<1 과목 소프트웨어 설계>

1 애자일 방법론(Agile)

- 빠른 릴리즈를 통해 문제점을 빠르게 파악할 수 있다.
- 진화하는 요구사항을 수용하는데 적합하다.
- 소프트웨어가 잘 실행되는데 가치를 둔다.
- 계획을 따르기보다는 변화에 대응하는 것에 더 가치를 둔다.
- 고객과의 의사소통을 중요하게 생각한다.
- 고객과의 피드백을 중요하게 생각한다.
- 절차와 도구보다 개인과 소통을 중요하게 생각한다.
- 계약 협상보다는 고객과의 협업에 더 가치를 둔다.
- 프로세스의 도구보다는 개인과 상호작용에 더 가치를 둔다.
- ~~정확한 결과 도출을 위해 계획 수립과 문서화에 중점을 둔다.~~
  ~~실제 작동하는 소프트웨어보다는 이해하기 좋은 문서에 더 가치를 둔다.~~
  ~~계획에 중점을 두어 변경 대응이 난해하다.~~
- 예시) XP(eXtreme Programming), 스크럼(Scrum), 칸반(Kanban), 크리스탈(Crystal), 린(LEAN), 기능 주도 개발(FDD, Feature Driven Development), ~~모듈중심 개발~~

2 XP 기법

XP(eXtreme Programming)의 핵심 가치

- 용기(Courage), 의사소통(Communication), 피드백(Feedback), 존중(Respect), 단순성(Simplicity)

XP 의 기본원리

- Continuous Integration (지속적인 통합)
- Collective Ownership (공동 소유권)
- Pair Programming (짝 프로그래밍)
- Whole Team(전체 팀)
- Small Releases(소규모 릴리즈)
- Test-Driven Development(테스트 주도 개발)
- Design Improvement(디자인 개선)
- Refactoring(리팩토링)
- ~~Linear Sequential Method~~

XP 의 특징

- 사용자의 요구사항은 언제든지 변할 수있다.
- 고객과 직접 대면하며 요구사항을 이야기하기 위해 사용자 스토리(User Story)를 활용할 수 있다.

- 기존의 방법론에 비해 실용성(Pragmatism)을 강조한 것이라고 볼 수있다.
- ~~빠른 개발을 위해 테스트를 수행하지 않는다.~~

3 UML

UML(Unified Modeling Language)의 구성 요소

- 사물(Things), 관계(Relationship), 다이어그램(Diagram)

1) 사물(Things)

- 구조 사물, 행동 사물, 그룹 사물, 주해 사물

2) 관계(Relationships)

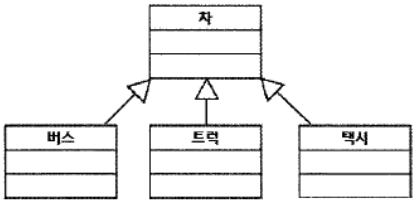
2-1) 연관 관계(Association)

- 두 사물간의 구조적 관계로 어느 한 사물 객체가 다른 사물 객체와 연결되어 있음 ('has-a')관계
- 기호 ㅡ로 표시

2-2) 일반화 관계(Generalization)

- 일반화된 사물과 좀 더 특수화된 사물 사이의 관계
- 일반적인 개념을 상위(부모), 구체적인 개념을 하위(자식)이라고 함 ('is-a')관계
- 기호 ㅡ▷로 표시

ex. 아래의 UML 모델에서 '차' 클래스와 각 클래스의 관계는? 일반화 관계



2-3) 의존 관계(Dependency)

- 한 사물의 명세가 바뀌면 다른 사물에 영향을 준다.
- 일반적으로 한 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우에 나타나는 관계
- 기호 -->로 표시

2-4) 실체화 관계(Realization)

- 한 객체가 다른 객체에게 오퍼레이션을 수행하도록 지정하는 의미적 관계이다.
- 기호 --▷로 표시

3) 다이어그램

3-1) 정적 다이어그램(Structural Diagram) => 구조적 (Structural)

- **클래스 다이어그램(Class Diagram)**
  - 시스템 내 클래스의 정적 구조를 표현하고 클래스와 클래스, 클래스의 속성 사이의 관계를 나타낸다.
- **객체 다이어그램(Object Diagram)**
- **컴포넌트 다이어그램(Component Diagram)**
- **배치 다이어그램(Deployment Diagram)**
- **복합체 구조 다이어그램(Composite Structure Diagram)**
- **패키지 다이어그램(Package Diagram)**

3-2) 동적 다이어그램(Behavioral Diagram) => 행위 (Behavioral)

● **유스케이스 다이어그램(Use Case Diagram)**

- 사용자의 요구를 추출하고 분석하기 위해 주로 사용한다.
- 연동의 개념은 양방향으로 데이터를 파일이나 정해진 형식으로 넘겨주는 것이다
- 액터는 대상 시스템과 상호 작용하는 사람이나 다른 시스템에 의한 역할이다.
- 시스템 액터는 본 시스템과 데이터를 주고받는 연동 시스템을 의미한다.
- 사용자 액터는 기능을 요구하는 대상이나 시스템의 수행결과를 통보받는 사용자 혹은 기능을 사용하게 될 대상으로 시스템이 제공해야하는 기능인 유스케이스의 권한을 가지는 대상, 역할이다.

***유스케이스 구성요소와의 관계**

- 연관 : 유스케이스와 액터의 관계
- **확장** : 기본 유스케이스 수행 시 특별한 조건을 만족할 때 수행하는 유스케이스
- 포함 : 시스템의 기능이 별도의 기능을 포함
- 일반화 : 하위 유스케이스(액션)이 상위 유스케이스(액터)에게 기능/역할을 상속받음
- 그룹화 : 여러개의 유스케이스를 단순화하는 방법

● **시퀀스(순차) 다이어그램(Sequence Diagram)**

- 객체들의 상호 작용을 나타내기 위해 사용한다.
- 시간의 흐름에 따라 객체들이 주고 받는 메시지의 전달 과정을 강조한다.
- 교류 다이어그램(Interaction Diagram)의 한 종류로 볼 수 있다.
- 정적 다이어그램보다 동적 다이어그램에 가깝다.

***순차 다이어그램의 구성 항목 # 액객생메실**

- 액터(Actor) : 시스템으로부터 서비스를 요청하는 외부요소로, 사람이나 외부 시스템 의미
- 객체(object) : 메시지를 주고받는 주체
- 생명선(Life line) : 객체가 메모리에 존재하는 기간으로, 객체 아래쪽에 점선을 그어 표현
- 메시지(Message) : 객체가 상호 작용을 위해 주고받는 메시지
- 실행 상자(Active Box) : 객체가 메시지를 주고받으며 구동되고 있음을 표현
- 확장

● **액티비티(활동) 다이어그램 (Activity Diagram)**

- 시스템이 어떤 기능을 수행하는지 객체의 처리로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현한다.

● **상태 다이어그램(State Diagram)**

- 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 상태 변화를 표현한다.

● **상호작용 개요 다이어그램(Interaction Overview Diagram)**

● **커뮤니케이션 다이어그램(Communication Diagram)**

● **타이밍 다이어그램(Timing Diagram)**

***UML 에 대한 설명 중 옳지 않은 것은?**

- 기능적 모델은 사용자 측면에서 본 시스템 기능이며, 유스케이스 다이어그램을 사용한다.
- 정적 모델은 객체, 속성, 연관관계, 오퍼레이션의 시스템의 구조를 나타내며, 클래스 다이어그램을 사용한다.
- 동적 모델은 시스템의 내부 동작을 말하며, UML 에서는 시퀀스, 상태, 액티비티 다이어그램을 사용한다.
- ~~상태 다이어그램은 객체들 사이의 메시지 교환을 나타내며, 시퀀스 다이어그램은 하나의 객체가 가진 상태와 그 상태의 변화에 의한 동작순서를 나타낸다.~~

***컴포넌트 다이어그램과 배치 다이어그램**은 구현 단계에서 사용되는 다이어그램이다.

***UML 확장 모델에서 스테레오 타입 객체를 표현할 때 사용하는 기호** : 《 》

4 UI (User Interface)

UI 의 구분

- CLI (Command Line Interface) : 대표적으로 DOS 및 Unix 등의 운영체제에서 조작을 위해 사용하던 것으로, 정해진 명령문자열을 입력하여 시스템을 조작하는 사용자 인터페이스
- GUI (Graphical User Interface) : 마우스로 선택해 작업을 하는 그래픽 환경의 인터페이스
- NUI (Natural User Interface) : 사용자의 말이나 행동으로 기기를 조작하는 인터페이스
- VUI (Voice User Interface) : 사람의 음성으로 기기를 조작하는 인터페이스
- OUI (Organic User Interface) : 모든 사물과 사용자 간의 상호작용을 위한 인터페이스

UI 의 기본 원칙

- 직관성 : 누구나 쉽게 이해하고 사용할 수 있어야 한다.
- 유효성 : 사용자의 목적을 정확하고 완벽하게 달성해야 한다.
- 학습성 : 누구나 쉽게 배우고 익힐 수 있어야 한다.
- 유연성 : 사용자의 요구사항을 최대한 수용하고 실수를 최소화해야 한다.

UI 의 특징

- 사용자와 시스템이 정보를 주고받는 상호작용이 잘 이루어지도록 하는 장치나 소프트웨어를 의미한다.
- 편리한 유지보수를 위해 사용자 중심으로 설계되어야 한다.
- 사용자 중심의 상호 작용이 되도록 한다.
- 배우기가 용이하고 쉽게 사용할 수 있도록 만들어져야 한다.
- 사용자 요구사항이 UI 에 반영될 수 있도록 구성해야 한다.
- 구현하고자 하는 결과의 오류를 최소화한다.
- 사용자의 편의성을 높임으로써 작업시간을 단축시킨다.
- 막연한 작업 기능에 대해 구체적인 방법을 제시하여 준다.

5 CASE (Computer-Aided Software Engineering)

- 소프트웨어 개발 과정의 일부 또는 전체를 **자동화하기 위한 도구**이다.
- 소프트웨어 모듈의 재사용성이 향상된다.
- 표준화된 개발 환경 구축 및 문서 자동화 기능을 제공한다.
- 자동화된 기법을 통해 소프트웨어 품질이 향상된다.
- 작업 과정 및 데이터 공유를 통해 작업자간 커뮤니케이션을 증대한다.
- 소프트웨어 유지보수를 간편하게 수행할 수 있다.
- 소프트웨어 개발 과정에서 사용되는 요구분석, 설계, 구현, 검사, 디버깅 과정 전체 또는 일부를 컴퓨터와 전용의 소프트웨어 도구를 사용하여 자동화하는 것이다.

~~- 소프트웨어 사용자들에게 사용 방법을 신속히 숙지시키기 위해 사용된다.~~

~~- 2000 년대 이후 소개되었으며, 객체지향 시스템에 한해 효과적으로 활용된다. => 1980 년대, 모든 분야에 적용~~

CASE 의 주요 기능

- **그래픽 지원**
- **소프트웨어 생명주기 전 단계의 연결**
- **다양한 소프트웨어 개발 모형 지원**
- ~~**언어 번역**~~
- 모델들 사이의 모순검사
- 오류검증
- 자료흐름도 등 다이어그램 작성
- 시스템 문서화 및 명세화를 위한 그래픽 지원

CASE 의 원천 기술

- 자동프로그래밍 기술
- 분산 처리 기술
- **구조적 기법**
- **프로토타이핑 기술**
- **정보 저장소 기술**
- ~~**일괄처리 기술**~~

CASE 도구의 분류

1) 상위 CASE 도구 : 요구분석, 설계 단계를 지원
   - **모델들 사이의 모순 검사 기능**
   - **모델의 오류검증 기능,**
   - **자료 흐름도 작성 기능**
2) 하위 CASE 도구 : 코드를 작성하고 테스트하며 문서화하는 과정 지원
   - 시스템 명세서
   - **전체 소스코드 생성 기능**

6 요구사항 개발 프로세스

요구사항 개발 프로세스

- 도출/추출(Elicitation) → 분석(Analysis) → 명세(Specification) → 확인(Validation)/검증(Valification)

① 요구사항 추출(Requirement Elicitation)

- 프로젝트 계획 단계에 정의한 문제의 범위 안에 있는 사용자의 요구를 찾는 단계이다.

② 요구사항 분석(Requirement Analysis)

- 소프트웨어 개발의 실제적인 첫 단계로 사용자의 요구에 대해 이해하는 단계라 할 수 있다.
- 도메인 분석(Domain Analysis)은 요구에 대한 정보를 수집하고 배경을 분석하여 이를 토대로 모델링을 한다.
- 분석 결과의 문서화를 통해 향후 유지보수에 유용하게 활용 할 수 있다.
- 자료흐름도, 자료 사전 등이 효과적으로 이용될 수 있다.
- 보다 구체적인 명세를 위해 소단위 명세서(Mini-Spec)가 활용될 수 있다.
- ~~개발 비용이 가장 많이 소요되는 단계이다.~~
- ~~기능적(Functional) 요구에서 시스템 구축에 대한 성능, 보안, 품질, 안정 등에 대한 요구사항을 도출한다.~~
- 비용과 일정에 대한 제약설정
- 타당성 조사

- 요구사항 정의 문서화
- ~~설계 명세서 작성~~

기능적 요구사항, 비기능적 요구사항

- 기능적 요구사항 : 시스템에서 제공해야 할 기능을 정의한 것
    - 입출력기능, 데이터베이스 기능, 통신 기능 등
- 비기능적 요구사항 : 시스템이 가져야 하는 기능 이외의 요구사항
    - 시스템의 전체적인 품질이나 고려해야 하는 제약사항 등
    - 사용 용이성, 효율성, 신뢰성, 이식성, 유연성, 확장성
    - 성능적인 면: 응답 속도, 자원 사용량 등
    - 보안 측면: 침입 대응, 침입 탐지, 사용자 인증, 권한 부여 등

요구사항 분석 시에 필요한 기술

- 청취과 인터뷰 질문 기술
- 분석과 중재기술
- 관찰 및 모델 작성 기술
- ~~설계 및 코딩 기술~~

요구사항 분석이 어려운 이유

- 개발자와 사용자 간의 지식이나 표현의 차이가 커서 상호 이해가 쉽지 않다.
- 사용자의 요구는 예외가 많아 열거와 구조화가 어렵다.
- 사용자의 요구사항이 모호하고 불명확하다.
- 소프트웨어 개발 과정 중에 요구사항이 계속 변할 수 있다.

③ 요구사항 명세(Requirements Specification)

- 비정형 명세기법은 사용자의 요구를 표현할 때 자연어를 기반으로 서술한다.
- ~~비정형 명세기법은 사용자의 요구를 표현할 때 Z 비정형 명세기법을 사용한다.~~
- 정형 명세기법은 사용자의 요구를 표현할 때 수학적인 원리와 표기법을 이용한다.
- 정형 명세기법은 비정형 명세기법에 비해 표현이 간결하다.

***정형 명세기법(수학적 기반) :** Z, VDM, Petri-Net, CSP, CCS, LOTOS

***비정형 명세기법(자연어 기반) :** FSM, Decision Table, ER 모델링, State chart(SADT), UseCase

④ 요구사항 검증(Requirements Validation)

- 요구사항이 고객이 정말 원하는 시스템을 제대로 정의하고 있는지 점검하는 과정이다.
- 개발완료 이후에 문제점이 발견될 경우 막대한 재작업 비용이 들 수 있기 때문에 매우 중요하다.
- 요구사항이 실제 요구를 반영하는지, 문서상의 요구사항은 서로 상충되지 않는지 등을 점검한다.
- ~~요구사항 검증 과정을 통해 모든 요구사항 문제를 발견할 수 있다~~
- 인터페이스 요구사항 검토 계획 수립 → 검토 및 오류 수정 → 베이스라인 설정

요구사항 검증 방법

- **동료 검토(Peer Review)**
    - 요구 사항 명세서 작성자가 요구 사항 명세서를 설명하고 이해관계자들이 설명을 들으면서 결함을 발견
- **인스펙션(Walk Through)**
    - 요구사항 명세서 작성자를 제외한 다른 검토 전문가들이 확인하면서 결함을 발견

- **워크 스루(Inspection)**
  - 검토회의 전에 요구사항 명세서를 미리 배포하여 사전 검토 후, 짧은 검토 회의를 통해 오류를 조기 검출

***다음 중 요구사항 모델링에 활용되지 않은 것은?**

- 애자일(Agile) 방법
- 유스케이스 다이어그램(Use Case Diagram)
- 시퀀스 다이어그램(Sequence Diagram)
- ~~단계 다이어그램(Phase Diagram)~~

***요구 사항 정의 및 분석·설계의 결과물을 표현하기 위한 모델링 과정에서 사용되는 다이어그램이 아닌 것은?**

- Data Flow Diagram
- UML Diagram
- E-R Diagram
- ~~AVL Diagram~~

***요구사항 관리 도구의 필요성이 아닌 것은?**

- 요구사항 변경으로 인한 비용 편익 분석
- 요구사항 변경의 추적
- 요구사항 변경에 따른 영향 평가
- ~~기존 시스템과 신규 시스템의 성능 비교~~

7 미들웨어(Middleware)

- 분산 컴퓨팅 환경에서 서로 다른 기종 간의 하드웨어나 프로토콜, 통신환경 등을 연결하여 응용프로그램과 운영환경 간에 원만한 통신이 이루어질 수 있게 서비스를 제공하는 소프트웨어
- 분산 시스템에서 다양한 부분을 관리하고 통신하며 데이터를 교환하게 해주는 소프트웨어로 볼 수 있다.
- 위치 투명성(Location Transparency)을 제공한다.
- 분산 시스템의 여러 컴포넌트가 요구하는 재사용 가능한 서비스의 구현을 제공한다.
- 애플리케이션과 사용자 사이 외에도 프로그램과 환경 간에서 분산서비스를 제공한다.
- 클라이언트와 서버 간의 통신을 담당하는 시스템 소프트웨어
- ~~애플리케이션과 사용자 사이에서만 분산서비스를 제공한다.~~

DB(Database)

- 클라이언트에서 원격의 데이터베이스와 연결하기 위한 미들웨어

WAS(Web Application Server, 앱 애플리케이션 서버)

- 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어
- ex. Web Logic, **WebSphere, JEUS, Tomcat, ~~Web Server~~**

RPC(Remote Procedure Call, 원격 프로시저 호출)

- 응용 프로그램의 프로시저를 사용해 원격 프로시저를 로컬 프로시저처럼 호출하는 방식의 미들웨어

TP Monitor(Transaction Processing Monitor, 트랜잭션 처리 모니터)

- 트랜잭션이 올바르게 처리되고 있는지 데이터를 감시하고 제어하는 미들웨어

- 사용자 수가 증가해도 빠른 응답 속도를 유지해야 하는 업무에 주로 사용됨

MOM(Message Oriented Middleware, 메시지 지향 미들웨어)
- 메시지 기반의 비동기형 메시지를 전달하는 방식의 미들웨어

Legacyware(레거시웨어)
- 기존 애플리케이션에 새로운 업데이트된 기능을 덧붙이고자 할 때 사용되는 미들웨어

ORB(Object Request Broker, 객체 요청 브로커)
- 객체 지향 미들웨어로 **코바(CORBA)** 표준 스펙을 구현한 미들웨어
- 객체 간 메시지 전달을 지원하는 미들웨어
- 코바(CORBA) : 네트워크에서 분산 프로그램 객체를 생성, 배포, 관리하기 위한 규격을 의미

8 디자인 패턴

디자인 패턴을 목적(Purpose)으로 분류할 때 **생성, 구조, 행위**로 분류할 수 있다.

디자인 패턴 사용의 장단점
- 개발자들 사이의 의사소통을 원활하게 할 수 있다.
- 소프트웨어의 품질과 생산성을 향상시킬 수 있다.
- 소프트웨어 구조 파악이 용이하다.
- 재사용을 위한 개발 시간이 단축된다.
- ~~개발 프로세스를 무시할 수 있다.~~
- ~~절차형 언어와 함께 이용될 때 효율이 극대화된다.~~

1) 생성 패턴(Creational Pattern) #추빌팩프싱

● **추상 팩토리 패턴(Abstract Factory)**
- 서로 연관, 의존하는 객체들을 그룹으로 생성해 추상적으로 표현한다.

