ tự trừ tiền trong tài khoản của bạn và gửi cho người cho vay theo tỉ lệ đúng như đã ký kết.
* Hợp đồng chuyển nhượng: nếu bạn trả 50% số tiền bạn sẽ được giữ cọc món hàng, nếu trả đủ 100% bạn sẽ được nhận món hàng đó, nếu bạn trả 50% nhưng không thanh toán đủ trong thời gian quy định thì bạn sẽ mất cọc, tất cả đều được tự động xử lý với hợp đồng thông minh.

Một ứng dụng rộng rãi của hợp đồng thông minh trong lĩnh vực tiền mã hóa đó là phát hành ICO. Một hợp đồng thông minh được viết ra để khi nhà đầu tư gửi vào địa chỉ của hợp đồng một số tiền, họ sẽ nhận lại được một số [token](https://www.payvnn.com/coin-va-token-la-gi-phan-biet-coin-va-token/) tương ứng với số tiền họ đã bỏ ra. Hợp đồng thông minh đó có thể bổ sung một số điều kiện như đóng băng số tiền nhận được trong một khoản thời gian quy định hoặc hủy các token không bán được.

Việc ứng dụng hợp đồng thông minh và blockchain trong ICO giúp các nhà đầu tư có thể theo dõi dự án đã gọi được bao nhiêu tiền, có đạt được mục tiêu hay không, và nhiều thông tin khác nữa.

3.Cách thức Smart Contract Etherem hoạt động

Bạn cần biết rằng Bitcoin chính là mô hình đầu tiên áp dụng smart contract cơ bản khi mạng lưới có thể chuyển giá trị của người này sang người khác. Các node trong mạng lưới chỉ xác thực giao dịch khi đạt được các điều kiện nhất định.

Nhưng Bitcoin chỉ giới hạn mục đích sử dụng trong tiền mã hóa.

Trái lại, Ethereum thay thế ngôn ngữ lập trình hạn chế của Bitcoin với một ngôn ngữ cho phép các nhà phát triển viết các ứng dụng của riêng mình.

Ethereum cho phép các nhà phát triển lập trình các smart contract của riêng họ, hoặc gọi là “autonomous agents” theo cách gọi trong white paper. Ngôn ngữ đó gọi là “Turing-complete”.

Smart contracts có thể:

* Hoạt động như một tài khoản “đa chữ ký”, nghĩa là tiền chỉ được sử dụng khi đạt được tỉ lệ phần trăm người đồng ý.
* Quản lý thỏa thuận giữa người dùng, ví dụ hợp đồng bảo hiểm.
* Cung cấp tiện ích cho các hợp đồng thông minh khác, tương tự như cách các thư viện phần mềm hoạt động.
* Lưu trữ các thông tin về ứng dụng, như thông tin đăng ký tên miền hoặc hồ sơ thành viên.

Một Ethereum Smart Contract tập hợp những đoạn mã được viết theo ngôn ngữ mà Ethereum blockchain hiểu được. Các đoạn mã này có nhiệm vụ thu thập các dữ liệu và so sánh với các điều kiện được đưa ra để dựa vào đó quyết định các thao tác sẽ được thực thi.

Trong phát hành token ICO với smart contract, bạn còn phải tuân thủ theo các quy chuẩn trong việc lập trình để tạo ra các token khau tùy theo mục đích của bạn, các token thông dụng trên smart contract như token [ERC20](https://www.payvnn.com/erc-token/), ERC223,…

II.GIỚI THIỆU VỀ ETHERUM

Cùng với sự chuyển động không ngừng nghỉ của thế giới, tiền điện tử ra đời như một quy luật tất yếu trong điều kiện công nghệ thông tin ngày một hiện đại. Hiện nay, trên thế giới có 2 loại tiền điện tử nổi bật nhất là Bitcoin và Ethereum. Trong bài viết này, chúng tôi sẽ cùng bạn tìm hiểu về ethereum để trả lời cho câu hỏi Ethereum là gì? Và giải đáp những thắc mắc liên quan đến nó một cách cụ thể và chi tiết nhất.

**1.Tiền mã hóa Ethereum là gì?**



Ethereum là nơi chứa các dữ liệu mở. Chúng hoạt động nhờ nền tảng blockchain ( hệ thống chuỗi các khối). Nó có tính phân tán cao nhờ các thuật toán máy tính, ứng với từng chức năng khác nhau qua hợp đồng thông minh. **Ethereum** cung cấp máy ảo ( EVM) để thực hiện các lệnh dựa vào nhiều điểm truy cập của hệ thống trên toàn cầu được công khai. Ngoài ra, Ethereum còn là một đồng tiền điện tử đã được mã hóa, có đơn vị tiền tệ gọi là Ether. Loại tiền này có thể được tìm kiếm và được giao dịch giữa các tài khoản với nhau. Tóm lại, hệ sinh thái tài chính phân tán đã được thiết lập riêng và cố định bởi Ether. Ngoài ra điểm nổi bật của nó là có mã nguồn mở và hệ điều hành có tính năng hợp đồng thông minh.

**Tìm hiểu về Ethereum và lịch sử hình thành Ethereum**



Tìm hiểu về Ethereum, chúng được biết đến đầu tiên thông qua bản thảo năm 2013 của Vitalik Buterin - Một người lập trình viên có ý tưởng muốn nâng cấp hoàn thiện Bitcoin hơn. Với mong muốn khắc phục những điểm hạn chế còn tồn đọng ở Bitcoin. Song, đề án của ông không đạt được nhóm cố vấn Bitcoin ủng hộ. Không từ bỏ hy vọng, ông quyết tâm xây dựng một nền tảng hoàn toàn mới mẻ với ngôn ngữ khái quát và mở rộng thêm. Dự án chính thức được biết đến vào năm 2014 bởi công ty Ethsuisse. Một thời gian sau, một tổ chức phi lợi nhuận về Ethereum cũng ra đời tại Thụy Sĩ, được đặt tên là Ethereum Foundation. Tiền mã hóa Ethereumbước đi đầu tiên được tài trợ bởi các thành viên trực tuyến trong khoảng tháng 7 - 8 ( 2014)