● **빌더 패턴(Builder)**
- 작게 분리된 인스턴스를 건축하듯이 조합하여 객체를 생성한다

● **팩토리 메소드 패턴(Factory Method)**
= Virtual-Constructor 패턴 (가상 생성자)
- 상위클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위클래스에서 인스턴스를 생성한다.
- 객체를 생성하기 위한 인터페이스를 정의하여, 어떤 클래스가 인스턴스화 될 건지는 서브클래스가 결정한다.

● **프로토타입 패턴(Prototype)**
- prototype 을 먼저 생성하고 인스턴스를 복제하여 사용하는 구조이다.

● **싱글톤 패턴(Singleton)**
- 특정 클래스의 인스턴스가 오직 하나임을 보장하고, 이 인스턴스에 대한 접근 방법을 제공한다.

2) 구조 패턴(Structural Pattern) # 어브컴데퍼플프

● **어댑터 패턴(Adapter)**
- 기존에 구현되어 있는 클래스에 기능 발생 시 기존 클래스를 재사용할 수 있도록 중간에서 맞춰준다.

● **브리지 패턴(Bridge)**
- 구현부에서 추상층을 분리하여, 독립적으로 확장이 가능하게 하는 패턴
- 기능과 구현을 두 개의 별도 클래스로 구현함

● **컴포지트 패턴(Composite)**
- 여러 객체를 가진 복합, 단일 객체를 구분 없이 다룰 때 사용하는 패턴

● **데코레이터 패턴(Decorator)**
- 상속을 사용하지 않고도 객체의 기능을 동적으로 확장해주는 패턴

● **퍼싸드 패턴(Façade)**
- 서브 클래스들의 기능을 간편하게 사용할 수 있도록 하는 패턴; ex) 리모컨

● **플라이웨이트 패턴(Flyweight)**
- 공유해서 사용함으로써 메모리를 절약하는 패턴

● **프록시 패턴(Proxy)**
- 접근이 어려운 객체를 연결해주는 인터페이스 역할을 수행하는 패턴

3) 행위 패턴(Behavioral Pattern) : 행위의 변경, 수정 등을 위한 패턴
- 클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법을 정의한다.

● **책임 연쇄 패턴(Chain of Responsibility)**
- 한 객체가 처리하지 못하면 다음 객체로 넘어가는 패턴

● **커맨드 패턴(Command)**
- 요청에 사용되는 각종 명령어들을 추상, 구체 클래스로 분리하여 단순화함

● **인터프리터 패턴(Interpreter)**
- 언어에 문법 표현을 정의하는 패턴

● **반복자 패턴(Iterator)**
- 동일한 인터페이스를 사용하도록 하는 패턴

● **중재자 패턴(Mediator)**
- 객체간의 통제와 지시의 역할을 하는 **중재자**를 두어 객체지향의 목표를 달성하게 해준다.

● **메멘토 패턴(Memento)**
- 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴

● **옵저버 패턴(Observer)**
- 관찰대상의 변화를 탐지하는 패턴
- 한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달
- 분산된 시스템 간에 이벤트 생성, 발행(Publish), 이를 수신(Subscribe)해야 할 때 이용함

● **상태 패턴(State)**
- 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴

● **전략 패턴(Strategy)**
- 클라이언트에 영향을 받지 않는 독립적인 알고리즘을 선택하는 패턴

● **템플릿 메소드 패턴(Template Method)**
- 유사한 서브 클래스를 묶어 공통된 내용을 상위 클래스에 정의하는 패턴

● **방문자 패턴(Visitor)**
- 필요할 때마다 해당 클래스에 방문해서 처리하는 패턴
- 각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성한다.
- 분리된 처리 기능은 각 클래스를 방문하여 수행

9 객체지향

객체(Object)

- 실세계에 존재하거나 생각할 수 있는 것을 말한다.

클래스(Class)

- 하나 이상의 유사한 객체들을 묶어서 하나의 공통된 특성을 표현한 것이다.
- 객체지향 프로그램에서 데이터를 추상화하는 단위

인스턴스(Instance)

- 같은 클래스에 속한 각각의 객체를 의미한다.

메서드(Method)

- 클래스로부터 생성된 객체를 사용하는 방법

메시지(Message)

- 객체에게 어떤 행위를 하도록 지시하는 명령

캡슐화(encapsulation)

- 연관된 데이터와 함수를 함께 묶어 외부와 경계를 만들고 필요한 인터페이스만을 밖으로 드러낸다.
- 객체지향에서 **정보 은닉**과 가장 밀접한 관계가 있는 것
- 인터페이스가 단순화 된다.
- 소프트웨어 재사용이 높아진다.
- 변경 발생 시 오류의 파급효과가 적다.
- 데이터와 데이터를 처리하는 함수를 하나로 묶는 것이다.
- 캡슐화된 객체의 세부 내용이 외부에 은폐되어 변경이 발생하게되 오류의 파급 효과가 적다.
- 인터페이스가 단순해지고 객체 간의 결합도가 낮아진다.
- 캡슐화된 객체들은 **재사용이 용이해진다**.

정보 은닉(Information Hiding)

- 객체가 가지고 있는 속성과 오퍼레이션의 일부를 감추어서 객체의 외부에서는 접근이 불가능하다
- 필요하지 않은 정보는 접근할 수 없도록 하여 한 모듈 또는 하부시스템이 다른 모듈의 구현에 영향을 받지 않게 설계되는것을 의미한다.
- 모듈들 사이의 독립성을 유지시키는 데 도움이 된다.
- 설계에서 은닉되어야 할 기본 정보로는 IP 주소와 같은 물리적 코드, 상세 데이터 구조 등이 있다.
- ~~모듈 내부의 자료 구조와 접근 동작들에만 수정을 국한하기 때문에 요구사항 등 변화에 따른 수정이 불가능하다. =>~~ 모듈 변경 시 영향을 받지 않아 수정, 시험, 유지보수 용이

상속(Inheritance)

- 상위클래스에서 속성이나 연산을 전달받아 새로운 형태의 클래스로 확장하여 사용하는 것을 의미한다.
- 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려받는 것을 의미한다.

다형성(Polymorphism)

- 상속받은 여러 개의 하위 객체들이 다른 형태의 특성을 갖는 객체로 이용될 수 있는 성질이다.

일반화(Generalization)

- 공통 성질을 상위 객체로 정의하고, 특수화된 객체들을 하위의 부분형 객체로 정의하는 추상화 방법이다.
- is a : 클래스들 간의 개념적인 포함 관계

집단화(Aggregation)

- 클래스들 사이의 '부분-전체(**part**-whole)' 또는 '부분(is-a-**part**-of)'의 관계로 설명되는 연관성을 나타낸다.
- 클래스 간의 구조적인 집약 관계 "클래스 A 는 클래스 B 와 클래스 C 로 구성된다"
- 서로 관련 있는 여러 개의 객체를 묶어 한 개의 상위 객체를 만드는 것이다.

추상화(Abstraction)

- 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화 시키는 것

***소프트웨어 설계에 사용되는 대표적인 3 가지 추상화 기법**

- **제어 추상화, 과정 추상화, 자료 추상화, ~~강도 추상화~~**

객체지향 설계 원칙 #SOLID

● **SRP, 단일 책임 원칙(Single Responsibility Principle)**

- 객체는 단 하나의 책임만 가져야 한다.

● **OCP, 개방-폐쇄 원칙(Open-Closed Principle)**

- 기존의 코드를 변경하지 않으면서 기능을 추가할 수 있도록 설계가 되어야 한다.
- 확장에 대해 열려 있어야 하고, 수정에 대해서는 닫혀 있어야 한다

● **LSP, 리스코프 치환 원칙(Liskov Substitution Principle)**

- 서브타입(상속받은 하위 클래스)은 어디에서나 자신의 기반타입(상위클래스)으로 교체할 수 있어야 한다.

● **ISP, 인터페이스 분리 원칙(Interface Segregation Principle)**

- 클라이언트는 자신이 사용하지 않는 메서드와 의존관계를 맺으면 안된다.
- 클라이언트가 사용하지 않는 인터페이스 때문에 영향을 받아서는 안된다.

● **DIP, 의존 역전 원칙(Dependency Inversion Principle)**

- 의존 관계를 맺을 때, 변화하기 쉬운 것보다 변화하기 어려운 것에 의존해야 한다.

객체지향 분석 방법론

1) Coad 와 Yourdon 방법

- **E-R 다이어그램**을 사용하여 객체의 행위를 데이터 모델링하는데 초점을 둔 방법이다.
- 객체 식별, 구조 식별, 주체 정의, 속성 및 관계 정의, 서비스 정의 등의 과정으로 구성되는 것

2) Booch 방법(부치)

- **미시적** 개발 프로세스와 **거시적** 개발 프로세스를 모두 사용하는 방법이다.

3) Jacobson 방법

- **Use-Case** 를 강조하여 사용하는 방법이다.

4) Wirfs-Brocks 방법

- 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 분석 방법

객체지향 분석 기법의 특징

- 동적 모델링 기법이 사용될 수 있다.
- 데이터와 행위를 하나로 묶어 객체를 정의내리고 추상화시키는 작업이다.
- 소프트웨어를 개발하기 위한 비즈니스(업무)를 객체/속성, 클래스/멤버, 전체/부분 등으로 나눠 분석한다.

- 코드 재사용에 의한 프로그램 생산성 향상 및 요구에 따른 시스템의 쉬운 변경이 가능하다.
- ~~기능 중심으로 시스템을 파악하며 순차적인 처리가 중요시되는 하향식(Top-down)방식으로 볼 수 있다.~~

10 럼바우의 객체지향 분석 기법

- 객체 모델링 : **객체 다이어그램**(정보 모델링)
- 동적 모델링 : **상태 다이어그램**(상태도)
- 기능 모델링 : **자료 흐름도(DFD)**

11 DFD (자료 흐름도, Data Flow Diagram)

자료 흐름도의 구성요소

- 프로세스(Process) => 원으로 표시
- 자료 흐름(Data Flow) => 화살표로 표시
- 자료 저장소(Data Store) => 평행선으로 표시
- 단말(Terminator) => 사각형으로 표시

자료 흐름도의 특징

- 자료 흐름 그래프 또는 버블(bubble) 차트라고도 한다.
- 구조적 분석 기법에 이용된다.
- 시간 흐름을 명확하게 표현할 수 없다.
- 자료 흐름과 기능을 자세히 표현하기 위해 단계적으로 세분화된다.
- DFD 의 요소는 화살표, 원, 사각형, 직선(단선/이중선)으로 표시한다.

12 자료사전 기호

= : 자료의 정의 (**is composed of**)
+ : 자료의 연결 (**and**)
( ) : 자료의 생략 (**optional**)
[ ] : 자료의 **선택**. (**or**)
{ } : 자료의 반복 **(iteration of)**
* * : 자료의 설명(주석, **comment**)

13 소프트웨어 설계

바람직한 소프트웨어 설계 지침

- 모듈의 기능을 예측할 수 있도록 정의한다.
- 이식성을 고려한다.
- 적당한 모듈의 크기를 유지한다.
- 가능한 모듈을 독립적으로 생성하고 결합도를 최소화한다.
- 적당한 모듈의 크기를 유지한다.
- 모듈 간의 접속 관계를 분석하여 복잡도와 중복을 줄인다.
- 모듈 간의 효과적인 제어를 위해 설계에서 계층적 자료 조직이 제시되어야 한다.
- 모듈 간의 결합도는 약할수록 바람직하다. (응집도는 강할수록 좋다)

소프트웨어의 상위 설계 : 최하위 수준에서 각각의 모듈들을 설계하고, 모듈이 완성되면 이들은 결합하여 검사
- 데이터 설계
- 시스템 분할
- 아키텍처 설계
- 인터페이스 정의
- 사용자 인터페이스 설계

소프트웨어의 하위 설계 : 소프트웨어 설계시 제일 상위에 있는 main user function 에서 시작하여 기능을 하위 기능들로 분할해 가면서 설계하는 방식
- 모듈 설계
- 인터페이스 작성

***소프트웨어 아키텍처 설계에서 시스템 품질 속성이 아닌 것은?**
- 가용성 (Availability)
- 변경 용이성 (Modifiability)
- 사용성(Usability)
- ~~독립성 (Isolation)~~

***소프트웨어의 사용자 인터페이스 개발 시스템(User Interface Development System)의 기능이 아닌 것은?**
- 사용자 입력의 검증
- 에러 처리와 에러 메시지 처리
- 도움과 프롬프트(prompt) 제공
- ~~소스 코드 분석 및 오류 복구~~

***소프트웨어 설계시 구축된 플랫폼의 성능특성 분석에 사용되는 측정 항목이 아닌 것은?**
- 응답시간(Response Time)
- 가용성(Availability)
- 사용률(Utilization)
- ~~서버 튜닝(Server Tuning)~~

***소프트웨어 공학에서 모델링 (Modeling)의 특징이 아닌 것은?**
- 개발팀이 응용문제를 이해하는 데 도움을 줄 수 있다.
- 개발될 시스템에 대하여 여러 분야의 엔지니어들이 공통된 개념을 공유하는 데 도움을 준다.
- 절차적인 프로그램을 위한 자료 흐름도는 프로세스 위주의 모델링 방법이다.
- ~~유지보수 단계에서만 모델링 기법을 활용한다.~~

***소프트웨어 프로젝트 계획 수립 시 소프트웨어 영역(software scope) 결정사항에 기술될 주요사항이 아닌 것은?**
- 기능
- 제약조건
- 인터페이스
- ~~인적자원~~

14 소프트웨어 아키텍처 패턴

1) 레이어 패턴(Layers Pattern)

- 계층 모델이라고도 한다. (ex. OSI 7 계층)

2) 클라이언트-서버 패턴(Client-Server Pattern)

- 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴
- 클라이언트나 서버는 요청과 응답을 받기 위해 동기화 되는 경우를 제외하고는 서로 독립적이다.

3) 모델-뷰-컨트롤러 패턴(Model-View-Controller Pattern)

- 3 개의 서브시스템(모델, 뷰, 제어)으로 구성되어 있다.

4) 파이프 필터 패턴(Pipe-Filter Pattern)

- 데이터는 파이프를 통해 **단방향**으로 흐르며, 필터 이동 시 **오버헤드**가 발생할 수 있다.
- **서브시스템이 입력 데이터를 받아 처리하고 결과를 다른 시스템에 보내는 작업이 반복한다.**
- **ex. UNIX 의 쉘(Shell)**

5) 마스터-슬레이브 구조(Master-Slave Pattern)

- 일반적으로 **실시간** 시스템에서 사용된다.
- 마스터 프로세스는 일반적으로 연산, 통신, 조정을 책임진다.
- 마스터 프로세스는 슬레이브 프로세스들을 제어할 수 있다.
- ~~슬레이브 프로세스는 데이터 수집 기능을 수행할 수 없다.~~
- ex. 장애 허용 시스템, 병렬 컴퓨팅 시스템

6) 브로커 구조(Broker Pattern)

- 컴포넌트와 사용자를 연결해주는 패턴
- ex. 분산 환경 시스템

7) 피어-투-피어 구조(Peer-To-Peer Pattern)

- 피어를 하나의 컴포넌트로 간주한다.
- 각 피어는 서비스를 호출하는 클라이언트가 될 수도, 서비스를 제공하는 서버가 될 수도 있는 패턴
- ex. 멀티스레딩(Multi Threading) 방식 사용

8) 이벤트-버스 구조(Event-Bus Pattern)

- 소스가 특정 채널에 이벤트 메시지를 발행하면, 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리하는 방식
- 이벤트를 생성하는 소스(Source), 이벤트를 수행하는 리스너(Listener), 이벤트의 통로인 채널(Channel), 채널들을 관리하는 버스(Bus)

9) 블랙보드 구조(Blackboard Pattern)

- 해결책이 명확하지 않은 문제를 처리하는데 유용한 패턴
- ex. 음성인식, 차량 식별, 신호 해석

10) 인터프리터 구조(Interpreter Pattern)

- 특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트를 설계할 때 사용

15 코드

코드의 기본 기능 : 배열, 분류, 식별, 표준화, ~~복잡성,~~ 간소화, 연상, 암호화, 오류 검출 기능

코드의 분류

1) 순차 코드(Sequence Code, 일련 번호 코드)
- 코드 설계에서 일정한 일련번호를 부여하는 방식
- ex) 1, 2, 3, 4, ...

2) 블록 코드(Block Code, 구분 코드)
- 공통성이 있는 것끼리 블록으로 구분하고, 각 블록 내에서 일련번호를 부여하는 방법
- 코드화 대상을 미리 파악하여 블록으로 구분한 후 그 안에서 순서대로 코드를 부여
- ex) 1001~1100: 총무부, 1101~1200: 영업부

3) 10 진 코드(Decimal Code, 도서 분류식 코드)
- 0~9 까지 10 진 분할하고, 다시 각각에 대해 10 진 분할하는 방법을 필요한 만큼 반복하는 방법
- 코드화 대상물을 일정한 소속으로 구분하여 십진수 한 자리씩 구분하여 대분류하고, 같은 방법으로 중분류, 소분류한 코드
- ex) 1000: 공학, 1100: 소프트웨어 공학, 1110: 소프트웨어 설계

4) 그룹 분류 코드(Group Classification Code)
- 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고, 각 그룹 안에서 일련번호를 부여하는 방법
- 구분 코드를 세분화한 형태로 대분류, 중분류, 소분류 등 각 분류별로 자릿수를 구성
- ex) 1-01-001: 본사-총무부-인사계, 2-01-001: 지사-총무부-인사계

5) 연상 코드(Mnemonic Code, 기호 코드)
- 명칭이나 약호와 관계있는 **숫자나 문자, 기호**를 이용하여 코드를 부여하는 방법
- 숫자나 문자를 조합해서 나타내는 것으로 어떤 내용을 기억할 수 있도록 표시한 기호 코드
  ex) TV-40: 40 인치 TV, L-15-220: 15W 220V 램프

6) 표의 숫자 코드(Significant Digit Code, 유효 숫자 코드)
- 코드화 대상 항목의 중량, 면적, 용량 등의 **물리적 수치**를 이용하여 만든 코드
- ex) 120-720-1500: 두께 X 폭 X 길이가 120X720X1500 인 강판

7) 합성 코드(Combined Code)
- 2 개 이상의 코드를 조합하여 만드는 방법
- ex) 연상 코드+순차 코드 → KE-711: 대한항공 711 기, AC-253: 에어캐나다 253 기

8) 코드 부여 체계
- 이름만으로 개체의 용도와 적용 범위를 알 수 있도록 코드를 부여하는 방식
- 각 개체에 유일한 코드 부여하여 개체들의 식별 및 추출을 용이하게 함
- 코드를 부여하기 전 각 단위 시스템의 고유한 코드와 개체를 나타내는 코드가 정의되야 함

- ex) PJC-COM-003: 전체 시스템 단위의 3 번째 공통 모듈
- ex) PY3-MOD-010: PY3 라는 단위 시스템의 10 번째 모듈

16 기타

시스템의 구성요소

- 입력(Input), 출력(Output), 처리(Process), 제어(Control), 피드백(Feedback), ~~유지보수(Maintenance)~~

연계시스템 구성 요소

1) 송신 시스템
   - 시스템 인터페이스를 구성하는 시스템으로, 연계할 데이터를 데이터베이스와 애플리케이션으로부터 연계 테이블 또는 파일 형태로 생성하여 **송신**하는 시스템이다.
2) 수신 시스템
   - 수신한 연계테이블, 파일데이터를 수신시스템에서 관리하는 데이터 형식에 맞게 변환하여 DB 에 저장하거나 애플리케이션에서 활용할 수 있도록 제공
3) 중계 서버
   - 송/수신 시스템 사이에서 데이터를 송수신하고, 연계데이터의 송수신 현황을 모니터링함, 연계데이터의 보안강화 및 다중플랫폼 지원 등이 가능

현행 시스템 분석에서 고려하지 않아도 되는 항목은?

- DBMS 분석
- 네트워크 분석
- 운영체제 분석
- ~~인적자원 분석~~

HIPO(Hierarchy Input Process Output)

- **하향식** 소프트웨어 개발을 위한 문서화 도구이다.
- HIPO 차트 종류에는 가시적 도표, 총체적 도표, 세부적 도표가 있다.
- 기능과 자료의 의존 관계를 동시에 표현할 수 있다.
- 보기 쉽고 이해하기 쉽다.

DBC(Design by Contract, 계약에 의한 설계)

*컴포넌트 설계시 "(**협약(Contract)**)에 의한 설계"를 따를 경우, 해당 명세에서는

(1) 컴포넌트의 오퍼레이션 사용 전에 참이 되어야 할 선행조건

(2) 사용 후 만족되어야 할 결과조건

(3) 오퍼레이션이 실행되는 동안 항상 만족되어야 할 불변조건 등이 포함되어야 한다.

실시간 소프트웨어 설계 시 고려해야 할 사항이 아닌 것은?

- 인터럽트와 문맥 교환의 표현
- 태스크들 간의 통신과 동기화
- 타이밍 제약의 표현
- ~~동기적인 프로세싱 => 비동기적 프로세싱~~

<2 과목 소프트웨어 개발>

1 자료구조

자료 구조의 분류

- 선형 구조(Linear Structure) : 배열, 스택, 큐, 데크, 선형 리스트
- 비선형 구조(Non-Linear Structure) : 트리, 그래프

1) 배열(Array)

- 정적인 자료 구조로 기억장소의 추가가 어렵고 메모리의 낭비 발생
- 반복적인 데이터 처리 작업에 적합한 구조

2) 스택(Stack)

- 입출력이 한쪽 끝으로만 제한된 리스트이다.
- 더 이상 삭제할 데이터가 없는 상태에서 데이터를 삭제하면 언더플로(Underflow)가 발생한다.
- 후입선출(LIFO, Last In First Out) 방식이다.

***스택을 이용한 연산**

- 재귀호출
- 후위표현(Post-fix expression)의 연산
- 깊이우선탐색
- ~~선택정렬~~
- 서브루틴 호출, 인터럽트 처리, 수식 계산 및 수식 표기법에 응용된다.

3) 큐(Queue)

- 한쪽에서는 삽입 작업, 다른 한쪽에서는 삭제 작업이 이뤄진다.
- Head(front)와 Tail(rear)의 2 개 포인터를 갖고 있다.
- 선입선출(FIFO, First In First Out) 방식이다.

***큐를 이용한 연산**

- 운영체제의 작업 스케줄링에 사용한다.

4) 데크(Deque)

- 삽입과 삭제가 리스트의 양쪽 끝에서 발생할 수 있는 자료 구조이다.
- 스택과 큐의 장점으로 구성한 것이다.
- Double Ended Queue 의 약자이다.
- 입력 제한 데크는 Scroll 이고, 출력 제한 테크는 Shelf 이다.

5) 선형 리스트(Linear List)

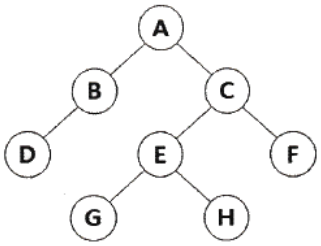
- 연속 리스트(Contiguous List) => **순차적**
  - 배열과 같이 연속되는 기억장소에 저장되는 자료 구조
  - 기억장소를 연속적으로 배정받아, 기억장소 이용 효율은 밀도가 1 로서 가장 좋음
  - 중간에 데이터를 삽입하기 위해 연속된 빈 공간이 있어야함
  - 삽입, 삭제 시 자료의 이동이 필요함

- 연결 리스트(Linked List) => **비순차적**
  - 자료들을 반드시 연속적으로 배열시키지 않고 임의의 기억공간을 기억시키되, 자료 항목의 순서에 따라 노드의 포인터 부분을 이용해 서로 연결시킨 자료 구조
  - 노드의 삽입, 삭제 작업이 용이
  - 기억공간이 연속적으로 놓여 있지 않아도 저장가능
  - 연결을 위한 포인터가 필요하기 때문에 순차 리스트에 비해 기억 공간의 효율이 좋지 않음
  - 연결을 위한 포인터를 찾는 시간이 필요하기 때문에 접근 속도가 느림
  - 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 힘듦

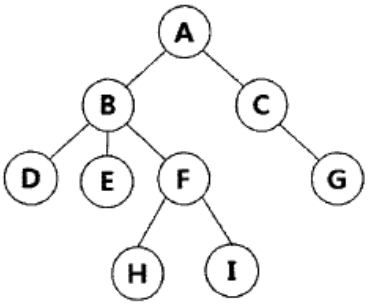
6) 트리(Tree)

- 그래프의 특수한 형태
- 노드(Node)와 선분(Branch)으로 되어 있고, 정점 사이에 사이클(Cycle)이 형성되어 있지 않으며, 자료 사이의 관계성이 계층 형식으로 나타나는 비선형 구조
- 근 노드(Root Node) : 트리의 맨 위에 있는 노드
- 디그리(Degree, 차수) : 각 노드에서 뻗어 나온 가지의 수
- 트리의 디그리 : 노드들의 디그리 중에서 가장 많은 수
- 단말 노드(Terminal Node) : 자식이 하나도 없는 노드, Degree 가 0 인 노드

*다음 트리의 차수(degree)와 단말 노드(terminal node)의 수는? **차수: 2 (A,C,E), 단말 노드: 4 (D,G,H,F)**



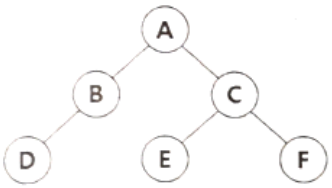
*다음 트리의 차수(degree)는? **3 (B)**



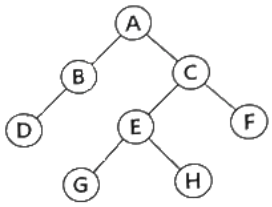
트리 순회방법

- 전위 순회(Preorder Traversal) : **Root** → Left → Right
- 중위 순회(Inorder Traversal) : Left → **Root** → Right
- 후위 순회(Postorder Traversal) : Left → Right → **Root**

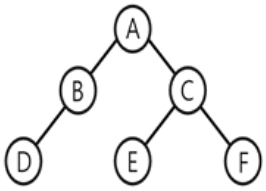
*다음 트리에 대한 중위 순회 운행 결과는? **D B A E C F**



*다음 트리를 Preorder 운행법으로 운행할 경우 다섯 번째로 탐색되는 것은? A B D C E G H F 이므로 **E**

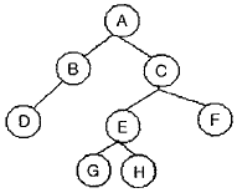


*다음 트리에 대한 INORDER 운행 결과는? **D B A E C F**

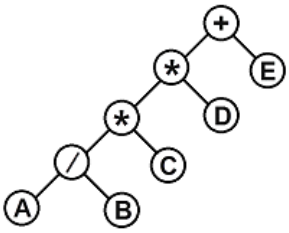


(Preorder : A B D C E F, Postorder : D B E F C A)

*다음 트리를 Preorder 운행법으로 운행할 경우 가장 먼저 탐색되는 것은? A B D C E G H F **이므로 A**



*다음 트리를 전위 순회(preorder traversal)한 결과는? **+ * * / A B C D E**



7) 그래프(Graph)

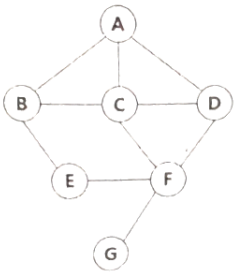
방향 그래프

- 정점을 연결하는 선에 방향이 있는 그래프
- n 개의 정점으로 구성된 방향 그래프의 최대 간선 수 = **n(n-1)**

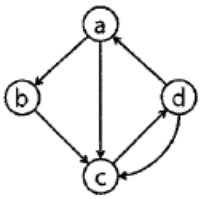
무방향 그래프

- 정점을 연결하는 선에 방향이 없는 그래프
- n 개의 정점으로 구성된 무방향 그래프의 최대 간선 수 = **n(n-1)/2**

***다음 그래프에서 정점 A 를 선택하여 깊이우선탐색(DFS)으로 운행한 결과는? ABEFGCD**



*제어흐름 그래프가 다음과 같을 때 McCabe 의 cyclomatic 수는 얼마인가? 4



선-점+2 = 6-4+2

[2] 알고리즘

이진 검색 알고리즘

- 탐색 효율이 좋고 탐색 시간이 적게 소요된다.
- 검색할 데이터가 정렬되어 있어야 한다.
- 비교횟수를 거듭할 때마다 검색 대상이 되는 데이터의 수가 절반으로 줄어든다.

피보나치 검색 알고리즘

- 피보나치 수열에 따라 다음에 비교할 대상을 선정하여 검색한다.

알고리즘 설계 기법

1) 분할과 정복(Divide and Conquer)

- 문제를 나눌 수 없을 때까지 나누고, 각각을 풀면서 다시 병합해 문제의 답을 얻음

2) 동적계획법(Dynamic Programming)

- 어떤 문제를 풀기 위해 그 문제를 더 작은 문제의 연장선으로 생각하고, 과거에 구한 해를 활용하는 방식

3) 탐욕법(Greedy)

- 결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 것을 해답으로 선택
- 현재 시점에서 가장 최적의 방법을 선택

4) 백트래킹(Backtracking)

- 어떤 노드의 유망성 점검 후, 유망하지 않으면 그 노드의 부모 노드로 되돌아간 후 다른 자손 노드를 검색
- 해를 찾는 도중 해가 아니어서 막히면, 되돌아가서 다시 해를 찾아간다.

시간 복잡도에 다른 알고리즘

복잡도	설명	대표 알고리즘
O(1)	- 상수형 복잡도 - **알고리즘 수행시간이 입력 데이터 수와 관계없이 일정**	해시 함수
O(logN)	- 로그형 복잡도 - 문제를 해결하기 위한 단계의 수가 $\log_2 N$ 번만큼의 수행 시간을 가짐	이진 탐색
O(N)	- 선형 복잡도 - 입력 자료를 차례로 하나씩 모두 처리	순차 탐색

	- 수행 시간이 자료 크기와 직접적 관계로 변함(정비례)	
O(N log2N)	- 선형 로그형 복잡도 - 문제를 해결하기 위한 단계의 수가 $N\log_2 N$ 번만큼 수행 시간을 가짐	퀵 정렬 합병 정렬
$O(N^2)$	- 제곱형 복잡도 - 주요 처리 루프 구조가 2 중인 경우 - N 의 크기가 작을 땐 N^2 이 $N\log_2 N$ 보다 느릴 수 있음	선택 정렬 버블 정렬 삽입 정렬

***최악의 경우 검색 효율이 가장 나쁜트리 구조는?**

- **이진 탐색트리 => O(n)**
- AVL 트리 => O(log n)
- 2-3 트리 => O(log 3n)
- 레드-블랙 트리 => O(log n)

[3] 정렬 알고리즘

● **삽입정렬**

- 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬

*삽입 정렬(Insertion Sort)을 이용하여 오름차순 정렬할 경우 1 회전 후의 결과는? **3, 8, 4, 9, 7**

초기 자료 : 8, 3, 4, 9, 7

● **쉘 정렬**

- 임의의 레코드 키와 매개변수(h)값만큼 떨어진 곳의 레코드 키를 비교하여 서로 교환해 가면서 정렬한다.
- 삽입정렬의 확장 개념. 입력파일이 부분적으로 정렬되어 있는 경우 유리한 방식.
- 입력파일을 매개변수값으로 서브파일 구성하고 각 서브파일을 삽입정렬로 순서 배열하는 과정을 반복함

● **선택정렬**

- n 개의 레코드 중에서 최소값을 찾아 첫 번째 레코드 위치에 놓고, 나머지 n-1 개 중에서 다시 최소값을 찾아 두 번째 레코드 위치에 놓는 방식을 반복하는 정렬

*다음 자료에 대하여 "Selection Sort"를 사용하여 오름차순으로 정렬한 경우 PASS 3 의 결과는? 3, 4, 7, 9, 8

초기상태 : 8, 3, 4, 9, 7

*다음 자료에 대하여 선택 정렬을 이용하여 오름차순으로 정렬하면 3 회전 후의 결과는? **14, 17, 35, 40, 37**

37, 14, 17, 40, 35

● **버블정렬**

- 주어진 파일에서 인접한 두 개의 레코드 키 값을 비교하여 그 크기에 따라 레코드 위치를 서로 교환한다.

*다음 자료를 버블 정렬을 이용하여 오름차순으로 정렬할 경우 Pass 2 의 결과는? **6, 3, 5, 7, 9**

*다음 자료를 버블 정렬을 이용하여 오름차순으로 정렬할 경우 PASS 3 의 결과는? **3, 5, 6, 7, 9**

9, 6, 7, 3, 5

● **퀵 정렬**

- 레코드의 많은 자료 이동을 없애고 하나의 파일을 부분적으로 나누어 가면서 정렬한다.

● **힙 정렬**
- 정렬할 입력 레코드들로 힙을 구성하고 가장 큰 키 값을 갖는 루트 노드를 제거하는 과정을 반복하여 정렬
- 평균 수행 시간은 O(nlog2n)이다.
- 완전 이진트리(complete binary tree)로 입력자료의 레코드를 구성한다.
- ~~최악의 수행 시간은 O(2n4)이다.~~

● **합병정렬**
- 정렬된 N 개의 데이터를 처리하는 데 O(Nlog2N)의 시간이 소요되는 정렬 알고리즘

4 화이트박스, 블랙박스 테스트

- 테스트 케이스에는 일반적으로 시험 조건, 테스트 데이터, 예상 결과가 포함되어야 한다.

● **화이트박스 테스트(White Box Test)**
- 화이트 박스 테스트는 모듈의 **논리적인** 구조를 체계적으로 점검할 수 있다.
- 모듈 안의 작동을 직접 **관찰** 할 수 있다.
- 산출물의 각 기능별로 적절한 프로그램의 제어구조에 따라 선택, 반복 등의 부분들을 수행함으로써 **논리적** 경로를 점검한다.
- Source Code 의 모든 문장을 한번 이상 수행함으로서 진행된다.
- ~~기본 경로(BasisPath)란 흐름 그래프의 시작노드에서 종료노드까지의 서로 독립된 경로로 싸이클을 허용하지 않는 경로다.~~
- ~~Base Path Testing, Boundary Value Analysis 가 대표적인 기법이다.~~

***화이트박스 검사로 찾기 힘든 오류는?**
- 논리흐름도, 루프구조, 순환복잡도, ~~자료구조~~

화이트박스 검사 기법 # 기조루데

1) 기초 경로 검사(Base Path Testing) : 테스트 측정 결과는 실행 경로의 기초를 정의하는 지침으로 사용
2) 조건 검사(Condition Testing) : 논리적 조건을 테스트
3) 루프 검사(Loop Testing) : 반복 구조에 맞춰 테스트
4) 데이터 흐름 검사(Data Flow Testing) : 변수의 정의와 변수가 사용되는 위치에 초점을 맞춰 테스트

● **블랙박스(Black Box Test)**
- 블랙박스 테스트는 프로그램의 구조를 고려하지 않는다.
- 모듈 안에서 어떤 일이 일어나는지 알 수 없음
- 소프트웨어가 수행할 특정 기능을 알기 위해 각 기능이 완전히 작동되는 것을 입증하는 테스트(기능 테스트)
- 소프트웨어 인터페이스에서 실시되는 테스트

블랙박스 검사 기법 # 동경원비오

1) 동등 분할 검사(Equivalence Partitioning Testing), 동치 분할 검사
- 프로그램의 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 해 테스트 케이스를 정하고, 해당 입력 자료에 맞는 결과가 출력되는지 확인하는 기법

2) 경계값 분석(Boundary Value Analysis)

- 입력 조건의 중간값보다 경계값에서 오류가 발생될 확률이 높다는 점을 이용해 입력 조건의 **경계값**을 테스트 케이스로 선정해 검사하는 기법

3) 원인-효과 그래프 검사(Cause-Effect Graphing Testing)

- 입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정해 검사하는 기법

4) 비교 검사(Comparison Testing)

- **여러 버전**의 프로그램에 동일한 테스트 자료를 제공해 **동일한 결과**가 출력되는지 테스트하는 기법

5) 오류 예측 검사(Error Guessing)

- 다른 블랙박스 테스트 기법으로 찾아낼 수 없는 오류를 찾아내는 일력의 보충적 검사 기법(데이터 확인 검사)

***블랙박스 테스트를 이용하여 발견할 수 있는 오류**

- 비정상적인 자료를 입력해도 오류 처리를 수행하지 않는 경우
- 정상적인 자료를 입력해도 요구된 기능이 제대로 수행되지 않는 경우
- 경계값을 입력할 경우 요구된 출력 결과가 나오지 않는 경우
- ~~반복 조건을 만족하는데도 루프 내의 문장이 수행되지 않는 경우~~

*평가 점수에 따른 성적부여는 다음 표와 같다. 이를 구현한 소프트웨어를 **경계값 분석 기법**으로 테스트 하고자 할 때 다음 중 테스트 케이스의 입력 값으로 옳지 않은 것은? **59, 80, ~~90~~, 101**

평가 점수	성적
80~100	A
60~79	B
0~59	C

[5] 형상 관리

형상 관리(SCM, Software Configuration Management)

- **개발 과정의 변경 사항을 관리하는 것**
- 소프트웨어에 가해지는 변경을 제어하고 관리한다.
- 소프트웨어에서 일어나는 수정이나 변경을 알아내고 제어하는 것을 의미한다.
- 형상 관리를 통해 가시성과 추적성을 보장함으로써 소프트웨어의 생산성과 품질을 높일 수 있다.
- 소프트웨어 개발의 전체 비용을 줄이고, 개발 과정의 여러 방해 요인이 최소화되도록 보증하는 것이다.
- 유지 보수 단계뿐만 아니라 개발 단계에도 적용할 수 있다.
- 형상관리의 기능 중 하나는 버전 제어 기술이다.
- 프로젝트 계획, 분석서, 설계서, 프로그램, 테스트 케이스 모두 관리 대상이다.
- 형상 관리를 통해 이전 리버전이나 버전에 대한 정보에 접근 가능하여 배포본 관리에 유용
- 불필요한 사용자의 소스 수정 제한
- 동일한 프로젝트에 대해 여러 개발자 동시 개발 가능
- ~~프로젝트 개발비용을 효율적으로 관리~~
- ~~형상 통제 과정에서는 형상 목록의 변경 요구를 즉시 수용 및 반영해야 한다.~~
- ~~대표적인 형상관리 도구로 Ant, Maven, Gradle 등이 있다. => 빌드 자동화 도구~~
- ~~형상관리를 위하여 구성된 팀을 "chief programmer team"이라고 한다.~~

형상 관리 도구의 주요 기능

- **저장소(Repository)** : 최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳
- **가져오기(Import)** : 버전 관리가 되고 있지 않은 아무것도 없는 저장소에 처음으로 파일을 복사하는 것
- **동기화(Update)** : 저장소에 있는 최신 버전으로 자신의 작업 공간(로컬 저장소)을 동기화하는 것
- **체크아웃(Check-Out)** : 프로그램을 수정하기 위해 저장소에서 파일을 받아오는 것
- **체크인(Check-In)** : 저장소에 새로운 버전의 파일로 갱신하는 것
- **커밋(Commit)** - 체크인을 수행할 때 충돌(Confilct) 날 경우 diff 도구를 이용해 수정 후 갱신 완료
- ~~**정규화(Normalization)**~~

형상 관리 절차 : 형상 식별 -> 형상 통제 -> 형상 감사 -> 형상 기록/보고

① 형상 식별
- 형상 관리 계획을 근거로 형상관리의 대상이 무엇인지 식별하는 과정이다.

② 형상 통제
- 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선(Baseline)이 잘 반영될 수 있도록 조정
- 형상 통제가 이루어지기 위해서는 형상 통제 위원회(CCB)의 승인을 통한 변경 통제가 이루어짐

③ 형상 감사
- 형상 관리 계획대로 형상관리가 진행되고 있는지, 형상 항목의 변경이 요구 사항에 맞도록 제대로 이뤄졌는지 등을 살펴보는 활동이다.