Ethereum ghi nhận sự quan tâm đông đảo của truyền thông. Đặc biệt, vào năm 2016 khi tổ chức phân cấp của Ethereum ra đời, lấy tên Dao. Dự án này đã thu về số tiền khủng lển tới 150 triệu USD nhờ các thành viên đóng góp. Tuy nhiên, dự án này đã nhanh chóng sụp đổ do bị hack. Từ đó, dẫn đến những tranh cãi nội bộ. Kết quả, đồng tiền mã hóa Ethereum được chia thành 2 loại khác nhau: Ethereum Blockchain cũ và Ethereum Classi

**Đặc điểm của tiền mã hóa Ethereum coin là gì?**



2.Những công cụ làm nên đặc điểm riêng của Ethereum

Ethereum khác biệt so với bitcoin và các đồng tiền điện tử khác ở chỗ, nó do 4 yếu tố chính tạo thành, là những bộ phận quan trọng nhất của Ethereum: Ether, hợp đồng thông minh, tài khoản và máy ảo Ethereum.

* Ether:

Là đơn vị tiền tệ của Ethereum. Nó được sử dụng để giao dịch tại sàn cho phép dùng tiền mã hóa. Ngoài ra, đồng tiền này còn áp dụng để thanh toán các dịch vụ khác trên mạng Ethereum

* Hợp đồng thông minh:

Hợp đồng thông minh là việc tạo ra những điều khoản, quy định giữa 2 bên tham gia giao dịch mà không cần bên trung gian do các phương tiện kỹ thuật soạn thảo. Chúng có nội dung như những bản hợp đồng pháp lý thực tế khác. Điểm hay của dạng hợp đồng này nằm ở chỗ chúng không cần những thông tin cá nhân mà vẫn có thể thiết lập hợp đồng. Nghĩa là, các cá nhân tham gia hoàn toàn có thể thỏa thuận hợp đồng mà không cần phải biết nhau là ai, không cần phải tin tưởng nhau

* Hệ thống Blockchain:

Khi những thông tin về địa chỉ, số ether, giao dịch giữa người dùng với nhau mã hóa xong, nó sẽ được chuyển vào hệ thống Blockchain để làm cơ sở dữ liệu cho những lần giao dịch kế tiếp

* Hệ thống đồng thuận:

Là chương trình mã hóa không có chủ định, đột ngột nhằm bảo vệ thông tin người dùng và số giao dịch của họ được cẩn thận, bảo mật hơn

* Thợ đào mỏ:

Hiểu đơn giản, thợ đào mỏ chính là những người làm nhiệm vụ giải mã dữ liệu cá nhân của hai bên người mua và bán. Đồng thời xác nhận giao dịch giữa 2 bên.

* Máy ảo Ethereum:

Đây là “nơi” để vận hành xử lý thông tin của mạng máy tính cung cấp. Những tổ chức, cá nhân chuyên xử lý thông tin sẽ tập hợp trong “ngôi nhà” này rồi phân chia nhau giải mã dữ liệu Blockchain. Đây được xem như hệ thống khủng để giải quyết tất cả thông tin về giao dịch trên phạm vi toàn thế giới.

3.Nguyên lý hoạt động của đồng Ethereum



Tiền mã hóa Ethereum hoạt động cũng giống như Bitcoin, tuy nhiên nó có nhiều đổi mới, sửa đổi, và sự khác biệt của riêng nó và dưới đây chúng tôi sẽ nói cụ thể.

Chúng tôi đã đề cập đến Hợp đồng Thông minh ở phần trên của bài viết này. Hợp đồng thông minh là trái tim của Ethereum, là "hợp đồng mã hóa" có thể được lập trình để thực hiện một chức năng, giao dịch cụ thể, tại một thời điểm nhất định hoặc sau khi các chí nhất định đã được đáp ứng. Bạn nên chọn thiết lập một thỏa thuận kinh doanh và có hợp đồng một cách tự động để thực hiện các điều khoản của thỏa thuận mà không cần sự can thiệp của bên thứ ba.

Ethereum cũng tạo ra một loại ứng dụng mới có sẵn bởi một mạng lưới các máy tính tham gia. Chúng được gọi là dApps, những ứng dụng được phân quyền. Ví dụ: ứng dụng nhắn tin có tên là WhatsDapp, thông điệp từ tôi đến bạn sẽ được phân tán giữa nhiều máy tính cá nhân khác nhau, mỗi người nhận được một mẩu tin và nó sẽ được lắp ráp lại khi nó được gửi đến thiết bị của bạn. Các chủ sở hữu của những máy tính này đang cho mượn năng lực xử lý từ máy tính của họ để đổi lấy ether.

**Giá tiền mã hóa Ethereum là bao nhiêu?**

[Etherem có giá bao nhiêu](https://bigcoinvietnam.com/ethereum-co-gia-bao-nhieu--nhung-yeu-to-tac-dong-den-gia-ethereum-) hoàn toàn phụ thuộc vào việc khai thác của thợ đào ra chúng. Khác với Bitcoin, Ethereum không hề bị giới hạn về số lượng. Do vậy, Ethereum càng ngày càng có người khát khao sở hữu chúng. Vì thế, nó kích tỷ giá Ethereum lên rất cao. Theo nghiên cứu, 1 ETH = 697 USD. Quy đổi với Bitcoin, ETH = 0.0750634 BTC. Tính ra tiền Việt, 1 ETH có giá vào khoảng 15.822.581 VNĐ. Như vậy, có thể thấy rằng dù ETH có tỷ giá vẫn thấp hơn so với BTC nhưng vẫn cho thấy giá trị rất cao mà nó mang lại cho thị trường và các nhà đầu tư. Nhìn chung, Ethereum vẫn còn rất mới đối với đại đa số người dân trên toàn cầu

**Cách sở hữu đồng tiền mã hóa Ethereum như thế nào?**

Mua Ethereum bằng tiền hợp lệ

Hãy dùng các đơn vị tền tệ hợp pháp như Euro, dollar để mua. Đặc biệt, đồng Kraken là một trong những cách tốt nhất để sở hữu Ethereum.