④ 형상 기록/보고
- 형상의 식별, 통제, 감사 작업의 결과를 기록, 관리하고 보고서를 작성하는 작업

***소프트웨어 형상 관리에서 '관리' 항목에 포함되지 않는 것은?**
- 프로젝트 요구 분석서
- 소스 코드
- 운영 및 설치 지침서
- ~~프로젝트 개발 비용~~

6 버전 관리

1) 공유 폴더 방식
- 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식
- 개발자들은 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사함
- 담당자는 공유 폴더의 파일을 자기 PC 로 복사해 컴파일 한 후 이상 유무 확인
- 파일의 변경 사항을 데이터베이스에 기록하며 관리
- ex. SCCS, RCS, PVCS, QVCS

2) 클라이언트/서버 방식
- 버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리되는 방식
- 서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사해 작업한 후 변경된 내용을 중앙 서버에 반영
- 모든 버전 관리는 서버에서 수행됨
- 하나의 파일을 서로 다른 개발자가 작업할 경우 경고 메시지 출력
- 서버에 문제가 생기면 다른 개발자와의 협업 및 버전 관리 작업은 중단됨
- ex. CVS, SVN(Subversion)

3) 분산 저장소 방식

- 버전관리 자료가 원격저장소와 로컬저장소에 함께 저장되어 관리된다.
- 로컬저장소에서 버전관리가 가능하므로 원격저장소에 문제가 생겨도 로컬저장소의 자료를 이용하여 작업할 수 있다.
- 대표적인 버전 관리 도구로 Git 이 있다.
- 하나의 원격 저장소와 분산된 개발자 PC 의 로컬 저장소에 함께 저장되어 관리되는 방식
- 개발자별로 원격 저장소의 자료를 자신의 로컬 저장소로 복사해 작업한 후 변경 된 내용을 로컬 저장소에서 우선 반영(Commit)한 다음 이를 원격 저장소에 반영(Push)
- 원격 저장소에 문제가 생겨도 로컬 저장소의 자료를 이용해 작업 가능
- 로컬 저장소에서 작업을 수행할 수 있어 처리속도가 빠름
- ex. Git, Bitkeeper

7 테스트

애플리케이션 테스트의 기본 원칙

1) 테스팅은 결함이 존재함을 밝히는 것

- 결함을 줄일 순 있지만, 결함이 없다고는 증명할 수 없음

2) 완벽한 테스팅은 불가능

- 무한 경로, 무한 입력 값으로 인한 어려움

3) 개발 초기에 테스팅 시작

- 테스팅 기간 단축, 재작업 감소로 개발 기간 단축 및 결함 예방

4) 결함 집중

- **파레토** 법칙이 좌우한다. (Pareto)
- 애플리케이션 결함의 대부분은 소수의 특정한 모듈에 집중되어 존재한다.
- 결함은 발생한 모듈에서 계속 추가로 발생할 가능성이 높다.

***Pareto 의 법칙** : 소프트 웨어 테스트에서 오류의 80%는 전체 모듈의 20% 내에서 발견된다.

***Brooks 의 법칙** : 지연되는 프로젝트에 인력을 더 투입하면 오히려 더 늦어진다.

5) 살충제 패러독스(Pesticide Paradox)

- 동일한 테스트 케이스에 의한 반복적 테스트는 새로운 버그를 찾지 못함

6) 테스팅은 정황에 의존적

- 소프트웨어 성격에 맞게 테스트 실시

7) 오류-부재의 궤변

- 요구사항을 충족시켜주지 못한다면, 결함이 없어도 품질이 높다 볼 수 없음

애플리케이션 테스트의 분류

1) 프로그램 실행 여부에 따른 테스트

- 정적 테스트

- 프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트
- 워크 스루, 인스펙션, 코드 검사

- 동적 테스트

- 프로그램을 실행하여 오류를 찾는 테스트
- 화이트박스 테스트, 블랙박스 테스트

2) 테스트 기반에 따른 테스트

- **명세 기반 테스트**
  - 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트
  - 동등 분할, 경계값 분석(블랙박스 테스트)
- **구조 기반 테스트**
  - 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트
  - 구문 기반, 결정 기반, 조건 기반(화이트박스 테스트)
- **경험 기반 테스트**
  - 테스터의 경험을 기반으로 수행하는 테스트
  - 에러 추정, 체크 리스트, 탐색적 테스팅

3) 시각에 따른 테스트

- **검증(Verification) 테스트**
  - 개발자의 시각에서 제품의 생산 과정을 테스트하는 것
  - 검증은 소프트웨어 개발 과정을 테스트하는 것이다. 작업 제품이 개발자의 기대를 충족시키는지 측정한다.
  - 단위 테스트, 통합 테스트, 시스템 테스트
- **확인(Validation) 테스트**
  - 사용자의 시각에서 생산된 제품의 결과를 테스트하는 것
  - 확인은 소프트웨어 결과를 테스트 하는 것이다. 사용자의 요구에 적합한지 측정한다.
  - 인수 테스트(알파 테스트, 베타 테스트)

4) 목적에 따른 테스트

- **회복(Recovery) 테스트** : 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복귀하는가?
- **안전(Security) 테스트** : 부당하고 불법적인 침입을 시도하여 보안시스템이 불법적인 침투를 잘 막아내는가?
- **강도(Stress) 테스트** : 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는가?
- **성능(Performance) 테스트** : 사용자의 이벤트에 시스템이 응답하는 시간, 특정 시간 내에 처리하는 업무량, 사용자 요구에 시스템이 반응하는 속도 등을 테스트
- **구조(Structure) 테스트** : 소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가
- **회귀(Regression) 테스트** : 소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인
- **병행(Parallel) 테스트** : 변경된 소프트웨어와 기존 소프트웨어에 동일 데이터를 입력하여 결과를 비교

테스트 커버리지 유형

- 구문 커버리지 : 프로그램 내 모든 문장을 적어도 한 번 이상 실행하는 것을 기준으로 수행
- 결정 커버리지 : 결정 조건 내 전체 조건식이 최소한 참/거짓 한 번의 값을 가지도록 측정
- 조건 커버리지 : 전체 조건식 결과와 관계없이 개별 조건식이 참/거짓 한번 모두 갖도록 개별 조건식을 조합
- 조건/결정 커버리지 : 전체 조건식이 참/거짓 한번씩 가지면서 개별 조건식이 참/거짓 모두 한번씩 갖도록 조합
- 변경/조건 결정 커버리지 : 각 개별 조건식이 다른 개별 조건식의 영향을 받지 않고 전체 조건식의 결과에 독립적으로 영향을 주도록 함으로써 조건/결정 커버리지를 향상
- 다중 조건 커버리지 : 결정 조건 내 모든 개발 조건식의 모든 가능한 조합을 100% 보장

성능 테스트 도구

- 애플리케이션의 처리량, 응답시간, 경과시간, 자원사용률에 대해 **가상의 사용자**를 생성하고 테스트를 수행함으로써 성능 목표를 달성하였는지를 확인하는 테스트 자동화 도구

8 단위 테스트(Unit Test)

- **개별 모듈을 시험**하는 것으로 모듈이 정확하게 구현되었는지, 예정한 기능이 제대로 수행되는지를 점검하는 것이 주요 목적인 테스트
- 명세 기반 테스트, 구조 기반 테스트 중에서 주로 **구조 기반 테스트를** 시행함

***단위 테스트를 통해 발견할 수 있는 오류가 아닌 것은?**

- 알고리즘 오류에 따른 원치 않는 결과
- 탈출구가 없는 반복문의 사용
- 틀린 계산 수식에 의한 잘못된 결과
- ~~**모듈 간**의 비정상적 상호작용으로 인한 원치 않는 결과 => 통합 테스트~~

9 통합 테스트(Integration Test)

- 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트
- 모듈 간 또는 통합된 컴포넌트 간의 상호 작용 오류 검사

1) 상향식 통합 테스트(Bottom Up Integration Test)

- 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법
- 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 **클러스터(Cluster)** 필요
- 상위 모듈 개발이 완료되지 않은 경우 **드라이버(Driver)**를 사용하기도 한다.
- 하위 모듈들을 클러스터(Cluster)로 결합
  → 더미 모듈인 드라이버(Driver) 작성
  → 통합된 클러스터 단위로 테스트
  → 테스트 완료 후 클러스터는 프로그램 구조의 상위로 이동해 결합하고 드라이버는 실제 모듈로 대체됨

2) 하향식 통합 테스트(Top Down Integration Test)

- 깊이 우선 방식 또는 너비 우선 방식이 있다.
- 상위 컴포넌트를 테스트 하고 점증적으로 하위 컴포넌트를 테스트한다.
- 하위 컴포넌트 개발이 완료되지 않은 경우 스텁(Stub)을 사용하기도 한다.
- 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있음
- 상위 모듈에서는 테스트 케이스 사용하기 어려움
- 주요 제어 모듈은 작성된 프로그램을 사용
  → 주요 제어 모듈의 종속 모듈은 **스텁(Stub)**으로 대체
  → 깊이 또는 너비 우선 방식에 따라 하위 모듈인 스텁(Stub)들이 한 번에 하나씩 실제 모듈로 교체됨
  → 모듈이 통합될 때마다 테스트 실시
  → 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트 실시

**스텁(Stub)*

- 하향식 통합 테스트를 위해 일시적으로 필요한 조건만을 가지고 제공되는 **시험용 임시** 모듈

3) 혼합식 통합 테스트

- 하위 수준에서는 상향식 통합을, 상위 수준에서는 하향식 통합을 사용해 최적의 테스트를 지원하는 방식
- 샌드위치식 통합 테스트 방법

4) 빅뱅 테스트

- 통합 테스트(Integration Test) 중 비점진적 통합 방식

5) 회귀 테스트(Regression Testing)

- 이미 테스트된 프로그램의 테스팅 반복
- 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인

10 인수 테스트(Acceptance Test)

- 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두는 테스트

1) 알파 테스트

- 개발자의 장소에서 사용자가 개발자 앞에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 검사하는 기법
- 사용자가 개발자 앞에서 검사한다.
- 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록한다.

2) 베타 테스트

- 필드 테스팅(field testing)이라고도 불리며 개발자 없이 고객의 사용 환경에 소프트웨어를 설치하여 검사를 수행하는 인수검사 기법
- 선정된 최종 사용자가 여러 명의 사용자 앞에서 검사한다.
- 통제된 환경에서 베타검사는 개발자에 의해 제어되지 않는 상태에서 검사한다.

3) 사용자 인수 테스트

4) 운영상의 인수 테스트

5) 계약 인수 테스트

6) 규정 인수 테스트

11 테스트 케이스, 테스트 시나리오, 테스트 오라클, 테스트 하네스

1) 테스트 케이스(Test Case)

- 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목 명세서
- 명세 기반 테스트(블랙박스 테스트)의 설계 산출물에 해당

테스트 케이스의 구성요소

- 식별자(항목 식별자, 일련번호)
- 테스트항목(테스트 대상-모듈 또는 기능)
- 입력 명세(테스트 데이터, 테스트 조건)
- 출력 명세(예상 결과)
- 환경 설정(필요한 하드웨어나 소프트웨어의 환경)
- 특수 절차 요구(테스트 케이스 수행 시 특별히 요구되는 절차)
- 의존성 기술(테스트 케이스 간의 의존성)

테스트 케이스에 일반적으로 포함되는 항목

- 테스트 조건
- 테스트 데이터
- 예상 결과
- ~~테스트 비용~~

2) 테스트 시나리오(Test Scenario)

- 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합
- 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서

3) 테스트 오라클(Test Oracle)

- 테스트의 결과가 참인지 거짓인지를 판단하기 위해서 사전에 정의된 참값을 입력하여 비교하는 기법 및 활동을 말한다.
- 종류에는 **참, 샘플링, 휴리스틱, 일관성 검사**가 존재한다.
- 테스트 오라클의 특징
  - 제한된 검증 : 모든 테스트 케이스에 적용할 수 없음
  - 수학적 기법 : 값을 수학적 기법을 이용해 구할 수 있음
  - 자동화 기능 : 프로그램 실행, 결과 비교, 커버리지 측정 등을 자동화할 수 있음

테스트 오라클의 종류

- **참(True) 오라클**
  - 모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하는 오라클
  - 발생된 모든 오류를 검출할 수 있음
- **샘플링(Sampling) 오라클**
  - 특정한 몇몇 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클
- **휴리스틱(Heuristic, 추정) 오라클**
  - 샘플링 오라클 개선.
  - 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고 나머지 입력 값들에 대해서는 추정으로 처리하는 오라클
- **일관성(Consistent) 검사 오라클**
  - 변경이 있을 때 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클

4) 테스트 하네스(Test Harness)

- **테스트 드라이버(Test Driver)**
  - 시험대상 모듈을 호출하는 간이 소프트웨어이다.
  - 필요에 따라 매개 변수를 전달하고 모듈을 수행한 후의 결과를 보여줄 수 있다.
  - **상향식** 통합 테스트에서 사용된다.

- **테스트 스텁(Test Stub)**
  - 테스트 대상 모듈이 호출하는 하위 모듈의 역할을 한다.
  - 하향식 통합 테스트에서 사용된다.
  - 테스트 대상의 상위 모듈을 대신하는, 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구

- **테스트 슈트(Test Suites)**
  - 테스트 대상 컴포넌트나 모듈 등 시스템에 사용되는 테스트 케이스의 집합

테스트 스크립트(Test Script)
- 자동화된 테스트 실행 절차에 대한 명세서

목 오브젝트(Mock Object)
- 사전에 사용자의 행위를 조건부로 입력해 두면, 그 상황에 맞는 예정된 행위를 수행하는 객체

***테스트 케이스 자동 생성 도구를 이용하여 테스트 데이터를 찾아내는 방법이 아닌 것은?**
- 입력 도메인 분석
- 랜덤 테스트
- 자료 흐름도
- ~~스터브(Stub)와 드라이버(Driver) => 통합 테스트때 사용함~~

12 DRM (디지털 저작권 관리, Digital Right Management)

디지털 저작권 관리의 구성 요소
- **콘텐츠 제공자(Contents Provider)** : 콘텐츠를 제공하는 저작권자
- **콘텐츠 분배자(Contents Distributor)** : 암호화된 콘텐츠를 유통하는 곳이나 사람
- **콘텐츠 소비자(Customer)** : 콘텐츠를 구매해서 사용하는 주체
- **패키저(Packager)** : 콘텐츠를 메타 데이터와 함께 배포 가능한 단위로 묶는다.
- **보안 컨테이너(Security Container)** : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치
- **클리어링 하우스(Clearing House)** : 키 관리 및 라이선스 발급 관리★
- **DRM 컨트롤러(DRM Controller)** : 배포된 콘텐츠의 이용 권한을 통제
- ~~Dataware house~~

디지털 저작권 관리의 기술 요소★
- **암호화(Encryption)** : 콘텐츠 및 라이선스를 암호화하고 전자서명을 할 수 있는 기술
- **키 관리(Key Management)** : 콘텐츠를 암호화한 키에 대한 저장 및 분배 기술
- 식별 기술(Identification) : 콘텐츠에 대한 식별 체계 표현 기술
- 저작권 표현(Right Expression) : 라이선스의 내용 표현 기술
- 암호화 파일 생성(Packager) : 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술
- **정책 관리(Policy Management)** : 라이선스 발급 및 사용에 대한 정책 표현 및 관리 기술
- **크랙 방지(Tamper Resistance)** : 크랙에 의한 콘텐츠 사용 방지 기술
- 인증(Authentication) : 라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술
- ~~**방화벽**~~

***디지털 저작권 관리(DRM) 기술과 거리가 먼 것은?**
1. 콘텐츠 암호화 및 키 관리
2. 콘텐츠 식별체계 표현
3. ~~**콘텐츠 오류 감지 및 복구**~~
4. 라이선스 발급 및 관리

13 애플리케이션 성능

애플리케이션 성능

- **처리량(Throughput)** : 일정 시간 내 애플리케이션이 처리하는 일의 양
- **응답 시간(Response Time)** : 애플레이케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
- **경과 시간(Turn Around Time)** : 애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
- **자원 사용률(Resource Usage)** : 애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률

클린 코드 작성 원칙

- 누구든지 쉽게 이해하는 코드 작성
- 다른 모듈에 미치는 영향 최소화
- 단순, 명료한 코드 작성
- 중복이 최소화된 코드 작성
- 의존성 배제, 중복성 최소화, 추상화

단순성

- 한 번에 한 가지 처리만 수행한다.
- 클래스/메서드/함수를 최소 단위로 분리한다.

코드의 간결성을 유지하기 위해 사용되는 지침

- 공백을 이용하여 실행문 그룹과 주석을 명확히 구분한다.
- 복잡한 논리식과 산술식은 괄호와 들여쓰기(Indentation)를 통해 명확히 표현한다.
- 빈 줄을 사용하여 선언부와 구현부를 구별한다.
- 코드의 중복을 최소화 한다.
- 누구든지 코드를 쉽게 읽을 수 있도록 작성한다.
- 간단하게 코드를 작성한다.
- ~~한 줄에 최대한 많은 문장을 코딩한다.~~
- ~~코드가 다른 모듈에 미치는 영향을 최대화하도록 작성한다.~~

공학적으로 잘된 소프트웨어(Well Engineered Software)

- 소프트웨어는 유지보수가 용이해야 한다.
- 소프트웨어는 신뢰성이 높아야 한다.
- 소프트웨어는 충분한 테스팅을 거쳐야 한다.
- ~~소프트웨어는 사용자 수준에 무관하게 일관된 인터페이스를 제공해야 한다.~~

테스트와 디버그의 목적

- 테스트는 오류를 찾는 작업이고 디버깅은 오류를 수정하는 작업이다.

소스 코드 품질분석 도구의 종류

- 정적 분석도구 : **pmd**, **cppcheck**, **checkstyle**, SonarQube, ccm, cobertuna
- 동적 분석도구 : Avalanche, Valgrind, **valance**

소스코드 정적 분석(Static Analysis)에 관한 것이 아닌 것은?

- 소스 코드를 실행시키지 않고 분석한다.
- 코드에 있는 오류나 잠재적인 오류를 찾아내기 위한 활동이다.
- 자료 흐름이나 논리 흐름을 분석하여 비정상적인 패턴을 찾을 수 있다.
- ~~하드웨어적인 방법으로만 코드 분석이 가능하다.~~

14 EAI

EAI(Enterprise Application Integration)

- 기업 내 각종 애플리케이션 및 플랫폼 간의 정보 전달, 연계, 통합 등 상호 연동이 가능하게 해주는 솔루션

EAI 구축유형

1) 포인트 투 포인트(Point to Point)

- 점 대 점으로 연결하는 방식, 변경 및 재사용이 어려움
- 가장 기본적인 애플리케이션 통합 방식으로, 애플리케이션을 1:1 로 연결

2) 허브 앤 스포크(Hub & Spoke)

- 단일 접점인 허브(Hub) 시스템을 통해 데이터를 전송하는 중앙 집중형 방식
- 확장 및 유지보수가 용이하지만 허브 장애 발생 시 시스템 전체에 영향을 미침

3) 메시지 버스(Message Bus, ESB 방식)

- 애플리케이션 사이에 미들웨어를 두어 처리하는 방식★
- 확장성이 뛰어나며 대용량 처리가 가능

4) 하이브리드(Hybrid)

- Hub & Spoke 와 Message Bus 의 혼합방식이다.
- 필요한 경우 한 가지 방식으로 EAI 구현이 가능하다.
- 데이터 병목현상을 최소화할 수 있다.

***ESB(Enterprise Service Bus)**

- 애플리케이션 간 연계, 데이터 변환, 웹 서비스 지원 등 표준 기반의 인터페이스를 제공하는 솔루션
- 애플리케이션 통합 측면에서 EAI 와 유사하지만 애플리케이션 보다는 서비스 중심의 통합을 지향
- 결합도(Coupling)를 약하게(Loosely) 유지함
- 관리 및 보안 유지가 쉽고, 높은 수준의 품질 지원이 가능

15 인터페이스 구현 검증 도구

- **xUnit** : Java(Junit), C++(Cppunit), .Net(Nunit) 등 다양한 언어를 지원하는 단위 테스트 프레임워크
- **STAF**
  - 서비스 호출, 컴포넌트 **재사용** 등 다양한 환경을 지원하는 테스트 프레임워크
  - 각 테스트 대상 분산 환경에 **데몬**을 사용하여 테스트 대상 프로그램을 통해 테스트를 수행하고, 통합하여 자동화하는 검증 도구
- **NTAF** : STAF 의 장점인 재사용 및 확장성과 FitNesse 의 장점인 협업 기능을 통합한 NHN(네이버)의 테스트
- **FitNesse** : 웹 기반 테스트케이스 설계, 실행, 결과 확인 등을 지원하는 테스트 프레임워크 자동화 프레임워크
- **Selenium** : 다양한 브라우저 및 개발 언어를 지원하는 웹 애플리케이션 테스트 프레임워크
- **watir** : Ruby 언어를 사용하는 애플리케이션 테스트 프레임워크

16 개발 지원 도구

통합 개발 환경(IDE; Integrated Development Environment)

- 개발에 필요한 환경, 즉 편집기(Editor), 컴파일러(Compiler), 디버거(Debugger) 등의 다양한 툴을 하나의 인터페이스로 통합해 제공하는 것을 의미함
- ex. Eclipse, Visual Studio, X Code, Android Studio, IDEA

빌드 자동화 도구

- Gradle 은 실행할 처리 명령들을 모아 태스크로 만든 후 태스크 단위로 실행한다.
- 빌드 자동화 도구는 지속적인 통합개발환경에서 유용하게 활용된다.
- 빌드 자동화 도구에는 Ant, Gradle, Jenkins 등이 있다.
- ~~Jenkins 는 Groovy 기반으로 한 오픈소스로 안드로이드 앱 개발 환경에서 사용된다.~~

17 해싱함수의 종류

1) 제산법(division)

- 레코드키로 해시표의 크기보다 큰 수 중에서 가장 작은소수로 나눈 나머지를 홈 주소로 삼는 방식

2) 제곱법(mid-square)

3) 폴딩법(중첩법)

- 레코드 키를 여러 부분으로 나누고, 나눈 부분의 각 숫자를 더하거나 XOR 한 값을 홈 주소로 사용하는 방식

4) 숫자분석법(계수분석법) (digit analysis)

- 키 값을 이루는 숫자의 분포를 분석하여 비교적 고른 자리를 필요한 만큼 택해서 홈 주소로 삼는 방식

5) 기수변환법

- 키 숫자의 진수를 다른 진수로 변환시켜 주소 크기를 초과한 높은 자릿수를 절단하고, 이를 다시 주소 범위에 맞게 조정하는 방법

6) 무작위 방법

18 소프트웨어 품질

소프트웨어 품질 목표

- **이식성(Portability)**
  - 하나 이상의 하드웨어 환경에서 운용되기 위해 쉽게 수정될 수 있는 시스템 능력
- **정확성(Correctness)**
  - 시스템의 사양과 설계, 구현에 있어서 오류가 없는 정도
- **유용성(Usability)**
  - 쉽게 배우고 사용할 수 있는 정도를 나타내는 것
- **효율성(Efficiency)**
  - 요구되는 기능을 수행하기 위해 필요한 자원의 소요 정도를 의미
- **신뢰성(Reliability)**
  - 주어진 시간동안 주어진 기능을 오류없이 수행하는 정도를 나타내는 것
  - 정확하고 일관된 결과로 요구된 기능을 수행하는 시스템 능력
- **무결성(Integrity)**
  - 시스템이 프로그램이나 데이터에 대한 허용되지 않거나 잘못된 접근을 막는 정도
- **적응성(Adaptablility)**

- 시스템을 변경하지 않고 설계된 환경에서 뿐만 아니라 다른 응용 분야나 환경에서도 사용될 수 있는 정도

- **정밀성(Accuracy)**
  - 구성된 시스템에 오류가 없는 정도. 정밀성은 정확성과 다르다.
  - 정밀성은 시스템이 정확하게 구성되었는지가 아닌 시스템이 용도대로 얼마나 잘 수행하는지를 결정한다.
- **견고성(Robustness)**
  - 시스템이 잘못된 입력이나 악조건에서도 기능을 계속해서 수행할 수 있는 정도
- **유연성**
- ~~**종속성**~~

***소프트웨어 품질 측정을 위해 개발자 관점에서 고려해야 할 항목이 아닌 것은?**

- 정확성
- 무결성
- 사용성
- ~~간결성~~

19 소프트웨어 패키징

- 패키징은 **사용자** 중심으로 진행한다.
- 신규 및 변경 개발소스를 식별하고, 이를 모듈화하여 상용제품으로 패키징 한다.
- 고객의 편의성을 위해 매뉴얼 및 버전관리를 지속적으로 한다.
- 범용 환경에서 사용이 가능하도록 일반적인 배포 형태로 패키징이 진행된다.

패키징 작업 순서

- 기능 식별→ 모듈화→ 빌드 진행→ 사용자 환경 분석→ 패키징 및 적용 시험→ 패키징 변경 개선→ 배포

***제품 소프트웨어 패키징 도구 활용 시 고려사항**

- 반드시 내부 콘텐츠에 대한 암호화 및 보안을 고려한다.
- 사용자 편의성을 위한 복잡성 및 비효율성 문제를 고려한다.
- 제품 소프트웨어 종류에 적합한 암호화 알고리즘을 적용한다.
- 추가로 다양한 이기종 연동을 고려한다.
- ~~보안상 단일 기종에서만 사용할 수 있도록 해야 한다.~~
- ~~내부 콘텐츠에 대한 보안은 고려하지 않는다.~~
- ~~보안을 위하여 이기종 연동을 고려하지 않아도 된다.~~

20 연산식 표기법

- 전위 표기법(prefix) - 연산자가 앞에
- 중위 표기법(infix) - 연산자가 안에
- 후위 표기법(postfix) - 연산자가 뒤에

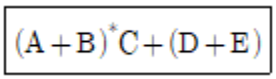
*다음 postfix 로 표현된 연산식의 연산 결과로 옳은 것은? (3*4)+(5*6) = **42**

3 4 * 5 6 * +

*다음 전위식(prefix)을 후위식(postfix)으로 옳게 표현한 것은? **A B C + * D / E -**

- / * A + B C D E

*다음 수식을 후위 표기법(postfix)으로 옳게 표시한 것은? **A B + C * D E + +**



21 JSON, AJAX, XML

JSON(JavaScript Object Notation)

- 웹과 컴퓨터 프로그램에서 용량이 적은 데이터를 교환하기 위해 데이터 객체를 속성·값의 쌍 형태로 표현하는 형식으로 자바스크립트(JavaScript)를 토대로 개발되어진 형식

XML(eXtensible Markup Language)

- 특수한 목적을 갖는 마크업 언어를 만드는 데 사용되는 다목적 마크업 언어

AJAX(Asynchronous JavaScript and XML)

- JavaScript 를 사용한 비동기 통신기술로 클라이언트와 서버 간에 XML 데이터를 주고 받는 기술

22 기타

모듈

- 소프트웨어 구조를 이루며, 다른 것들과 구별될 수 있는 독립적인 기능을 갖는 단위이다.
- 하나 또는 몇 개의 논리적인 기능을 수행하기 위한 명령어들의 집합이라고도 할 수 있다.
- 서로 모여 하나의 완전한 프로그램으로 만들어질 수 있다.

Fault

- 소프트웨어 개발 활동을 수행함에 있어서 시스템이 고장(Failure)을 일으키게 하며, 오류(Error)가 있는 경우 발생하는 것

외계인코드(Alien Code)

- 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 어려운 프로그램

제품 소프트웨어의 사용자 매뉴얼 작성절차

- 작성 지침 정의 -> 사용자 매뉴얼 구성 요소 정의 -> 구성 요소별 내용 작성 -> 사용자 매뉴얼 검토

소프트웨어 설치 매뉴얼에 포함될 항목

- 제품 소프트웨어 개요
- 설치 관련 파일
- 프로그램 삭제
- ~~소프트웨어 개발 기간~~

***S/W 유지보수 작업의 목적이 아닌 것은?**

- ~~설계수정~~
- 예방조치

- 환경적응
- 하자보수

소프트웨어 재공학이 소프트웨어 재개발에 비해 갖는 장점
- 위험부담 감소
- 비용 절감
- 시스템 명세의 오류억제
- 개발시간의 감소

S/W 재공학 관점에서 가장 연관 깊은 유지보수 유형
- Preventive maintenance (예방 유지보수)

S/W 재공학 활동 중 기존 S/W 를 다른 운영체제나 하드웨어 환경에서 사용할 수 있도록 변환하는 작업
- 이식(Migration)

<3 과목 데이터베이스 구축>

1 데이터베이스 설계

데이터베이스 설계 시 고려사항 : 무결성, 일관성, 회복, 보안, 효율성, 데이터베이스 확장

DBMS 분석시 고려사항 : 무결성(가용성), 효율성(성능), 일관성(상호 호환성), ~~네트워크 구성도~~

데이터베이스 설계 순서

1) 개념적 설계 (정보 모델링, 개념화)
- 개념 스키마 모델링
- 트랜잭션 모델링
- 독립적인 개념 스키마 설계
- E-R 다이어그램 모델
- 사용자의 요구에 대한 **트랜잭션을 모델링**한다.

2) 논리적 설계 (데이터 모델링)
- **목표 DBMS 에 맞는 스키마 설계**
- **트랜잭션 인터페이스 설계**
- 관계형 데이터베이스에서는 **테이블을 설계**하는 단계이다. (RDB)
- 데이터 타입 및 데이터 타입들 간의 관계로 표현한다.
- 스키마의 평가 및 정제
- 논리적 데이터베이스 구조로 매핑(Mapping)
- DBMS 에 맞는 논리적 스키마를 설계한다.

3) 물리적 설계 (데이터 구조화)
- 물리적 설계의 목적은 효율적인 방법으로 데이터를 저장하는 것이다.
- 트랜잭션 처리량과 응답시간, 디스크 용량 등을 고려해야 한다.
- 저장 레코드의 형식, 순서, 접근 경로와 같은 정보를 사용하여 설계한다.

- 레코드 집중의 분석 및 설계
- 접근 경로 설계
- 저장 레코드의 양식 설계

데이터베이스(Database) 특징

- 공용 데이터(Shared Data) : 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료
- 통합된 데이터(Integrated Data) : 자료의 중복을 최대로 배제한 데이터의 모임
- 운영 데이터(Operational Data) : 고유한 업무를 수행하는 데 없어서는 안 될 자료
- 저장된 데이터(Stored Data) : 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료

외부 스키마, 내부 스키마, 개념 스키마

1) 외부 스키마(External Schema) = 서브 스키마

- **사용자의 관점**에서 보여주는 데이터베이스 구조로 전체 데이터베이스의 일부이므로 **서브 스키마**라고도 함

2) 내부 스키마(Internal Schema)

- 물리적 저장 장치의 입장에서 본 데이터베이스 구조로서 실제로 데이터베이스에 저장될 레코드의 형식을 정의하고 저장 데이터 항목의 표현 방법, 내부 레코드의 물리적 순서 등을 나타낸다.

3) 개념 스키마(Conceptual Schema)

- 데이터베이스 **전체**를 정의한 것으로 데이터 개체, 관계, 제약조건, 접근권한, 무결성 규칙 등을 명세한 것

2 SQL

● 데이터 정의어 (DDL, Data Define Language)

- domin(도메인), schema(스키마), table(테이블), view(뷰), index(인덱스)를 정의, 변경, 삭제시 사용
- 데이터베이스에 저장될 데이터의 타입과 구조에 대한 정의, 이용 방식, 제약 조건 등을 명시

CREATE : DOMAIN, SCHEMA, TABLE, VIEW, INDEX 정의
ALTER : TABLE 에 대한 정의 변경 (ex. 테이블의 필드가 누락되어 추가할 경우)
DROP - DOMAIN, SCHEMA, TABLE, VIEW, INDEX 삭제

***CASCADE** : **참조 무결성을 유지하기 위하여 DROP 문에서 부모 테이블의 항목 값을 삭제할 경우 자동적으로 자식 테이블의 해당 레코드를 삭제하기 위한 옵션**

***RESTRICTED** : 다른 개체가 제거할 요소를 참조 중이면 제거 취소
- 개체 삭제 시 부모 테이블일 경우(나를 참조하고 있는 테이블이 있을 경우) 변경/삭제가 취소

● 데이터 조작어 (DML, Data Manipulation Language)

- SELECT(검색), INSERT(삽입), UPDATE(갱신), DELETE(삭제)로 저장된 데이터를 실질적으로 처리하는 데 사용

SELECT : 테이블에서 조건에 맞는 행 검색
- **SELECT** 컬럼 **FROM** 테이블명 [WHERE 조건];
- **HAVING** 절은 **GROUP BY** 와 함께 사용된다.
- 집계함수의 종류 : COUNT, SUM, AVG, MAX, MIN, STDDEV, VARIANCE, ROLLUP, CUBE

INSERT : 테이블에 새로운 행 삽입

- **INSERT INTO** 테이블명 **VALUES** 데이터;

UPDATE : 테이블에서 조건에 맞는 행의 내용 갱신(변경)

- **UPDATE** 테이블명 **SET** 컬럼명=데이터 [WHERE 조건];

DELETE : - 테이블에서 조건에 맞는 행 삭제

- **DELETE FROM** 테이블명 [WHERE 조건];

***SQL 의 논리 연산자가 아닌 것은?**

- AND, OR, NOT, ~~OTHER~~

● **데이터 제어어 (DCL, Data Control Language)**

- **데이터 보안, 무결성 유지, 병행수행 제어,** ~~논리적, 물리적 데이터 구조 정의~~

COMMIT : 트랜잭션 처리가 정상적으로 종료되어 수행한 변경 내용을 DB 에 반영하는 명령어

ROLLBACK : 트랜잭션의 실행이 실패했으므로 트랜잭션이 수행한 결과를 원래의 상태로 원상 복귀시킨다.

***SAVEPOINT (CHECKPOINT)** : 트랜잭션 내에 ROLLBACK 할 위치인 저장점을 지정하는 명령어

GRANT : 데이터베이스 사용자에게 사용 권한 부여

- **GRANT** 권한 리스트 **ON** 개체 **TO** 사용자 [WITH GRANT OPTION];

***WITH GRANT OPTION** : 부여받은 권한을 다른 사용자에게 다시 부여할 수 있는 권한

ex) REVOKE GRANT OPTION FOR UPDATE ON 고객(테이블) FROM 홍길동 CASCADE;

REVOKE : 데이터베이스 사용자의 사용 권한 취소

- **REVOKE** [GRANT OPTION FOR] 권한 리스트 **ON** 개체 FROM 사용자 [CASCADE];

***GRANT OPTION FOR** : 다른 사용자에게 권한을 부여할 수 있는 권한을 취소

ex) GRANT UPDATE ON 고객(테이블) TO 홍길동 WITH GRANT OPTION;

*테이블 R1, R2 에 대하여 다음 SQL 문의결과는?

```
(SELECT 학번 FROM R1)
INTERSECT
(SELECT 학번 FROM R2)
```

[R1] 테이블

학번	학점 수
20201111	15
20202222	20

[R2] 테이블

학번	과목번호
20202222	CS200
20203333	CS300

INTERSECT 교집합이므로

학번
20202222

3 트랜잭션(Transaction)

- 데이터베이스에서 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산
- 데이터베이스의 상태를 변환시키기 위하여 논리적 기능을 수행하는 하나의 작업 단위

트랜잭션의 특성 #ACID

1) 원자성(Atomicity)

- **트랜잭션 연산은 모두 실행되거나, 모두 실행되지 않아야 한다. (All or Nothing)**
- **Commit 과 Rollback 명령어에 의해 보장 받는다.**

2) 일관성(Consistency)

- 시스템이 가지고 있는 고정요소는 트랜잭션 수행 전과 트랜잭션 수행 완료 후의 상태가 같아야 한다.

3) 독립성(Isolation)

- 하나로 둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행 중에 다른 트랜잭션의 연산이 끼어들 수 없음을 의미한다.

4) 영속성(Durability)

- 성공적으로 완료된 트랜잭션 결과는 시스템이 고장나더라도 영구적으로 반영되어야 한다.

CRUD 매트릭스(CRUD 분석)

- 데이터베이스에 영향을 주는 생성, 읽기, 갱신, 삭제 연산으로 프로세스와 테이블 간에 매트릭스를 만들어서 트랜잭션을 분석하는 것

회복(Recovery)

- 트랙잭션을 수행하는 도중 장애로 인해 손상된 데이터베이스를 손상되기 전의 정상적인 상태로 복구시킨다.

[4] 관계 대수, 관계 해석

관계 대수(Relational Algebra)

- 릴레이션 조작을 위한 연산의 집합으로 피연산자와 결과가 **모두 릴레이션**이다.
- **질의에 대한 해를 구하기 위해 수행해야 할 연산의 순서를 명시한다.**
- **일반 집합 연산과 순수 관계 연산으로 구분된다.**
- 관계형 데이터베이스에서 원하는 정보와 그 정보를 검색하기 위해서 **어떻게(How) 유도하는가를 기술하는 절차적인 언어이다.**

*다음의 관계 대수식을 SQL 질의로 옳게 표현 한 것은? **select A from r1, r2 where P;**

$$\pi_A(\sigma_p(r1 \times r2))$$

● 순수 관계 연산자

1) 셀렉트(Select, σ)

- 조건을 만족하는 릴레이션의 **수평적 부분 집합**으로 구성한다.
- 릴레이션의 행에 해당하는 튜플들을 구하는 것이므로 수평 연산이라 함
- 릴레이션에서 조건을 만족하는 튜플 반환

2) 프로젝트(Project, π)

- 릴레이션의 **일부 속성만 추출하여 중복되는 튜플은 제거**한 후 새로운 릴레이션을 만든다.
- 속성들의 부분 집합, 중복은 제거됨(**수직 연산**)
- 릴레이션에서 주어진 속성들의 값으로만 구성된 튜플 반환

3) 조인(Join, ▷◁)

- 두 개의 릴레이션이 **공통으로 가지고 있는 속성을 이용**하여 두 개의 릴레이션을 하나로 합쳐서 새로운 릴레이션을 만든다.

4) 디비전(Division, ÷)

- R 릴레이션에서 S 릴레이션의 속성 도메인 값과 일치하는 R 릴레이션의 튜플들을 찾아낸다.

● 일반 집합 연산자

1) 합집합(Union, ∪) : 두 개의 릴레이션의 합이 추출되고, 중복은 제거됨 (ex. R∪S)

2) 교집합(Intersection, ∩) : R 릴레이션과 S 릴레이션의 중복되는 값들만 추출 (ex. R∩S)

3) 차집합(Difference, —) : R 릴레이션에서 S 릴레이션에 중복되지 않는 값들만 추출 (ex. R – S)

4) 카티션 프로덕트(Cartesian product, X) : 두 릴레이션의 가능한 모든 튜플들의 집합.
차수(Degree)는 더하고, 카디널리티(Cardinality)는 곱해서 값을 구함 (ex. R x S)

***다음 R 과 S 두 릴레이션에 대한 Division 연산의 수행 결과는?**

R

D1	D2	D3
a	1	A
b	1	A
c	2	A
d	2	B

S

D2	D3
1	A

정답 :

D1
a
b

***두 릴레이션 RI 과 R2 의 카티션 프로덕트(cartesian product) 수행 결과는?**

R1

학년
1
2
3

R2

학과
컴퓨터
국문
수학

정답 :

학년	학과
1	컴퓨터
1	국문
1	수학
2	컴퓨터
2	국문
2	수학
3	컴퓨터
3	국문
3	수학

*릴레이션 R 의 차수가 4 이고 카디널리티가 5 이며, 릴레이션 S 의 차수가 6 이고 카디널리티가 7 일 때, 두 개의 릴레이션을 카티션 프로덕트한 결과의 새로운 릴레이 션의 차수와 카디널리티는? **10, 35**

관계 해석(Relational Calculus)

- 코드(Codd)가 수학의 **프레디킷 해석**(Predicate Calculus)에 기반을 두고 관계 데이터베이스를 위해 제안함
- 원하는 정보가 **무엇(What)**이라는 것만 정의하는 **비절차적인** 언어이다. (계산 수식의 유연적 사용),
- 기본적으로 관계해석과 관계대수는 관계 데이터베이스를 처리하는 기능과 능력면에서 동등하다.
- 관계대수로 표현한 식은 관계해석으로 표현할 수 있다.
- 관계 데이터의 연산을 표현하는 방법으로, 원하는 정보를 정의할 때는 계산 수식을 사용한다.
- **튜플 관계 해석, 도메인 관계 해석**이 있다.