Mua Bitcoin trước, Ethereum sau

Nếu không tìm ra được một loại tiền có thể giao dịch tốt thì hãy làm theo cách khác. Bạn nên mua Bitcoin trước bằng cách dùng Coinbase. Tuy nhiên, hình thức này chỉ được phép áp dụng ở một số quốc qua. Thanh toán theo hình thức ghi nợ giới hạn số lượng trong ngày. Lúc này, bạn hoàn toàn dễ mua Bitcoin để có ETH thông qua trao đổi mua theo cặp BTC - ETH.

Đào Ethereum

Nếu bạn không biết [mua tiền mã hóa ethereum ở đâu](https://bigcoinvietnam.com/mua-tien-ma-hoa-ethereum-o-dau-uy-tin--cach-thuc-mua-nhu-the-nao-) thì Một cách khác để sở hữu Ethereum là có hợp đồng đào Ether hoặc bạn tự đào chúng. Bạn có thể tự tìm hiểu cách đào mỏ hoặc tham gia vào hội nhóm với những người bạn cùng suy nghĩ. Còn nếu chọn phương pháp mua hợp đồng, hãy bỏ ra ít lãi để trả cho người làm quản lý, lập chi phí giúp bạn

Lưu trữ tiền mã hóa Ethereum ở đâu?

Cũng giống như Bitcoin, Ethereum thường được lưu trữ trong một chiếc ví ( Ether Wallet). Chiếc ví nhỏ này có thể cho phép người sử dụng tạo ra địa chỉ công khai . Qua đó, bạn hay người khác có thể gửi/nhận tiền vào trong đó. Bạn hãy yên tâm rằng độ bảo mật của ví Ethẻ khá tốt vì nó chỉ cho phéo mình bạn truy cập. Ngay cả quản trị viên cũng không thể vào nếu họ không có “chìa khóa” mở của. Điều bạn cần làm là không để lộ các thông tin ra ngoài

**Ưu, nhược điểm của đồng Ethereum**

Ưu điểm của Ethereum

* Giao dịch nhanh chóng: Thời gian tạo khối chỉ vỏn vẹn khoảng 15 giây khiến chúng trở thành mạng giao dịch có tốc độ cao và sử dụng hình thức GHOST. Nhờ đó, tiết kiệm được nhiều thời gian, tiền bạc cho cả người dùng mua - bán
* Không giới hạn số lượng Ethereum: Ethereum không bị đào ngược Ether nên bạn yên tâm đầu tư mà không lo gặp khó khăn
* Phí giao dịch tính trên khối lượng lưu trữ sử dụng
* Được các nhà đầu tư lớn tin tưởng, ủng hộ
* Mạng lưới đang lan tỏa rộng, trải dài khắp thế giới.
* Ethereum của hệ thống được mặc định tự động hóa
* Không lo bị giả: Ethereum được hiện hữu dưới dạng tiền điện tử, không tồn tại dưới dạng vật chất. Vì thế, không phải đau đầu lo đơn vị tiền tệ Ether bị làm giả

Nhược điểm của đồng tiền mã hóa Ethereum

* Vấn đề xã hội: Tệ nạn rửa tiền, tin tặc...có khả năng hoành hành với phạm vi xuyên quốc gia
* Dễ bị hacker tấn công

Ít người dùng: Tại các quốc gia đang phát triển như Việt Nam, Lào, Campuchia...người dân tại những đất nước này đã quá quen sử dụng tiền mặt để giao dịch mua bán trong cuộc sống hàng ngày. Vì vậy, để họ dùng Ethereum thay thế là điều không dễ vì không hiểu biết nhiều về Ethereum và tính thông dụng của nó trong nội địa.

* Khó sử dụng: Dù là sản phẩm có tính công nghệ cao. Song, đây lại là con dao hai lưỡi với nó. Người am hiểu, thạo công nghệ máy tính sẽ sử dụng dễ dàng. Và ngược lại, một số người ở độ tuổi trung niên, người già...Trình độ sử dụng máy tính, tiếp thu kiến thức kém sẽ gặp khó khăn trong việc sử dụng loại tiền này

III/ TOKEN

Token trong hệ thống Etherum là một loại hàng hóa có thể trao đổi. Token thực hiện theo một chuẩn (ERC20) nên luôn tương thích Ethereum wallet bất kỳ client hay hợp đồng có thể sử dụng

Trước tiên, ERC là viết tắt của Ethereum Request for Comment, là dạng sơ khởi của EIP (Ethereum Improvement Proposal - Đề xuất cải tiến Ethereum). Trong đó, EIP20 là một đề xuất liên quan đến việc chuẩn hoá API dành cho token. Trước khi có đề xuất này, toàn bộ những token khi được tạo ra đều phải cài đặt tất cả những hàm và biến số liên quan tới định nghĩa về số token supply, kiểm tra số dư token trên một địa chỉ, chuyển token giữa các địa chỉ v.v.

ERC20 là một tiêu chuẩn mà bạn có thể tuân theo hoặc không tuân theo khi tạo token, và khi tuân theo thì token bạn tạo ra sẽ được gọi là token ERC20.

Chuẩn token mô tả các hàm và sự kiện mà  Ethereum token contract có thể triển khai

Một chuẩn ERC20 Token bao gồm:

contract ERC20Interface {

function totalSupply() public constant returns (uint);

function balanceOf(address tokenOwner) public constant returns (uint balance);

function allowance(address tokenOwner, address spender) public constant returns (uint remaining);

function transfer(address to, uint tokens) public returns (bool success);

function approve(address spender, uint tokens) public returns (bool success);

function transferFrom(address from, address to, uint tokens) public returns (bool success);

event Transfer(address indexed from, address indexed to, uint tokens);

event Approval(address indexed tokenOwner, address indexed spender, uint tokens);

}

Trong đó các hàm ta cần quan tâm như:

* totalSupply() trả về số tổng số token
* balanceOf(addr) trả về số token từng địa chỉ
* transfer(addr,uni) gửi uni token đến địa chỉ người nhận
* allowance(address tokenOwner, address spender) ủy quyền cho spender được phép gửi token
* approve() đồng thuận cho phép gửi bao nhiêu token

Ngoài ra có thể khởi tạo một số thông tin

Để tiển khai một smart contract cần thực hiện các bước sau:

**Bước 1: Lên đặc điểm của token**

Để tạo ra một token ERC20, bạn phải định nghĩa những thuộc tính sau:

* Tên của token
* Mã của token
* Số chữ số thập phân sau dấu phẩy của token
* Tổng số token muốn phát hành

Ví dụ chúng ta sẽ tạo token như sau:

* Tên: Cryptolife
* Mã: CTL
* Số chữ số thập phân sau dấu phẩy: 0
* Tổng số token phát hành: 1000

**Bước 2: Viết mã cho phần Smart Contract chuẩn ERC20**

**Bước 3: Test token trên Testnet**

Ở bước này, chúng ta sẽ phát hành token lên một mạng lưới test (Test net) để xem nó có hoạt động ok hay không. Chúng ta sẽ chọn Ropsten Testnet để test token này.

Biên dịch mã nguồn contract bằng cách sử dụng  Solidity Remix Compiler

Sau đó submit deloy lên Testnet

IV. SOLIDITY

1.Giới thiệu Solidity

Đây là ngôn ngữ  contract-oriented, ngôn ngữ bậc cao triển khai các smart contracts. Tương như c++, python được thiết kế dành cho EVM

Ngôn ngữ Solidity định kiểu mạnh, hỗ trợ thừa kế, thư viện và kiểu định nghĩa người dùng phức tạp cùng với các tính năng khác.

IDE dành phát triển solidity như Remix, Mix, Etherum studio

https://remix.ethereum.org/#optimize=false&version=soljson-v0.4.21+commit.dfe3193c.js

Ví dụ về smart contract:

pragma solidity ^0.4.0;

contract SimpleStorage {

uint storedData;

function set(uint x) public {

storedData = x;

}

function get() public view returns (uint) {

return storedData;

}

}

Hay:

pragma solidity ^0.4.21;

contract Coin {

*// The keyword "public" makes those variables*

*// readable from outside.*

address public minter;

mapping (address => uint) public balances;

*// Events allow light clients to react on*

*// changes efficiently.*

event Sent(address from, address to, uint amount);

*// This is the constructor whose code is*

*// run only when the contract is created.*

function Coin() public {

minter = msg.sender;

}

function mint(address receiver, uint amount) public {

if (msg.sender != minter) return;

balances[receiver] += amount;

}

function send(address receiver, uint amount) public {

if (balances[msg.sender] < amount) return;

balances[msg.sender] -= amount;

balances[receiver] += amount;

emit Sent(msg.sender, receiver, amount);

}

}

Như ta thấy, smart contract là kiểu contract mà chứa các kiểu dữ liệu và các hàm thực thi một số lệnh được triển khai bằng solidity. Để hiểu rõ hơn cách thức thực hiện smart contract, ta tìm hiểu sâu về solidity.

Thứ nhất, về Layout of a Solidity Source File

Khai báo Version Pragma

* pragma solidity ^0.4.0;

Importing other Source Files

* import "filename";
* import \* as symbolName from "filename";
* import "filename" as symbolName;

Chú ý:

* Trình biên dịch không thể chạy version sớm hơn 0.4.0 và chưa hoạt động version 0.5.0. Khai báo version 0.4.x thông qua từ khóa pragma
* Paths: Sử dụng “/” chỉ paths các FileName, . Chỉ thư mục hiện tai và .. Chỉ thư mục parent

Comments: chú thích từng dòng  (//), chú thích nhiều dòng (/\*...\*/)

Ví dụ:

*// This is a single-line comment.*

*/\**

*This is a*

*multi-line comment.*

*\*/*

## **2. Cấu trúc Smart Contract**

Class trong object-oriented languages

Một contract chứa State Variables, Functions, Function Modifiers, Events, Struct Types và Enum Types

Kế thừa từ các contract khác

Trong đó:

State Variables: chứa dữ liệu trong  contract storage

pragma solidity ^0.4.0;

contract SimpleStorage {

uint storedData; *// State variable*

*// ...*

}

Functions: thực thi các lệnh trong smart contract

pragma solidity ^0.4.0;

contract SimpleAuction {

function bid() public payable { *// Function*

*// ...*

}

}

Function Modifiers: được sử dụng chỉnh sửa các hàm đã được định nghĩa

pragma solidity ^0.4.22;

contract Purchase {

address public seller;

modifier onlySeller() { *// Modifier*

require(

msg.sender == seller,

"Only seller can call this."

);

}

function abort() public onlySeller { *// Modifier usage*

*// ...*

}

}

Events: khai báo một sự kiện thông qua event và kích hoạt sự kiện trong hàm bằng từ khóa emit

pragma solidity ^0.4.21;

contract SimpleAuction {

event HighestBidIncreased(address bidder, uint amount); *// Event*

function bid() public payable {

*// ...*

emit HighestBidIncreased(msg.sender, msg.value); *// Triggering event*

}

}

Struct Types: kiểu dữ liệu định nghĩa chứa nhiều loại dữ liệu khác nhau

pragma solidity ^0.4.0;

contract Ballot {

struct Voter { *// Struct*

uint weight;

bool voted;

address delegate;

uint vote;

}

}

Enum Types

**pragma** solidity **^**0.4.0;

**contract** Purchase {

**enum** State { Created, Locked, Inactive } *// Enum*

}

3. Các loại dữ liệu

a) Value Types

Booleans:

* Giá trị true và false
* Toán tử : ! , &&, ||, == , !=

Integers:

* int/uint với size khác nhau. Size từ 8 bit đến 256 bit được khai báo bằng cách thêm số bit số sau. VD: int8/unit8, int16/uint116. Int/uint là viết tắt kiểu dữ liệu int/uint 256 bit
* Các phép toán so sánh: <=, <, ==, !=, >=, >
* Bit operators: &, |, ^ , ~
* Phép tính: +, -, \*, /, % , \*\*, << , >>