연산자

- OR 연산(V) : 원자식 간 "또는"이라는 관계로 연결
- AND 연산(∧) : 원자식 간 "그리고"라는 관계로 연결
- NOT 연산(¬) : 원자식에 대해 부정

정량자

- 전칭 정량자(Universal Quantifier, ∀) : 모든 가능한 튜플 "For All". # All 의 'A'를 뒤집은 형태
  - **'모든 것에 대하여(for all)' = ∀**
- 존재 정량자(Existential Quantifier, ∃) : 어떤 튜플 하나라도 존재 "There Exists". # Exists 의 'E'를 뒤집은 형태
  - **(**There Exists**) = ∃**

5 정규화(Normalization)

- 정규화는 데이터베이스의 **논리적 설계** 단계에서 수행한다.
- 중복을 배제하여 삽입, 삭제, 갱신 이상의 발생을 방지한다.
- 데이터 삽입 시 릴레이션을 재구성할 필요성을 줄인다.
- 데이터 구조의 안정성 최대화
- 수정, 삭제 시 이상현상의 최소화
- 테이블 불일치 위험의 최소화
- ~~중복 데이터의 활성화~~
- 어떠한 릴레이션이라도 데이터베이스 내에서 표현 가능하게 만든다.
- 데이터 삽입시 릴레이션을 재구성할 필요성을 줄인다.
- 효과적인 검색 알고리즘을 생성할 수 있다.

정규화 과정 #도부이결다조

● **1NF (제 1 정규형)**
- 릴레이션에 속한 모든 **도메인이 원자 값(Atomic Value)** 만으로 되어 있는 정규형

● **2NF (제 2 정규형)**
- 기본키가 아닌 모든 속성이 기본키에 대해 완전 함수적 종속을 만족하는 **부분적 함수 종속을 제거**한 정규형
- **키가 아닌 모든 속성이 기본키에 대하여 완전 함수적 종속 관계를 만족해야 한다.**

● **3NF (제 3 정규형)**
- 기본키가 아닌 모든 속성이 기본키에 대해 **이행적 함수 종속을 제거**한 정규형
- **A → B 이고 B → C 일 때 A → C 를 만족하는 관계(=이행 규칙, 이행적 함수 종속 관계)**

*어떤 릴레이션 R 에서 X 와 Y 를 각각 R 의 애트리뷰트 집합의 부분 집합이라고 할 경우 애트리뷰트 X 의 값 각각에 대해 시간에 관계없이 항상 애트리뷰트 Y 의 값이 오직 하나만 연관되어 있을 때 **Y 는 X 에 함수 종속**이라고 한다. 이 함수 종속의 표기는? **X → Y**

● **BCNF (보이스/코드 정규형)**
- 릴레이션 R 에서 **결정자가 후보키가 아닌 함수 종속 제거**

● **4NF (제 4 정규형)**
- 릴레이션 R 에 **다중치 종속 제거(다치 종속 제거)**

● **5NF (제 5 정규형)**
- 후보키를 통하지 않는 **조인 종속(JD : Join Dependency) 제거**해야 만족하는 정규형

***릴레이션 R 의 모든 결정자(determinant)가 후보키이면 릴레이션 R 은 어떤 정규형에 속하는가?★**
- 보이스/코드 정규형

*위쪽 릴레이션을 아래쪽 릴레이션으로 정규화를 하였을 때 어떤 정규화 작업을 한 것인가? **제 1 정규형**

국가	도시
대한민국	서울, 부산
미국	워싱턴, 뉴욕
중국	베이징

↓

국가	도시
대한민국	서울
대한민국	부산
미국	워싱턴
미국	뉴욕
중국	베이징

6 이상(Anomaly)

- 데이터의 중복으로 인하여 관계연산을 처리할 때 예기치 못한 곤란한 현상이 발생하는 것이다.
- 데이터 속성 간의 종속성에 대한 엄밀한 고려없이 잘못 설계된 데이터베이스에서는 데이터 처리 연산 수행 시 각종 이상 현상이 발생할 수 있다.
- 릴레이션 조작 시 데이터들이 불필요하게 **중복**되어 예기치 않게 발생하는 곤란한 현상을 의미한다.

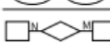
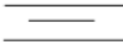
이상의 종류

- **삽입 이상(Insertion Anomaly)**
  - 릴레이션에 데이터를 삽입할 때 의도와 상관없이 원하지 않은 값들도 함께 삽입된다..
- **삭제 이상(Deletion Anomaly)**
  - 릴레이션에서 한 튜플을 삭제할 때 의도와는 상관없는 값들도 함께 삭제되는 연쇄 삭제된다.
- **갱신 이상(Update Anomaly)**
  - 릴레이션에서 튜플에 있는 속성 값을 갱신할 때 일부 튜플의 정보만 갱신되어 정보에 모순이 생긴다.
- ~~검색 이상~~
- ~~종속 이상~~

7 E-R 다이어그램(개체-관계)

- 개념적 데이터 모델의 가장 대표적인 것
- 1976 년 피터 첸(Peter Chen)에 의해 제안되고 기본적인 구성 요소가 정립됨
- 데이터를 개체(Entity), 속성(AttributE), 관계(Relationship)으로 묘사

피터 첸 표기법

기호	기호이름	의미
	사각형	· 개체(Entity) 타입
	마름모	· 관계(Relationship) 타입
	타원	· 속성(Attribute)
	이중 타원	· 다중값 속성(복합 속성)
	밑줄 타원	· 기본키 속성
	복수 타원	· 복합 속성 예) 성명과 성과 이름으로 구성
	관계	· 1:1, 1:N, N:M 등의 개체 간 관계에 대한 대응수를 선 위에 기술함
	선 링크	· 개체 타입과 속성을 연결

*개체-관계 모델(E-R)의 그래픽 표현

- 개체타입 – 사각형
- 속성 – 원형(타원)

- 관계타입 - 마름모
- 연결 - 선
- 개체타입과 속성을 연결 – 선

8 관계형 데이터베이스 (릴레이션, 튜플, 애트리뷰트)

릴레이션

- 모든 속성 값은 원자 값을 갖는다.
- 한 릴레이션에 포함된 튜플은 모두 상이하다.
- 한 릴레이션에 포함된 튜플 사이에는 순서가 없다.
- 한 릴레이션을 구성하는 속성 사이에는 순서가 상관이 없다.
- 튜플들의 삽입, 삭제 등의 작업으로 인해 릴레이션은 시간에 따라 변한다.
- 한 릴레이션에 포함된 튜플들은 모두 상이하다.
- 애트리뷰트는 논리적으로 쪼갤 수 없는 원자값으로 저장한다.
- ~~한 릴레이션에 포함된 튜플 사이에는 순서가 있다.~~

튜플(Tuple), 행(Row), 레코드(Record)

- 속성의 모임으로 구성됨
- 파일 구조상 레코드(실제 데이터)와 같은 의미
- **튜플의 수 = 카디널리티(Cardinality) = 기수 = 대응수**

속성(Attribute), 열(Column), 필드(Field)

- **개체의 특성을 기술한다.**
- **데이터베이스를 구성하는 가장 작은 논리적 단위이다.**
- **파일 구조상 데이터 항목 또는 데이터 필드에 해당된다.**
- **속성의 수 = 디그리(Degree) = 차수 = 애트리뷰트의 수**

도메인(Domain)

- **하나의 애트리뷰트가 가질 수 있는 원자값들의 집합을 의미하는 것**
- **하나의 속성이 가질 수 있는 같은 타입의 모든 값의 집합으로 각 속성의 도메인은 원자값을 갖는다.**
- ex) 성별 속성(Attribute)의 도메인은 '남', '여'로 그 외의 값은 입력될 수 없음(일반적)

*A1, A2, A3 3 개 속성을 갖는 한 릴레이션에서 A1 의 도메인은 3 개 값, A2 의 도메인은 2 개 값, A3 의 도메인은 4 개 값을 갖는다. 이 릴레이션에 존재할 수 있는 가능한 튜플(Tuple)의 최대 수 : **24 (3*2*4)**

*한 릴레이션 스키마가 4 개 속성, 2 개 후보키 그리고 그 스키마의 대응 릴레이션 인스턴스가 7 개 튜플을 갖는다면 그 릴레이션의 차수(degree) : **4**

9 키(Key)

후보키(Candidate Key)

- 릴레이션에 있는 모든 튜플에 대해 **유일성**과 **최소성**을 만족시켜야 한다.
- 튜플을 유일하게 식별하기 위해 사용하는 속성들의 부분집합, 즉 기본키로 사용할 수 있는 속성들
- 모든 릴레이션에는 반드시 하나 이상의 후보키가 존재

기본키(Primary Key)

- 후보키 중에서 특별히 선정된 주키(Main Key)로, 중복된 값과 NULL 값을 가질 수 없음
- 후보키의 성질인 유일성과 최소성을 가지며 튜플을 식별하기 위해 반드시 필요한 키

대체키(Alternate Key)

- 기본키를 제외한 나머지 후보키

슈퍼키(Super Key)

- 한 릴레이션 내에 있는 속성들의 집합으로 구성된 키로서, 릴레이션을 구성하는 모든 튜플에 대한 **유일성은 만족시키지만 최소성은 만족시키지 못한다.**

외래키(Foreign Key)

- 다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합
- 참조되는 릴레이션의 기본키와 대응돼 릴레이션 간의 참조 관계를 표현

10 무결성(Integrity)

무결성 규정(Integrity Rule)

- 데이터가 만족해야 될 제약 조건, 규정을 참조할 때 사용하는 식별자 등의 요소가 포함될 수 있다.
- 무결성 규정의 대상으로는 도메인, 키, 종속성 등이 있다.
- 릴레이션 무결성 규정(Relation Integrity Rules)은 릴레이션을 조작하는 과정에서의 의미적 관계를 명세함
- ~~정식으로 허가 받은 사용자가 아닌 불법적인 사용자에 의한 갱신으로부터 데이터베이스를 보호하기 위한 규정이다.~~

1) 개체 무결성 (Entity Integrity)

- 기본키는 NULL 값을 가져서는 안되며, 릴레이션 내에 오직 하나의 값만 존재해야 한다.
- 릴레이션에서 기본 키를 구성하는 속성은 널(Null)값이나 중복 값을 가질 수 없다.
- 기본키의 속성 값이 널(NULL)값이 아닌 원자 값을 갖는 성질
- 기본키에 속해 있는 애트리뷰트는 널값이나 중복값을 가질 수 없다.

2) 도메인 무결성 (Domain Integrity)

- 릴레이션 내의 튜플들이 각 속성(Attribute)의 도메인에 지정된 값 만을 가져야 함
- 주어진 속성 값이 정의된 도메인에 속한 값이어야 한다.

3) 참조 무결성 (Referential Integrity)

- 릴레이션 R1 에 속한 애튜리뷰트의 조합인 외래키를 변경하려면 이를 참조하고 있는 릴레이션 R2 의 기본키도 변경해야 한다.

4) 사용자 정의 무결성 (User-Defined Integrity)

- 속성 값들이 사용자가 정의한 제약 조건에 만족해야 함

11 병행 제어

로킹 단위

- 로킹의 대상이 되는 객체의 크기를 로킹 단위라고 한다.
- 데이터베이스, 파일, 레코드 등은 로킹 단위가 될 수 있다.
- 한꺼번에 로킹할 수 있는 객체의 크기를 로킹 단위라고 한다.

- 로킹 단위가 커지면 로크의 수가 적어지고, 로킹 오버헤드가 감소하고, 병행성 수준이 낮아지고, 데이터베이스 공유도가 감소한다. 병행 제어 기법이 간단해진다.
- 로킹 단위가 작으면 로크의 수가 많아지고, 로킹 오버헤드가 증가하고, 병행성 수준이 높아지고, 데이터베이스 공유도가 증가한다. 병행 제어 기법이 복잡해진다.

데이터베이스 병행제어 기법의 종류

1) 로킹 기법

- 잠금(Lock)을 설정한 트랜잭션이 해제(Unlock)할 때까지 독점적으로 사용할 수 있게 상호배제 기능을 제공

2) 타임스탬프 기법(Timestamp)

- 동시성 제어를 위한 직렬화 기법으로 트랜잭션 간의 **처리 순서**를 미리 정하는 방법

3) 낙관적 검증(최적 병행 수행기법)

- 일단 트랜잭션을 수행하고 난 후 트랜잭션 종료시 검증을 수행하여 데이터베이스에 반영하는 기법

4) 다중 버전 기법

- 트랜잭션의 타임스탬프와 접근하려는 데이터의 타임스탬프를 비교하여 직렬 가능성이 보장되는 적절한 버전을 선택하여 접근하도록 하는 기법

~~**시분할 기법**~~

12 인덱스(Index)

- 인덱스의 기본 목적은 검색 성능을 최적화하는 것으로 볼 수 있다.
- B-트리 인덱스는 분기를 목적으로 하는 Branch Block 을 가지고 있다.
- BETWEEN 등 범위(Range) 검색에 활용될 수 있다.
- 데이터베이스 성능에 많은 영향을 주는 DBMS 의 구성 요소
- 테이블과 클러스터에 연관되어 독립적인 저장 공간을 보유
- ~~시스템이 자동으로 생성하여 사용자가 변경할 수 없다.~~

13 뷰(View)

- DBA 는 보안 측면에서 뷰를 활용할 수 있다.
- 뷰 위에 또 다른 뷰를 정의할 수 있다.
- ~~뷰 자체로 인덱스를 가짐~~
- **독립적인 인덱스를 가질 수 없다.**
- **뷰에 대한 삽입, 갱신, 삭제 연산 시 제약사항이 따른다.**
- 뷰가 정의된 기본 테이블이 제거되면 뷰도 자동적으로 제거된다.
- 뷰의 정의는 ALTER 문을 이용하여 변경할 수 없다.
- ~~뷰의 정의는 기본 테이블과 같이 ALTER 문을 이용하여 변경한다. => 뷰를 삭제하고 재생성해야 한다~~
- 데이터 보안 용이
- 논리적 독립성 제공
- 사용자 데이터 관리 용이
- 뷰는 CREATE 문을 사용하여 정의한다.
- 뷰는 데이터의 논리적 독립성을 제공한다.
- 뷰를 제거할 때에는 DROP 문을 사용한다.
- ~~뷰는 저장장치 내에 물리적으로 존재한다. => 논리적으로 구성되어 있음~~

14 시스템 카탈로그(System Catalog)

- 시스템 자신이 필요로 하는 스키마 및 여러 객체에 관한 정보를 포함하고 있는 시스템 데이터베이스이다.
- 시스템 카탈로그에 저장되는 내용을 **메타 데이터**라고도 한다.
- 시스템 카탈로그는 DBMS 가 스스로 생성하고 유지한다.
- 데이터베이스에 포함되는 데이터 객체에 대한 정의나 명세에 대한 정보를 유지관리한다.
- DBMS 가 스스로 생성하고 유지하는 데이터베이스 내의 특별한 테이블의 집합체이다.
- ~~사용자가 직접 시스템 카탈로그의 내용을 갱신하여 데이터베이스 무결성을 유지한다.~~
- ~~시스템 카탈로그의 갱신은 무결성 유지를 위하여 SQL 을 이용하여 사용자가 직접 갱신하여야 한다.~~
- ~~일반 사용자도 SQL 을 이용하여 시스템 카탈로그를 직접 갱신할 수 있다. => **조회는 가능하지만 갱신 불가**~~
- DBMS 는 자동적으로 시스템 카탈로그 테이블들의 행을 삽입, 삭제, 수정한다.

데이터 디렉터리(Data Directory)

- 데이터 사전(Data Dictionary)에 수록된 데이터를 실제로 접근하는 데 필요한 정보를 관리 유지하는 시스템
- 시스템만 접근할 수 있음
- 시스템 카탈로그는 사용자와 시스템 모두 접근할 수 있음

15 파티셔닝 테이블

***파티션(Partition) # 범해조리**

1) 범위 분할, 레인지 파티셔닝(Range Partitioning)

- 지정한 열의 값을 기준으로 분할 ex) 일별, 월별, 분기별 등

2) 해시 분할, 해시 파티셔닝(Hash Partitioning)

- 해시 함수에 따라 데이터 분할

3) 조합 분할, 컴포지트 파티셔닝(Composite Partitioning)

- 범위분할 이후 해시 함수를 적용하여 다시 분할 ex) 범위분할 + 해시분할

4) 리스트 파티셔닝(List Partitioning)

- 미리 정해진 그룹핑 기준에 따라 분할

***병렬 데이터베이스 환경 중 수평 분할에서 활용되는 분할 기법이 아닌 것은?**

- 라운드-로빈
- 범위 분할
- 해시 분할
- ~~예측 분할~~

16 분산 데이터베이스의 투명성

분산 데이터베이스의 목표 #병이 위중 복분장

● **위치 투명성(Location Transparency)**

- 하드웨어와 소프트웨어의 물리적 위치를 사용자가 알 필요가 없다.

● **중복 투명성(Replication Transparency, 복제 투명성)**

- 동일 데이터가 여러 곳에 중복되어 있더라도 사용자는 마치 하나의 데이터만 존재하는 것처럼 사용하고, 시스템은 자동으로 여러 자료에 대한 작업을 수행

● **병행 투명성(Concurrency Transparency)**

- 다중 사용자들이 자원들을 자동으로 공유할 수 있다.
- 다수의 트랜잭션들이 동시에 실현되더라도 그 트랜잭션의 결과는 영향을 받지 않음

● **분할 투명성(Division Transparency)**
- 하나의 논리적 릴레이션이 여러 단편으로 분할되어 각 단편의 사본이 여려 시스템에 저장되어 있음을 인식할 필요가 없음

● **장애 투명성(Failure Transparency)**
- 데이터베이스의 분산된 물리적 환경에서 특정 지역의 컴퓨터 시스템이나 네트워크에 장애가 발생해도 데이터 무결성이 보장된다

● **이주 투명성**
- 자원들이 한 곳에서 다른 곳으로 이동하면 자원들의 이름도 자동으로 바꾸어지지 **않는다.**

● **복제 투명성**
- 사용자에게 통지 할 필요 없이 시스템 안에 파일들과 자원들의 부가적인 복사를 자유로이 할 수 있다.