Fixed Point Numbers:

* fixed / ufixed: ufixedMxN và fixedMxN trong đó M là số bit dùng, N chữ số thập phân từ 0 đến 80. Size từ 8 bit đến 256 bit được khai báo bằng cách thêm số bit số sau. VD: fixed8x19/ufixed8x19
* ufixed and fixed viết tắt  ufixed128x19 và fixed128x19
* Toán tử:
* Các phép so sánh: <=, <, ==, !=, >=, > (evaluate to bool)
* Phép toán: +, -, unary -, unary +, \*, /, % (remainder)

Address:

* Độ dài 20bytes, có các thuộc tính
* Toán tử: <=, <, ==, !=, >= and >
* Members of Addresses

*balance and transfer*

address x = 0x123;

address myAddress = this;

if (x.balance < 10 && myAddress.balance >= 10) x.transfer(10);

*Send*: tranfer mức thấp

*call, callcode and delegatecall*

address nameReg = 0x72ba7d8e73fe8eb666ea66babc8116a41bfb10e2;

nameReg.call("register", "MyName");

nameReg.call(bytes4(keccak256("fun(uint256)")), a);

Fixed-size byte arrays

* Có các kiểu bytes1, bytes2, bytes3, …, bytes32. Số phía sau chỉ size từng kiểu dữ liệu. trong đó byte chỉ bytes1
* Các phép so sánh: <=, <, ==, !=, >=, > (evaluate to bool)
* Bit operators: &, |, ^ (bitwise exclusive or), ~ (bitwise negation), << (left shift), >> (right shift)
* Truy cập các phần tử của mãng: x[k] trong đó k là vị trí của phần tử và 0<=k< L(chiều dài mảng).
* Thuộc tính .length: xác định số phần tử array

Dynamically-sized byte array: bytes và string

Address Literals: kiểu Hexadecimal  khoảng  39 và 41 ký tự và phải phù hợp quy định checksum test produce

Rational and Integer Literals

* Integer literals là chuỗi 0-9
* Số thập được định nghĩa cách “.” Ex: 1.3

String Literals : biểu diện “” hoặc ‘’. Có chứa các ký tự đặc biệt \n, \xNN and \uNNNN. \xNN

Hexadecimal Literals: bắt đâu hex và nội dung trong “” or ‘’. Ex: hex"001122FF"

Enums

pragma solidity ^0.4.16;

contract test {

enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }

ActionChoices choice;

ActionChoices constant defaultChoice = ActionChoices.GoStraight;

function setGoStraight() public {

choice = ActionChoices.GoStraight;

}

*// Since enum types are not part of the ABI, the signature of "getChoice"*

*// will automatically be changed to "getChoice() returns (uint8)"*

*// for all matters external to Solidity. The integer type used is just*

*// large enough to hold all enum values, i.e. if you have more values,*

*// `uint16` will be used and so on.*

function getChoice() public view returns (ActionChoices) {

return choice;

}

function getDefaultChoice() public pure returns (uint) {

return uint(defaultChoice);

}

}

Function Types: Có 2 loại i*nternal* và *external* functions

Công thức chung:

function (<parameter types>) {internal|external} [pure|constant|view|payable] [returns (<return types>)]

Trong đó:

* Từ khóa funtion khai báo hàm
* parameter types:thêm các tham số vào trong hàm. Có thể có hoặc không
* internal|external: lựa chọn để hàm là hàm i*nternal* hoặc *external*
* pure|constant|view|payable: làm rõ ràng hoặc minh bạch các hàm.
* Cuối cùng trả về giá trị. Có thể có hoặc không.

Ví dụ về sử dụng internal:

pragma solidity ^0.4.16;

library ArrayUtils {

*// internal functions can be used in internal library functions because*

*// they will be part of the same code context*

function map(uint[] memory self, function (uint) pure returns (uint) f)

internal

pure

returns (uint[] memory r)

{

r = new uint[](self.length);

for (uint i = 0; i < self.length; i++) {

r[i] = f(self[i]);

}

}

function reduce(

uint[] memory self,

function (uint, uint) pure returns (uint) f

)

internal

pure

returns (uint r)

{

r = self[0];

for (uint i = 1; i < self.length; i++) {

r = f(r, self[i]);

}

}

function range(uint length) internal pure returns (uint[] memory r) {

r = new uint[](length);

for (uint i = 0; i < r.length; i++) {

r[i] = i;

}

}

}

contract Pyramid {

using ArrayUtils for \*;

function pyramid(uint l) public pure returns (uint) {

return ArrayUtils.range(l).map(square).reduce(sum);

}

function square(uint x) internal pure returns (uint) {

return x \* x;

}

function sum(uint x, uint y) internal pure returns (uint) {

return x + y;

}

}

Về sử dụng external:

pragma solidity ^0.4.22;

contract Oracle {

struct Request {

bytes data;

function(bytes memory) external callback;

}

Request[] requests;

event NewRequest(uint);

function query(bytes data, function(bytes memory) external callback) public {

requests.push(Request(data, callback));

emit NewRequest(requests.length - 1);

}

function reply(uint requestID, bytes response) public {

*// Here goes the check that the reply comes from a trusted source*

requests[requestID].callback(response);

}

}

contract OracleUser {

Oracle constant oracle = Oracle(0x1234567); *// known contract*

function buySomething() {

oracle.query("USD", this.oracleResponse);

}

function oracleResponse(bytes response) public {

require(

msg.sender == address(oracle),

"Only oracle can call this."

);

*// Use the data*

}

}

**b) Reference Types**

Data location: lưu trữ trong memory or in storage

Theo mặc định: tham số trong hàm lưu memory, local variables lưa storage. Ngoài ra còn có calldata nơi dùng lưu trử các tham số của hàm external

Ví dụ:

pragma solidity ^0.4.0;

contract C {

uint[] x; *// the data location of x is storage*

*// the data location of memoryArray is memory*

function f(uint[] memoryArray) public {

x = memoryArray; *// works, copies the whole array to storage*

var y = x; *// works, assigns a pointer, data location of y is storage*

y[7]; *// fine, returns the 8th element*

y.length = 2; *// fine, modifies x through y*

delete x; *// fine, clears the array, also modifies y*

*// The following does not work; it would need to create a new temporary /*

*// unnamed array in storage, but storage is "statically" allocated:*

*// y = memoryArray;*

*// This does not work either, since it would "reset" the pointer, but there*

*// is no sensible location it could point to.*

*// delete y;*

g(x); *// calls g, handing over a reference to x*

h(x); *// calls h and creates an independent, temporary copy in memory*

}

function g(uint[] storage storageArray) internal {}

function h(uint[] memoryArray) public {}

}

Arrays

An array of fixed size k thì được viết T[k], an array of dynamic size T[]. Ví dụ: 5 phần tử T[][5]. Truy xuất phần tử 2 trong uint 3: T[2][1]

Allocating Memory Arrays

pragma solidity ^0.4.16;

contract C {

function f(uint len) public pure {

uint[] memory a = new uint[](7);

bytes memory b = new bytes(len);

*// Here we have a.length == 7 and b.length == len*

a[6] = 8;

}

}

Array Literals / Inline Arrays

pragma solidity ^0.4.16;

contract C {

function f() public pure {

g([uint(1), 2, 3]);

}

function g(uint[3] \_data) public pure {

*// ...*

}

}

Thuộc tính:

* length-trả về chiều dài
* Push- thêm một phần tử vào mảng
* Pop- xóa một phần tử cuối ở mảng

Structs

pragma solidity ^0.4.11;

contract CrowdFunding {

*// Defines a new type with two fields.*

struct Funder {

address addr;

uint amount;

}

struct Campaign {

address beneficiary;

uint fundingGoal;

uint numFunders;

uint amount;

mapping (uint => Funder) funders;

}

uint numCampaigns;

mapping (uint => Campaign) campaigns;

function newCampaign(address beneficiary, uint goal) public returns (uint campaignID) {

campaignID = numCampaigns++; *// campaignID is return variable*

*// Creates new struct and saves in storage. We leave out the mapping type.*

campaigns[campaignID] = Campaign(beneficiary, goal, 0, 0);

}

Mappings:

Kiểu dữ liệu mapping(\_KeyType => \_ValueType). Trong đó: \_KeyType có thể tất cả kiểu dữ liệu ngoại trừ các kiểu mapping, a dynamically sized array, a contract, an enum and a struct. \_ValueType có thể là kiểu dữ liệu nào bao gồm cả mapping

Ví dụ:

pragma solidity ^0.4.0;

contract MappingExample {

mapping(address => uint) public balances;

function update(uint newBalance) public {

balances[msg.sender] = newBalance;

}

}

contract MappingUser {

function f() public returns (uint) {

MappingExample m = new MappingExample();

m.update(100);

return m.balances(this);

}

}  
Operators Involving Lvalues:

* a += e tương đương a=a+e
* Operators -=, \*=, /=, %=, |=, &= và ^=
* a++ and a– hoặc ++a, --a

Delete

pragma solidity ^0.4.0;

contract DeleteExample {

uint data;

uint[] dataArray;

function f() public {

uint x = data;

delete x; *// sets x to 0, does not affect data*

delete data; *// sets data to 0, does not affect x which still holds a copy*

uint[] storage y = dataArray;

delete dataArray; *// this sets dataArray.length to zero, but as uint[] is a complex object, also*

*// y is affected which is an alias to the storage object*

*// On the other hand: "delete y" is not valid, as assignments to local variables*

*// referencing storage objects can only be made from existing storage objects.*

}

}

Conversions between Elementary Types:

Implicit Conversions(Chuyển đổi ngầm): chuyển đổi nếu 2 kiểu dữ liệu cùng kích thước hoặc dữ liệu cùng kiểu mà không làm mất dữ liệu. Vi dụ như uint8 có thể convert uint16 và int128 đến int256.

Explicit Conversions:

int8 y = -3;

uint x = uint(y);

Type Deduction:

uint24 x = 0x123;

var y = x;

c) Units and Globally Available Variables

Ether Units: wei, finney, szabo hoặc ether. Ví dụ: 2 ether == 2000 finney

Time Units: seconds, minutes, hours, days, weeks và years

Special Variables and Functions:

*Block and Transaction Properties:*

block.blockhash(uint blockNumber) returns (bytes32): hash of the given block - only works for 256 most recent, excluding current, blocks - deprecated in version 0.4.22 and replaced by blockhash(uint blockNumber).

block.coinbase (address): current block miner’s address

block.difficulty (uint): current block difficulty

block.gaslimit (uint): current block gaslimit

block.number (uint): current block number

block.timestamp (uint): current block timestamp as seconds since unix epoch

gasleft() returns (uint256): remaining gas

msg.data (bytes): complete calldata

msg.gas (uint): remaining gas - deprecated in version 0.4.21 and to be replaced by gasleft()

msg.sender (address): sender of the message (current call)

msg.sig (bytes4): first four bytes of the calldata (i.e. function identifier)

msg.value (uint): number of wei sent with the message

now (uint): current block timestamp (alias for block.timestamp)

tx.gasprice (uint): gas price of the transaction

tx.origin (address): sender of the transaction (full call chain)

*Error Handling:*

assert(bool condition):

invalidates the transaction if the condition is not met - to be used for internal errors.

require(bool condition):

reverts if the condition is not met - to be used for errors in inputs or external components.

require(bool condition, string message):

reverts if the condition is not met - to be used for errors in inputs or external components. Also provides an error message.

revert():

abort execution and revert state changes

revert(string reason):

abort execution and revert state changes, providing an explanatory string

Mathematical and Cryptographic Functions:

addmod(uint x, uint y, uint k) returns (uint):

compute (x + y) % k where the addition is performed with arbitrary precision and does not wrap around at 2\*\*256. Assert that k != 0 starting from version 0.5.0.

mulmod(uint x, uint y, uint k) returns (uint):

compute (x \* y) % k where the multiplication is performed with arbitrary precision and does not wrap around at 2\*\*256. Assert that k != 0 starting from version 0.5.0.