17 반정규화(Denormalization)

- 정규화된 엔티티, 속성, 관계를 시스템의 **성능** 향상과 개발 운영의 **단순화**를 위해 중복, 통합, 분리 등을 수행하는 데이터 모델링 기법

반정규화 방법
- 테이블 통합
  - 1:1 관계 테이블 통합
  - 1:N 관계 테이블 통합
  - 슈퍼타입/서브타입 테이블 통합
- 테이블 분할 (기본키의 유일성 관리가 어려워짐)
  - 수평 분할
  - 수직 분할
- 중복 테이블 추가
  - 집계 테이블의 추가
  - 진행 테이블의 추가
  - 특정 부분만을 포함하는 테이블의 추가
  - ~~빌드 테이블의 추가~~
- 중복 속성 추가
  - 자주 사용하는 속성을 하나 더 추가하는 것

18 데이터 모델

데이터 모델의 구성 요소 : 개체(Entity), 속성(Attribute), 관계(Relationship)

데이터 모델에 표시할 요소
- 논리적 데이터 구조(Structure) : 논리적인 개체 타입들 간의 관계, 데이터 구조 및 정적 성질을 표현
- 연산(Operation) : 실제 데이터를 처리하는 작업에 대한 명세로, 조작하는 기본 도구
- 제약 조건(Constraint) : DB 에 저장될 수 있는 실제 데이터의 논리적인 제약 조건
- ~~출력 구조~~

개념적 데이터 모델

- 현실 세계에 대한 인간의 이해를 돕기 위해 현실 세계에 대한 인식을 추상적 개념으로 표현하는 과정
- ex. E-R 모델

논리적 데이터 모델

- 개념적 모델링 과정에서 얻은 개념적 구조를 컴퓨터가 이해하고 처리할 수 있는 컴퓨터 세계의 환경에 맞도록 변환하는 과정
- ex. 관계 모델, 계층 모델, 네트워크 모델

물리적 데이터 모델

-

19 절차형 SQL

- 프로그래밍 언어처럼 연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL
- 일반적인 프로그래밍 언어에 비해 효율이 떨어짐
- 연속적인 작업들을 처리하는데 적합
- BEGIN ~ END 형식으로 작성되는 블록(Block) 구조로 기능별 모듈화 가능

프로시저(Procedure)

- 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업 수행, 처리 결과는 한 개 이상의 값 혹은 반환을 아예 하지 않음

트리거(Trigger)

- 데이터베이스 시스템에서 삽입, 갱신, 삭제 등의 이벤트가 발생할 때마다 관련 작업이 자동으로 수행되는 절차형 SQL

사용자 정의 함수

- 프로시저와 유사하게 SQL 을 사용해 일련의 작업을 연속적으로 처리함, 종료 시 예약어 RETURN 을 사용해 처리 결과를 단일값으로 반환

PL/SQL

- 선언부(Declare) : 실행부에서 참조할 모든 변수, 상수, CURSOR, EXCEPTION 선언
- 실행부(Begin/End) : BEGIN 과 END 사이에 기술되는 영역, 데이터를 처리할 SQL 문과 PL/SQL 블록을 기술
- 예외부(Exception) : 실행부에서 에러가 발생했을 때 문장 기술
- 장점 : 컴파일 불필요, 모듈화 기능, 절차적 언어 사용, 에러 처리
- PL/SQL 을 활용한 저장형 객체 활용
    - 저장된 프로시저, 저장된 함수, 저장된 패키지, 트리거(Trigger)

20 DMBS 접속 기술

데이터 접속(Data Mapping)

- SQL Mapping : 프로그래밍 코드 내 SQL 을 직접 입력해 DBMS 의 데이터에 접속 (JDBC, ODBC, MyBatis)
- ORM : 객체(Object)와 관계형데이터베이스(RDB)의 데이터를 연결(Mapping)하는 기술(JPA, Hibernate, Django)

SQL Mapping

- JDBC (Java Database Connectivity)
  - JAVA 언어로 다양한 종류의 데이터베이스에 접속하고 SQL 문을 수행할 때 사용되는 표준 API
  - 접속하려는 DBMS 에 대한 드라이버가 필요
- ODBC (Open Database Connectivity)
  -데이터베이스에 접근하기 위한 표준 개방형 API 로 개발 언어에 관계없이 사용 가능
  - ODBC 도 접속하려는 DMBS 에 맞는 드라이버가 필요하지만, 접속하려는 DBMS 의 인터페이스를 알지 못하더라도 ODBC 문장을 사용해 SQL 을 작성하면 ODBC 에 포함된 드라이버 관리자가 해당 DBMS 의 인터페이스에 맞게 연결해줌 → DBMS 의 종류를 몰라도 됨

ORM(Object-Relational Mapping)

- 객체(Object)와 관계형데이터베이스(RDB)의 데이터를 연결(Mapping)하는 기술
- ORM 으로 생성된 가상의 객체지향 데이터베이스는 프로그래밍 코드 또는 데이터베이스와 독립적이므로 재사용 및 유지보수 용이

ORM 프레임워크

- JAVA : JPA, Hibernate, Eclipse Link, Data Nucleus, Ebean 등
- C++ : ODB, QxOrm 등
- Python : Django, SQL Alchemy, Storm 등
- iOS : Core Date, Database Objects 등
-.NET : NHibernate, Database Objects, Dapper 등
- PHP : Doctrine, Propel, RedBean 등

21 데이터베이스 이중화

데이터베이스 이중화(Database Replication)

- 시스템 오류로 인한 데이터베이스 서비스 중단이나 물리적 손상 발생 시 이를 복구하기 위해 동일한 데이터베이스를 복제해 관리하는 것

데이터베이스 이중화의 분류

- Eager 기법 : 트랜잭션 수행 중 데이터 변경이 발생하면 이중화 된 모든 데이터베이스에 즉신 전달해 변경 내용이 즉시 적용되도록 하는 기법
- Lazy 기법 : 트랜잭션의 수행이 종료되면 변경 사실을 새로운 트랜잭션에 작성해 각 데이터베이스에 전달되는 기법 → 데이터베이스마다 새로운 트랜잭션이 수행되는 것으로 간주됨

데이터베이스 이중화 구성 방법

- 활동-대기(Active-Standby) : 한 DB 가 활동 상태로 서비스하고 있으면 다른 DB 는 대기하고 있다가 활동 DB 에 장애가 발생하면 대기 상태에 있던 DB 가 자동으로 모든 서비스를 대신 수행

→ 구성 방법 및 관리가 쉬워 많은 기업에서 이용함

- 활동-활동(Active-Active) : 두 개의 DB 가 서로 다른 서비스를 제공하다가 둘 중 한쪽 DB 에 문제가 발생하면 나머지 다른 DB 가 서비스를 제공

→ 두 DB 모두 처리를 하기 때문에 처리율이 높지만 구성 방법 및 설정이 복잡함

22 기타

쿼리 성능 최적화

RBO(Rule Based Optimizer)

- 최적화 기준 : 규칙에 정의된 우선순위
- 성능 기준 : 개발자의 SQL 숙련도
- 특징 : 실행 계획 예측이 쉬움
- 고려사항 : 개발자의 규칙 이해도, 규칙의 효율성

CBO(Cost Based Optimizer)

- 최적화 기준 : 액세스 비용
- 성능 기준 : 옵티마이저 알고리즘의 예측 성능
- 특징 : 성능 통계치 정보 활용, 예측이 복잡함
- 고려사항 : 비용 산출 공식의 정확성

데이터 지역화(Data Locality) => 구역성

- 데이터베이스의 저장 데이터를 효율적으로 이용할 수 있도록 저장하는 방법

1) 시간적 구역성

- 최근에 참조된 기억장소가 가까운 장래에 계속 참조될 가능성이 높은 특성
- Stack(스택), Subroutine(서브루틴), Loop(루프), Counting(카운팅), Totaling(집계)
- for, while 같은 반복문에 사용하는 조건 변수

2) 공간적 구역성

- 최근에 참조된 기억장소와 가까운 기억정보가 가까운 장래에 계속 참조될 가능성이 높은 특성
- Array(배열), Sequential Code(순차적 코드)
- A[0], A[1] 같은 배열에 연속 접근

3) 순차적 구역성

- 별도의 분기가 없는 한, 데이터가 기억장치에 저장된 순서대로 인출되고 실행될 가능성이 높은 특성
- 1:1, 1:N, N:M 관계 존재

데이터 지역화를 활용한 관리 기법

- 기억장치 계층구조(Hierarchy)
  - CPU → 캐시 메모리 → 메인 메모리 순서로 접근시간(Access Time)을 효과적으로 단축
- 캐시 접근시간 단축
  - 캐시 적중률(Cache Hit Ratio)의 극대화 가능
- 워킹셋(Working Set)
  - 하나의 페이지(Page)가 자주 접근하는 페이지들의 집합
  - 페이지 폴트(Page Fault)를 줄여 스레싱(Thrashing) 감소

클러스터

- 데이터 저장 시 데이터 액세스 효율을 향상시키기 위해 동일한 성격의 데이터를 동일한 데이터 블록에 저장하는 물리적 저장 방법
- 인덱스의 단점을 해결한 기법 → 분포도(Selectivity)가 넓을수록 오히려 유리함

- 분포도가 넓은 "테이블"의 클러스터링은 저장 공간의 절약이 가능
- 대량의 범위를 자주 액세스(조회)하는 경우 적용
- 인덱스를 사용한 처리 부담이 되는 넓은 분포도에 활용

<4 과목 프로그래밍 언어 활용>

1 결합도, 응집도

결합도(Coupling)

- 결합도(Coupling)는 두 **모듈간의** 상호작용, 또는 의존도 정도를 나타내는 것이다.
- 모듈간의 결합도를 약하게 하면 모듈 독립성이 향상된다.
- 결합도가 강할수록 품질이 낮으며, 시스템 구현 및 유지보수 작업이 어렵다.
- 결합도는 낮을수록 Good = 독립적인 모듈

자스제외공내 (약→강)

● **자료 결합도(Data Coupling)**
- 어떤 모듈이 다른 모듈을 호출하면서 매개 변수(파라미터)나 인수로 데이터를 넘겨주고, 호출 받은 모듈은 받은 데이터에 대한 처리 결과를 다시 돌려주는 결합도

● **스탬프 결합도(Stamp Coupling)**
- 두 모듈이 매개변수로 자료를 전달할 때, **자료구조** 형태로 전달되어 이용될 때 데이터가 결합되어 있다.

● **제어 결합도(Control Coupling)**
- **어떤 모듈이 다른 모듈의 내부 논리 조직을 제어하기 위한 목적으로 제어신호를 이용하여 통신하는 경우이며, 하위 모듈에서 상위 모듈로 제어신호가 이동하여 상위 모듈에게 처리 명령을 부여하는 권리 전도현상이 발생하게 되는 결합도**

● **외부 결합도(External Coupling)**
- 어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도 (순차적)

● **공통 결합도(Common Coupling)**
- 두 모듈이 동일한 전역 데이터를 접근한다면 공통결합 되어 있다. (전역 변수)

● **내용 결합도(Content Coupling)**
- 하나의 모듈이 직접적으로 다른 모듈의 내용을 참조할 때 두 모듈은 내용적으로 결합되어 있다고 한다.

응집도(Cohesion)

- **한 모듈 내에 있는 처리요소들 사이의 기능적인 연관 정도를 나타낸다.**
- 응집도는 낮을수록 Good = 독립적인 모듈

기순교절시논우 (강→약)

● **기능적 응집도(Functional Cohesion)**
- 모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우

● **순차적 응집도(Sequential Cohesion)**
- 모듈 내 하나의 활동으로부터 나온 출력 데이터(출력값)를 그 다음 활동의 입력 데이터로 사용할 경우

● **교환적 응집도(Communication Cohesion)**
- 동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우

● **절차적 응집도(Procedural Cohesion)**

- 모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 **순차적**으로 수행할 경우

● **시간적 응집도(Temporal Cohesion)**

- 모듈 내 구성 요소들이 서로 다른 기능을 **같은 시간대에** 함께 실행하는 경우

● **논리적 응집도(Logical Cohesion)**

- 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우

● **우연적 응집도(Coincidental Cohesion)**

- 서로 간에 어떠한 **의미 있는 연관관계도** 지니지 않은 기능 요소로 구성되는 경우이며, 서로 다른 상위 모듈에 의해 호출되어 처리상의 연관성이 없는 서로 다른 기능을 수행하는 경우

***모듈의 독립성을 높이기 위한 결합도(Coupling)와 관련한 설명으로 틀린 것은?**

- 오류가 발생했을 때 전파되어 다른 오류의 원인이 되는 파문 효과(Ripple Effect)를 최소화해야 한다.
- 인터페이스가 정확히 설정되어 있지 않을 경우 모듈 사이의 의존도는 높아지고 결합도가 증가한다.
- 다른 모듈과 데이터 교류가 필요한 경우 전역변수보다는 매개변수(Parameter)를 사용하는 것이 좋다.
- ~~모듈들이 변수를 공유하여 사용하게 하거나 제어 정보를 교류하게 함으로써 결합도를 낮추어야 한다.~~

2 OSI 7 계층

#**아(A)파(P)서(S) 티(T)내(Ne)다(Da) 피(Phy)**나다

● **응용 계층(Application Layer, 7 계층)**

- 사용자와 네트워크 간 응용서비스 연결, 데이터 생성
- **HTTP, FTP, TELNET, SMTP, SNTP, DNS**

● **표현 계층(Presentation Layer, 6 계층)**

- 구문 검색, 코드 변환, 암/복호화, 데이터 압축, 문맥 관리 기능
- **JPEG, MPEG**

● **세션 계층(Session Layer, 5 계층)**

- 연결 접속(유지), 동기 제어, 동기점(대화)
- **SSH, TLS**

● **전송 계층(Transport Layer, 4 계층)**

- **단말기 사이에 오류 수정과 흐름제어를 수행하여 신뢰성 있고 명확한 데이터를 전달하는 계층**
- **종단간 신뢰성 있고 효율적인 데이터를 전송하기 위해 오류검출과 복구, 흐름 제어를 수행하는 계층**
- 종단간(End to End) 신뢰성 있는 데이터 전송, 흐름 제어(슬라이딩 윈도우), 오류 및 혼잡 제어
- **TCP / UDP, RTCP → 세그먼트(Segment)**

● 네트워크 계층(Network Layer, 3 계층)

- 패킷을 발신지로부터 최종 목적지까지 전달하는 책임을 진다.
- 패킷에 발신지와 목적지의 논리 주소를 추가한다.
- 라우터 또는 교환기는 패킷 전달을 위해 경로를 지정하거나 교환 기능을 제공한다.
- 단말기 간 데이터 전송을 위한 최적화된 경로(라우팅) 제공
- **IP, ICMP, IGMP, ARP, RARP, RIP, OSPF → 패킷(Packet)**

라우터(Router) : 서로 다른 **네트워크** 대역에 있는 호스트들 상호간에 통신할 수 있도록 해주는 네트워크 장비

***인터페이스 보안을 위해 네트워크 영역에 적용되는 솔루션이 아닌 것은?**

- PSec, SSL, S-HTTP, ~~SMTP~~

● 데이터 링크 계층(Data Link Layer, 2 계층)

- 물리적 연결을 이용해 신뢰성 있는 정보를 전송하려고 동기화, 오류제어, 흐름제어 등의 전송에러를 제어
- 링크의 설정과 유지 및 종료를 담당하며, 노드간의 오류제어와 흐름제어 기능을 수행
- 한 노드로부터 다른 노드로 **프레임**을 전송하는 책임을 진다.
- 인접 시스템(노드) 간 물리적 연결을 이용해 데이터 전송, 동기화, 오류 및 흐름제어, 오류검출 및 재전송
- **HDLC, PPP, LLC, MAC → 프레임(Frame)**

스위치(Switch) : 브리지와 같이 LAN 과 LAN 을 연결하여 훨신 더 큰 LAN 을 만든다. (하드웨어 기반)

브리지(Bridge) : LAN 과 LAN 을 연결하거나 LAN 안에서의 컴퓨터 그룹을 연결한다.

● 물리 계층(Physical Layer, 1 계층)

- 매체 간의 전기적, 기능적, 절차적 기능 정의
- **RS-232C, X.21 → 비트(Bit)**

리피터(Repeater) : 신호가 왜곡되거나 약해질 경우 원래의 신호 형태로 재생하여 다시 전송한다

허브(Hub) : 한 사무실이나 가까운 거리의 컴퓨터들을 연결하는 장치

3 TCP/IP 프로토콜

TCP/IP 의 구조

OSI	TCP/IP	기능
응용 계층(A) 표현 계층(P) 세션 계층(S)	응용 계층	응용 프로그램 간의 데이터 송, 수신 제공 # HTTP, FTP, TELNET, SMTP / SNTP, DNS (TCP 를 사용하는 서비스 / UDP 사용 서비스)
전송 계층(T)	전송 계층	호스트들 간의 신뢰성 있는 통신 제공 # TCP / UDP, RTCP
네트워크 계층(Ne)	인터넷 계층	데이터 전송을 위한 주소 지정, 경로 설정(Routing) 제공 # IP, ICMP, IGMP, ARP, RARP, RIP, OSPF
데이터 링크 계층(Da) 물리 계층(Phy)	네트워크 액세스 계층	실제 데이터(프레임)를 송, 수신하는 역할 # Ethernet, IEEE 802, HDLC, X.25, RS-232C, ARQ

응용 계층의 주요 프로토콜

1) HTTP(Hypertext Transfer Protocol) - HTML 문서를 송, 수신하기 위한 표준 프로토콜
2) FTP(File Transfer Protocol) - 파일을 주고받을 수 있는 원격 파일 전송 프로토콜
3) TELNET - 원격지 컴퓨터에 접속하여 자신의 컴퓨터처럼 사용할 수 있도록 해주는 서비
4) SMTP(Simple Mail Transfer Protocol) - 전자 우편을 교환하는 서비스
5) SNTP(Simple Network Management Protocol) - TCP/IP 의 네트워크 관리 프로토콜
6) DNS(Domain Name System) - 도메인 네임을 IP 주소로 매핑하는 시스템

전송 계층의 주요 프로토콜

1) TCP(Transmission Control Protocol)

- OSI 7 계층의 전송 계층(4 계층)에 해당한다.

- 신뢰성이 있는 연결 지향형 전달 서비스이다.
- 전이중(Full Duplex) 방식의 양방향 가상회선을 제공한다.
- 스트림 전송 기능을 제공한다.
- 순서제어, 오류제어, 흐름제어 기능을 제공한다.
- 전송 데이터와 응답 데이터를 함께 전송할 수 있다.
- 흐름 제어(Flow Control)의 기능을 수행한다.
- ~~기본 헤더 크기는 100byte 이고 160byte 까지 확장 가능하다. => 기본 헤더 크기는 최소 20byte 최대 60byte~~
- ~~인접한 노드 사이의 프레임 전송 및 오류를 제어한다. => 데이터 링크~~

***TCP 헤더**

- 순서번호(Sequence Number)는 전달하는 바이트마다 번호가 부여된다.
- 수신번호확인(Acknowledgement Number)은 상대편 호스트에서 받으려는 바이트의 번호를 정의한다.
- 체크섬(Checksum)은 데이터를 포함한 세그먼트의 오류를 검사한다.
- ~~윈도우 크기는 송수신 측의 버퍼 크기로 최대크기는 32767bit 이다. => 16 비트로 2^16 = 65536byte~~

2) UDP(User Datagram Protocol)

- 양방향 연결형 서비스를 제공한다.
- 송신중에 링크를 유지관리하므로 신뢰성이 높다.
- 흐름제어나 순서제어가 없어 전송속도가 빠르다.
- 신뢰성보다는 속도가 중요시되는 네트워크에서 사용
- 실시간 전송에 유리
- **비연결성 서비스 제공**
- 단순한 헤더구조로 오버헤드 적음

***UDP 헤더** - Source Port, Destination Port, Length, Checksum, Data

***RTP 프로토콜 (Real-Time Transport Protocol)**

- **실시간 특성**을 가지는 데이터의 종단간 전송을 제공해주는 UDP 기반의 프로토콜이다.

3) RTCP(Real-Time Control Protocol)

- 패킷의 전송 품질을 제어하기 위한 제어 프로토콜
- 세션에 참여한 각 참여자들에게 주기적으로 제어 정보 전송
- 데이터 패킷과 제어 패킷의 다중화(Multiplexing) 제공 → 하위 프로토콜
- 최소한의 제어와 인증 기능만을 제공하고 항상 32 비트의 경계로 끝남

인터넷 계층의 주요 프로토콜

1) IP(Internet Protocol)

- OSI 7 계층의 네트워크 계층(3 계층)에 해당한다.
- 데이터 그램을 기반으로 하는 비연결형 서비스 제공
- 패킷의 분해/조립, 주소 지정, 경로 선택 기능(Routing) 제공

2) ICMP(Internet Control Message Protocol)

- IP 와 조합하여 통신중에 발생하는 오류의 처리와 전송 경로 변경 등을 위한 제어 메시지를 관리
- 헤더는 8Byte 로 구성됨

3) IGMP(Internet Group Management Protocol)

- 멀티캐스트를 지원하는 호스트나 라우터 사이에서 멀티캐스트 그룹 유지를 위해 사용됨

4) ARP(Address Resolution Protocol)

- TCP/IP 에서 사용되는 논리주소를 물리주소로 변환시켜 주는 프로토콜
- TCP/IP 네트워크에서 IP 주소를 MAC 주소로 변환하는 프로토콜
- 네트워크에서 두 호스트가 성공적인 통신을 위하여 각 하드웨어의 물리적인 주소문제를 해결해 줄 수 있다.
- ARP 캐시를 사용하므로 캐시에서 대상이 되는 IP 주소의 MAC 주소를 발견하면 이 MAC 주소가 통신을 위해 사용된다.
- ~~ARP 캐시를 유지하기 위해서는 **TTL 값이 0 이 되면** 이 주소는 ARP 캐시에서 영구히 보존된다.~~
~~=> TTL(주소의 유효기간) 값이 0 이 되면 해당 주소는 폐기된다.~~

5) RARP(Reverse Address Resolution Protocol)

- ARP 와 반대로 물리적 주소(MAC Address)를 IP 주소로 변환하는 기능을 함
- MAC 주소 → IP 주소

네트워크 액세스 계층의 주요 프로토콜

1) Ethernet(IEEE 802.3) - CSMA/CD 방식의 LAN

2) IEEE 802 - LAN 을 위한 표준 프로토콜

3) HDLC - 비트 위주의 데이터 링크 제어 프로토콜

***HDLC(HIGI-level Data Link Control) 프레임형식**

플래그	주소 영역	제어 영역	정보 영역	FCS	플래그

***HDLC 프레임 구조 중 헤더를 구성하는 플래그(flag)에 대한 설명으로 틀린 것은?**

1. ~~프레임의 최종목적 주소를 나타낸다.~~
2. 동기화에 사용된다.
3. 프레임의 시작과 끝을 표시한다.
4. 01111110 의 형식을 취한다.