keccak256(...) returns (bytes32):

compute the Ethereum-SHA-3 (Keccak-256) hash of the [(tightly packed) arguments](https://solidity.readthedocs.io/en/develop/abi-spec.html#abi-packed-mode)

sha256(...) returns (bytes32):

compute the SHA-256 hash of the [(tightly packed) arguments](https://solidity.readthedocs.io/en/develop/abi-spec.html#abi-packed-mode)

sha3(...) returns (bytes32):

alias to keccak256

ripemd160(...) returns (bytes20):

compute RIPEMD-160 hash of the [(tightly packed) arguments](https://solidity.readthedocs.io/en/develop/abi-spec.html#abi-packed-mode)

ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address):

recover the address associated with the public key from elliptic curve signature or return zero on error ([example usage](https://ethereum.stackexchange.com/q/1777/222))

*Address Related*

<address>.balance (uint256):

balance of the [Address](https://solidity.readthedocs.io/en/develop/types.html#address) in Wei

<address>.transfer(uint256 amount):

send given amount of Wei to [Address](https://solidity.readthedocs.io/en/develop/types.html#address), throws on failure, forwards 2300 gas stipend, not adjustable

<address>.send(uint256 amount) returns (bool):

send given amount of Wei to [Address](https://solidity.readthedocs.io/en/develop/types.html#address), returns false on failure, forwards 2300 gas stipend, not adjustable

<address>.call(...) returns (bool):

issue low-level CALL, returns false on failure, forwards all available gas, adjustable

<address>.callcode(...) returns (bool):

issue low-level CALLCODE, returns false on failure, forwards all available gas, adjustable

<address>.delegatecall(...) returns (bool):

issue low-level DELEGATECALL, returns false on failure, forwards all available gas, adjustable

*Contract Related:*

this (current contract’s type):

the current contract, explicitly convertible to [Address](https://solidity.readthedocs.io/en/develop/types.html#address)

selfdestruct(address recipient):

destroy the current contract, sending its funds to the given [Address](https://solidity.readthedocs.io/en/develop/types.html#address)

suicide(address recipient):

deprecated alias to selfdestruct

d) Expressions and Control Structures

*Input Parameters and Output Parameters*

Input Parameters:

pragma solidity ^0.4.16;

contract Simple {

function taker(uint \_a, uint \_b) public pure {

*// do something with \_a and \_b.*

}

}

Output Parameters:

pragma solidity ^0.4.16;

contract Simple {

function arithmetics(uint \_a, uint \_b)

public

pure

returns (uint o\_sum, uint o\_product)

{

o\_sum = \_a + \_b;

o\_product = \_a \* \_b;

}

}

*Control Structures:*

if, else, while, do, for, break, continue, return, ? : tương tự như C

*Returning Multiple Values*: return (v0, v1, ..., vn)

*Function Calls*

Internal Function Calls: gọi trực tiếp tên hàm

pragma solidity ^0.4.16;

contract C {

function g(uint a) public pure returns (uint ret) { return f(); }

function f() internal pure returns (uint ret) { return g(7) + f(); }

}

External Function Calls: cần khởi tạo hàm trước khi gọi

pragma solidity ^0.4.0;

contract InfoFeed {

function info() public payable returns (uint ret) { return 42; }

}

contract Consumer {

InfoFeed feed;

function setFeed(address addr) public { feed = InfoFeed(addr); }

function callFeed() public { feed.info.value(10).gas(800)(); }

}

Named Calls and Anonymous Function Parameters:

pragma solidity ^0.4.0;

contract C {

function f(uint key, uint value) public {

*// ...*

}

function g() public {

*// named arguments*

f({value: 2, key: 3});

}

}

Omitted Function Parameter Names:

pragma solidity ^0.4.16;

contract C {

*// omitted name for parameter*

function func(uint k, uint) public pure returns(uint) {

return k;

}

}

Creating Contracts via new: sử dụng new khởi tạo contract

pragma solidity ^0.4.0;

contract D {

uint x;

function D(uint a) public payable {

x = a;

}

}

contract C {

D d = new D(4); *// will be executed as part of C's constructor*

function createD(uint arg) public {

D newD = new D(arg);

}

function createAndEndowD(uint arg, uint amount) public payable {

*// Send ether along with the creation*

D newD = (new D).value(amount)(arg);

}

}

Destructuring Assignments and Returning Multiple Values:

pragma solidity >0.4.23 <0.5.0;

contract C {

uint[] data;

function f() public pure returns (uint, bool, uint) {

return (7, true, 2);

}

function g() public {

*// Variables declared with type and assigned from the returned tuple.*

(uint x, bool b, uint y) = f();

*// Common trick to swap values -- does not work for non-value storage types.*

(x, y) = (y, x);

*// Components can be left out (also for variable declarations).*

(data.length,,) = f(); *// Sets the length to 7*

*// Components can only be left out at the left-hand-side of assignments, with*

*// one exception:*

(x,) = (1,);

*// (1,) is the only way to specify a 1-component tuple, because (1) is*

*// equivalent to 1.*

}

}

*Error handling*: Assert, Require, Revert and Exceptions. Các hàm xử lý ngoại lệ

pragma solidity ^0.4.22;

contract Sharer {

function sendHalf(address addr) public payable returns (uint balance) {

require(msg.value % 2 == 0, "Even value required.");

uint balanceBeforeTransfer = this.balance;

addr.transfer(msg.value / 2);

*// Since transfer throws an exception on failure and*

*// cannot call back here, there should be no way for us to*

*// still have half of the money.*

assert(this.balance == balanceBeforeTransfer - msg.value / 2);

return this.balance;

}

}

4.Contracts

a) Giới thiệu

IDE: Remix

Function Modifiers: check điều kiện trước khi thực thi hàm

Constant State Variables: constant

Functions:

View Functions:

pragma solidity ^0.4.16;

contract C {

function f(uint a, uint b) public view returns (uint) {

return a \* (b + 42) + now;

}

}

Pure Functions:

pragma solidity ^0.4.16;

contract C {

function f(uint a, uint b) public pure returns (uint) {

return a \* (b + 42);

}

}

Fallback Function:

pragma solidity ^0.4.0;

contract Test {

*// This function is called for all messages sent to*

*// this contract (there is no other function).*

*// Sending Ether to this contract will cause an exception,*

*// because the fallback function does not have the `payable`*

*// modifier.*

function() public { x = 1; }

uint x;

}

*// This contract keeps all Ether sent to it with no way*

*// to get it back.*

contract Sink {

function() public payable { }

}

contract Caller {

function callTest(Test test) public {

test.call(0xabcdef01); *// hash does not exist*

*// results in test.x becoming == 1.*

*// The following will not compile, but even*

*// if someone sends ether to that contract,*

*// the transaction will fail and reject the*

*// Ether.*

*//test.send(2 ether);*

}

}

Function Overloading

Visibility and Getters: Functions có thể khai báo external, public, internal or private, mặc định public. Với state variables, external không có và mặc định internal.

* External: gọi từ bênh ngoài hoặc transactions
* Public: gọi từ internal hoặc via messages
* Internal: truy cập từ Internally
* Private: chỉ được truy cập từ contract định nghĩa

pragma solidity ^0.4.0;

contract C {

uint private data;

function f(uint a) private returns(uint b) { return a + 1; }

function setData(uint a) public { data = a; }

function getData() public returns(uint) { return data; }

function compute(uint a, uint b) internal returns (uint) { return a+b; }

}

contract D {

function readData() public {

C c = new C();

uint local = c.f(7); *// error: member `f` is not visible*

c.setData(3);

local = c.getData();

local = c.compute(3, 5); *// error: member `compute` is not visible*

}

}

contract E is C {

function g() public {

C c = new C();

uint val = compute(3, 5); *// access to internal member (from derived to parent contract)*

}

}

Events

pragma solidity ^0.4.0;

contract ClientReceipt {

event Deposit(

address indexed \_from,

bytes32 indexed \_id,

uint \_value

);

function deposit(bytes32 \_id) public payable {

*// Events are emitted using `emit`, followed by*

*// the name of the event and the arguments*

*// (if any) in parentheses. Any such invocation*

*// (even deeply nested) can be detected from*

*// the JavaScript API by filtering for `Deposit`.*

emit Deposit(msg.sender, \_id, msg.value);

}

}

Sử dụng JavaScript API:

var abi = */\* abi as generated by the compiler \*/*;

var ClientReceipt = web3.eth.contract(abi);

var clientReceipt = ClientReceipt.at("0x1234...ab67" */\* address \*/*);

var event = clientReceipt.Deposit();

*// watch for changes*

event.watch(function(error, result){

*// result will contain various information*

*// including the argumets given to the `Deposit`*

*// call.*

if (!error)

console.log(result);

});

*// Or pass a callback to start watching immediately*

var event = clientReceipt.Deposit(function(error, result) {

if (!error)

console.log(result);

});

Low-Level Interface to Logs:

pragma solidity ^0.4.10;

contract C {

function f() public payable {

bytes32 \_id = 0x420042;

log3(

bytes32(msg.value),

bytes32(0x50cb9fe53daa9737b786ab3646f04d0150dc50ef4e75f59509d83667ad5adb20),

bytes32(uint256(msg.sender)),

\_id

);

}

}

b)Tính chất

Inheritance(tính kế thừa)

pragma solidity ^0.4.22;

contract owned {

constructor() public { owner = msg.sender; }

address owner;

}

contract mortal is owned {

function kill() public {

if (msg.sender == owner) selfdestruct(owner);

}

}

contract Base1 is mortal {

function kill() public { */\* do cleanup 1 \*/* mortal.kill(); }

}

contract Base2 is mortal {

function kill() public { */\* do cleanup 2 \*/* mortal.kill(); }

}

contract Final is Base1, Base2 {

}

Constructors(khởi tạo)

pragma solidity ^0.4.22;

contract A {

uint public a;

constructor(uint \_a) internal {

a = \_a;

}

}

contract B is A(1) {

constructor() public {}

}

Arguments for Base Constructors

pragma solidity ^0.4.22;

contract Base {

uint x;

constructor(uint \_x) public { x = \_x; }

}

contract Derived1 is Base(7) {

constructor(uint \_y) public {}

}

contract Derived2 is Base {

constructor(uint \_y) Base(\_y \* \_y) public {}

}

Multiple Inheritance and Linearization

pragma solidity ^0.4.0;

contract X {}

contract A is X {}

contract C is A, X {}

Abstract Contracts( contracts từu tượng)

pragma solidity ^0.4.0;

contract Feline {

function utterance() public returns (bytes32);

}

contract Cat is Feline {

function utterance() public returns (bytes32) { return "miaow"; }

}

Interfaces: tương tự như Abstract Contracts nhưng không có triển khai các hàm

Libraries: có thể triển khai một thư viện qua từ khóa libary các contract có thể sử dụng

pragma solidity ^0.4.22;

library Set {

*// We define a new struct datatype that will be used to*

*// hold its data in the calling contract.*

struct Data { mapping(uint => bool) flags; }

*// Note that the first parameter is of type "storage*

*// reference" and thus only its storage address and not*

*// its contents is passed as part of the call. This is a*

*// special feature of library functions. It is idiomatic*

*// to call the first parameter `self`, if the function can*

*// be seen as a method of that object.*

function insert(Data storage self, uint value)

public

returns (bool)

{

if (self.flags[value])

return false; *// already there*

self.flags[value] = true;

return true;

}

function remove(Data storage self, uint value)

public

returns (bool)

{

if (!self.flags[value])

return false; *// not there*

self.flags[value] = false;

return true;

}

function contains(Data storage self, uint value)

public

view

returns (bool)

{

return self.flags[value];

}

}

contract C {

Set.Data knownValues;

function register(uint value) public {

*// The library functions can be called without a*

*// specific instance of the library, since the*

*// "instance" will be the current contract.*

require(Set.insert(knownValues, value));

}

*// In this contract, we can also directly access knownValues.flags, if we want.*

}