4) X.25

- 공중데이터망에서 패킷형 터미널을 위한 DCE 와 DTE 사이의 접속규격을 나타내는 것
- DTE 와 DCE 간 상호접속 및 통신절차 규정
- 패킷 교환망을 통한 DTE 와 DCE 간의 인터페이스를 제공하는 프로토콜

5) RS-232C

- 공중 전화 교환망(PSTN)을 통한 DTE 와 DCE 간의 인터페이스를 제공하는 프로토콜

MQTT 프로토콜

- TCP/IP 기반 네트워크에서 동작하는 **발행-구독 기반의 메시징** 프로토콜로 최근 IoT 환경에서 자주 사용된다.
- 사물통신, 사물인터넷과 같이 대역폭이 제한된 통신환경에 최적화하여 개발된 **푸시기술** 기반의 경량 메시지 전송 프로토콜
- 메시지 매개자(Broker)를 통해 송신자가 특정 **메시지를 발행**하고 수신자가 **메시지를 구독**하는 방식
- IBM 이 주도하여 개발

4 IP

IP 주소(Internet Protocol Address)

- 인터넷에 연결된 모든 컴퓨터 자원을 구분하기 위한 고유한 주소
- 숫자로 8 비트씩 4 구역으로 총 **32 비트**로 구성됨

IP 주소의 분류

- A Class : (**0**.0.0.0 ~**127**.255.255.255) (국가나 대형 통신망)
- B Class : (**128**.0.0.0 ~**191**.255.255.255) (중대형 통신망)
- **C Class : (192.0.0.0 ~ 223.255.255.255)** (소규모 통신망)
- D Class : (**224**.0.0.0 ~ **239**.255.255.255) (멀티캐스트용)
- E Class : (**240**.0.0.0 ~ **255**.255.255.255) (연구용. 실험적 주소이며 공용되지 않음)

서브네팅(Subnetting)

- 할당된 네트워크 주소를 다시 여러 개의 작은 네트워크로 나누어 사용하는 것
- **서브넷 마스크(Subnet Mask)** : 4 바이트의 IP 주소 중 네트워크 주소와 호스트 주소를 구분하기 위한 비트로, 이를 변경해 네트워크 주소를 여러 개로 분할해 사용

*CIDR(Classless Inter-Domain Routing) 표기로 203.241.132.82/27 과 같이 사용되었다면, 해당 주소의 서브넷 마스크(subnet mask)는? 11111111.11111111.11111111.11100000 = **255.255.255.224**

*128.107.176.0/22 네트워크에서 호스트에 의해 사용될 수 있는 서브넷 마스크는?
11111111 11111111 11111100 00000000 = **255.255.252.0**

*192.168.1.0/24 네트워크를 FLSM 방식을 이용하여 4 개의 Subnet 으로 나누고 IP Subnet-zero 를 적용했다. 이 때 Subnetting 된 네트워크 중 4 번째 네트워크의 4 번째 사용가능한 IP 는 무엇인가? **192.168.1.196**

*200.1.1.0/24 네트워크를 FLSM 방식을 이용하여 10 개의 Subnet 으로 나누고 ip subnet-zero 를 적용했다. 이때 서브네팅된 네트워크 중 10 번째 네트워크의 broadcast IP 주소는? **200.1.1.159**

IPv6(Internet Protocol version 6)

- IPv4 보다 **보안성**이 강화되었다.
- 보안과 인증 확장 헤더를 사용함으로써 인터넷 계층의 보안기능을 강화하였다.
- IPv6 확장 헤더를 통해 네트워크 기능 **확장**이 용이하다.
- 애니캐스트(Anycast)는 하나의 호스트에서 그룹 내의 가장 가까운 곳에 있는 수신자에게 전달하는 방식이다.
- **128 비트** 주소체계를 사용한다.
- 멀티미디어의 실시간 처리가 가능하다.
- 자동으로 네트워크 환경구성이 가능하다.
- ~~멀티캐스팅(Multicast) 대신 브로드캐스트(Broadcast)를 사용한다.~~
- ~~패킷 크기가 64Kbyte 로 고정되어 있다.~~
- IPv6 의 패킷 크기는 임의로 큰 크기의 패킷을 주고 받을 수 있다. (IPv4 의 패킷 크기가 64Kbyte 로 제한)
- Traffic Class, Flow Label 을 이용하여 등급별, 서비스별로 패킷을 구분할 수 있어 품질 보장(QoS)이 용이

주소체계

- IPv4 : 유니캐스트, 멀티캐스트, **브로드캐스트**
- IPv6 : 유니캐스트, 멀티캐스트, **애니캐스트**

***IPv6 의 헤더 항목이 아닌 것은? ★**

1. Flow label
2. Payload length
3. HOP limit
4. ~~Section~~

***IETF 에서 고안한 IPv4 에서 IPv6 로 전환(천이)하는데 사용되는 전략이 아닌 것은?**

1. Dual stack
2. Tunneling
3. Header translation
4. ~~Source routing~~

5 운영체제

- 다중 사용자와 다중 응용프로그램 환경에서 자원의 현재 상태를 파악하고 자원 분배를 위한 스케줄링 담당
- CPU, 메모리 공간, 기억 장치, 입출력 장치 등의 자원을 관리한다.
- 입출력 장치와 사용자 프로그램을 제어한다.
- 사용자의 편리한 환경 제공
- 처리능력 및 신뢰도 향상
- 컴퓨터 시스템의 성능 최적화
- ~~운영체제의 종류로는 매크로 프로세서, 어셈블러, 컴파일러 등이 있다.~~
- ~~언어번역기능을 통한 실행 가능한 프로그램 생성 => 컴파일러, 어셈블러, 인터프리터가 담당한다~~
- 사용자 > 응용 프로그램 > 유틸리티 > **운영체제(OS)** > 하드웨어

운영체제를 기능에 따라 분류할 경우 제어 프로그램

- **데이터 관리 프로그램** : 주/보조기억장치 사이의 데이터 전송, 파일과 데이터를 처리 유지 보수 기능 수행
- **작업 제어 프로그램** : 작업의 연속 처리를 위한 스케줄 및 시스템 자원 할당 등을 담당
- **감시 프로그램** : 프로그램과 시스템 작동상태를 감시 감독
- ~~서비스 프로그램~~

운영체제의 목적

- 처리 능력(Throughput) 향상 : 일정 시간 내에 시스템이 처리하는 일의 양
- 반환 시간(Turn Around Time) 단축 : 시스템에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
- 신뢰도(Reliability) : 시스템이 주어진 문제를 정확하게 해결하는 정도
- 가용성(Availability) : 시스템을 사용할 필요가 있을 때 즉시 사용 가능한 정도
- ~~Availability 감소 (가용성)~~

운영체제의 주요 자원 관리

- 프로세스 관리 : 프로세스 스케줄링 및 동기화 관리 담당
- 기억장치 관리 : 프로세스에게 메모리 할당 및 회수 관리 담당

- 주변장치 관리 : 입출력장치 스케줄링 및 전반적인 관리 담당
- 파일 관리 : 파일의 생성과 삭제, 변경, 유지 등의 관리 담당

운영체제의 종류 : Windows, UNIX, LINUX, MacOS, MS-DOS

7 UNIX

- 하나 이상의 작업에 대하여 백그라운드에서 수행 가능하다.
- **계층 구조(트리 구조)**의 파일 시스템을 갖는다.
- 이식성이 높으며 장치 간의 호환성이 높다.
- 다중 사용자(Multi-User), 다중 작업(Multi-tasking)을 지원한다.
- 시분할 시스템(Time Sharing System)을 위해 설계된 대화식 운영체제

→ 하드웨어 > **커널(Kernel)** > **쉘(Shell)** > 유틸리티(Utility) > 사용자(User)

커널(Kernel)

- 프로세스, 기억장치, 입출력 관리를 수행한다.
- 프로세스 관리, 파일관리, 입출력 관리, 기억장치 관리 등의 기능을 수행한다.
- UNIX 의 가장 핵심적인 부분
- 컴퓨터가 부팅될 때 주기억장치에 적재된 후 상주하면서 실행됨
- 하드웨어를 보호하고, 프로그램과 하드웨어 간의 인터페이스 역할을 담당

운영체제에서 커널의 기능

- 프로세스 생성, 종료
- 기억 장치 할당, 회수
- 파일 시스템 관리
- ~~사용자 인터페이스 => 쉘~~

운영체제의 커널(Kemel)을 찾아 메모리에 적재하는 과정 : Bootstrapping

쉘(Shell)

- **명령어 해석기이다.**
- **시스템과 사용자 간의 인터페이스를 담당한다.**
- **여러 종류의 쉘이 있다.**
- 사용자의 명령어를 인식하여 프로그램을 호출한다.
- 주기억장치에 상주하지 않고, 명령어가 포함된 파일 형태로 존재하며 보조 기억장치에서 교체 처리가 가능
- 파이프라인 기능 지원 및 입, 출력 재지정을 통해 입, 출력의 방향 변경 가능

***UNIX SHELL 환경 변수를 출력하는 명령어가 아닌 것은?**

- printenv
- env
- setenv
- ~~configenv~~

***bash 쉘 스크립트에서 사용할 수 있는 제어문이 아닌 것은?**

- if
- for
- ~~repeat_do~~
- while

***리눅스 Bash 쉘(Shell)에서 export 란?**

- export 가 매개변수 없이 쓰일 경우 현재 설정된 **환경변수**들이 출력된다.
- 사용자가 생성하는 변수는 export 명령어 표시하지 않는 한 현재 쉘에 국한된다.
- 변수를 export 시키면 **전역(Global)변수**처럼 되어 끝까지 기억된다.
- ~~변수를 출력하고자 할 때는 export 를 사용해야 한다. => echo~~

UNIX 명령어

- **fork** : UNIX 에서 새로운 프로세스를 생성하는 명령어
- **uname** : 운영체제 분석을 위해 리눅스에서 버전을 확인하고자 할 때 사용되는 명령어
- cat : 파일 내용 화면 표시, 커널 버전 확인
- chdir : 현재 사용할 디렉터리의 위치 변경
- chmod : 파일의 사용 허가 지정, 파일의 속성 변경
- chown : 소유자 변경, change own
- cp : 파일 복사, copy
- rm : 파일 삭제, remove
- exec : 새로운 프로세스 수행, execute
- find : 파일 찾기
- fsck : 파일 시스템 검사 및 보수, filesystem check
- ls : 현재 디렉터리 내의 파일 목록 확인, list = DIR (Windows 명령어)
- mount/unmount : 파일 시스템 마운팅/마운팅 해제

UNIX 에서의 프로세스 간 통신

- 각 프로세스는 시스템 호출을 통해 커널의 기능을 사용하며, 프로세스 간 통신은 시그널(Signal), 파이프(Pipe), 소켓(Socket)을 사용한다.
- **시그널**(Signal) : 간단한 메시지를 이용하여 통신하는 것, 초기 UNIX 시스템에서 사용
- **파이프**(Pipe) : 한 프로세스의 출력이 다른 프로세스의 입력으로 사용되는 단방향 통신 방식
- **소켓**(Socket) : 프로세스 사이의 대화를 가능하게 하는 쌍방향 통신 방식

소켓 기술

- 통신을 위한 프로그램을 생성하여 포트를 할당하고, 클라이언트의 통신 요청 시 클라이언트와 연결하는 내·외부 송·수신 연계기술

i -node

- UNIX 에서 각 파일에 대한 정보를 기억하고 있는 자료구조로서 파일 소유자의 식별번호, 파일 크기, 파일의 최종 수정시간, 파일 링크 수 등의 내용을 가지고 있는 것

8 프로세스, PCB

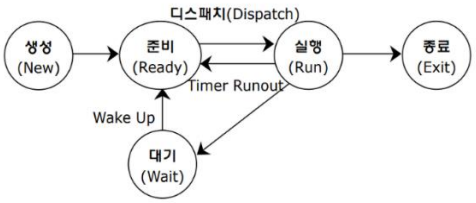
프로세스의 정의

- 프로시저가 활동중인 것
- 비동기적 행위를 일으키는 주체
- 운영체제가 관리하는 실행 단위
- 실행중인 프로그램
- PCB(Process Control Block)을 가진 프로그램
- 실기억장치에 저장된 프로그램
- 프로세서가 할당되는 실체로서, 디스패치가 가능한 단위
- 프로그램 카운터, 레지스터 같은 현재 사용되는 자원에 대한 정보를 가짐
- 실행을 위한 메모리 영역, 프로세서 제어 블록 등의 지원을 할당받은 상태의 프로그램

프로세스 상태 종류

- 보류 (pending)
- 준비 (**Ready**)
- 실행 (**Running**)
- 대기 (blocked)
- 교착 (deadlock)
- 완료 (**Exit**, terminated)
- ~~Request~~

프로세스 상태 전이



1) 디스패치(Dispatch) : 준비 → 실행
- 프로세스가 준비 상태에서 프로세서가 배당되어 실행 상태로 변화하는 것이다.

2) 할당시간초과(Time Run Out) : 실행 -> 준비

3) 대기(Block) : 실행 -> 대기

4) Wake Up : 대기 -> 준비

문맥 교환(Context Switching)

- 이전 프로세스의 상태 레지스터 내용을 보관하고 다른 프로세스의 레지스터를 적재하는 과정이다.
- 현재 CPU 를 사용하여 실행되고 있는 프로세스의 상태 정보를 저장하고 제어 권한을 ISR(Interrupt Service Routine)에게 넘기는 작업
- 하나의 프로세스에서 다른 프로세스로 CPU 가 할당되는 과정에서 발생된다.

스풀링(Spooling)

- 나중에 한꺼번에 입출력하기 위해 디스크에 저장하는 과정

PCB(Process Control Block, 프로세스 제어 블록)

- 프로세스 식별자, 프로세스 상태 등의 정보로 구성된다.
- 프로세스에 대한 정보를 저장해 놓은곳
- 운영체제가 그 프로세스를 관리하는데 필요한 모든 정보를 유지하는 자료구조 테이블

PCB 가 갖고 있는 정보

- 프로그램 카운터 : 실행될 명령어의 주소를 가지고 있는 레지스터
- CPU 레지스터 정보 : 누산기, 인덱스 레지스터, 범용 레지스터 등에 대한 정보
- 입출력 상태 정보 : 입출력장치, 개방된 파일 목록
- 메모리장치 관리 정보 : 기준 레지스터, 페이지 테이블에 대한 정보
- 포인터 : 프로세스가 위치한 메모리 및 할당된 자원에 대한 포인터
- 계정 정보 : CPU 사용 시간, 실제 사용 시간, 한정된 시간
- 프로세스의 현재 상태 : 준비, 대기, 실행 등의 프로세스 상태
- 프로세스의 고유 식별자
- 스케줄링 및 프로세스의 우선순위 : 스케줄링 정보 및 프로세스가 실행될 우선 순위
- ~~할당되지 않은 주변장치의 상태 정보~~

스레드(Thread)

- 한 개의 프로세스는 여러 개의 스레드를 가질 수 **있다**.
- 커널 스레드의 경우 운영체제에 의해 스레드를 운용한다.
- 사용자 스레드의 경우 사용자가 만든 라이브러리를 사용하여 스레드를 운용한다.
- 스레드를 사용함으로써 하드웨어, 운영체제의 성능과 응용 프로그램의 처리율을 향상시킬 수 있다.
- 하드웨어, 운영체제의 성능과 응용프로그램의 처리율을 향상시킬 수 있다.
- 스레드는 그들이 속한 프로세스의 자원과 메모리를 공유한다.
- 다중 프로세스 구조에서 각 스레드는 다른 프로세스에서 병렬로 실행될 수 있다.
- 스레드는 동일 프로세스 환경에서 서로 다른 독립적인 **다중 수행이 가능**하다.
- 프로세스의 실행단위
- 프로세스 내에서의 작업 단위로서 시스템의 여러 자원을 할당받아 실행하는 단위
- 프로세스의 일부 특성을 갖고 있기 때문에 경량 프로세스라고도 한다.
- 커널 스레드 : 운영체제 커널에 의해 스레드 운용, 구현이 쉬우나 속도 느림
- 사용자 스레드 : 사용자가 만든 라이브러리를 사용해 스레드 운용, 속도가 빠르나 구현 어렵다.

9 교착상태

교착상태가 발생할 수 있는 조건

- **상호 배제(Mutual exclusion)**
- **점유와 대기(Hold and wait)**
  - 프로세스가 수행되기 전에 필요한 모든 자원을 할당시켜 준다.
  - 자원이 점유되지 않은 상태에서만 자원을 요구하도록 한다.
- **비선점(Non-preemption)**
- **환형 대기(Circular wait)**
- ~~Linear wait~~

교착상태의 해결 기법

1) Detection(탐지)

- 교착상태 발생을 허용하고 발생 시 원인을 규명하여 해결
- ex. 자원할당 그래프

2) Recovery(복구)

- 교착상태 발견 후 현황대기를 배제시키거나 자원을 중단하는 메모리 할당 기법
- ex. 선점, 프로세스 중지(희생자 선택)

3) Avoidance(회피)

- 교착상태 가능성을 배제하지 않고 적절하게 피해나가는 방법
- ex. 은행원 알고리즘(Banker's Algorithm)

4) Prevention(예방)

- 교착상태의 필요조건(4 개 조건)을 부정함으로써 교착상태가 발생하지 않도록 미리 예방하는 방법
- 교착 상태의 원인이 되는 조건 중 하나를 제거
- 일반적으로 자원의 낭비가 가장 심함
- ex. 현황대기, 비선점, 점유와 대기, 상호배제 4 가지 조건의 부정

***Monitor**

- 병행프로세스의 문제점을 해결하기 위한 방안 중 **상호배제**의 한 형태인 동기화기법 중 하나.
- 세마포어, 모니터 중 하나

10 스케쥴링

1) 장기 스케줄링(작업 스케줄링, 상위 스케줄링)

- 어떤 프로세스가 시스템의 자원을 차지할 수 있도록 할 것인가를 결정하여 준비상태 큐로 보내는 작업
- 작업 스케줄러에 의해 수행됨

2) 중기 스케줄링

- 어떤 프로세스들이 CPU 를 할당받을 것인지 결정하는 작업

3) 단기 스케줄링(프로세서 스케줄링, 하위 스케줄링)

- 프로세스가 실행되기 위해 CPU 를 할당받는 시기와 특정 프로세스를 지정하는 작업
- 프로세서 스케줄링 및 문맥 교환은 프로세서 스케줄러에 의해 수행됨

스케줄링의 목적

- 공정성 : 모든 프로세스에 공정하게 할당
- 처리량 증가 : 단위 시간당 프로세스 처리량 증가
- CPU 이용률 증가 : CPU 낭비 시간 줄이고, 사용되는 시간 비율 증가
- 우선순위 제도 : 우선순위가 높은 프로세스 먼저 실행
- 오버헤드 최소화 : 오버헤드 최소화
- 응답시간(Response Time, 반응 시간) 최소화 : 작업 지시 및 반응 시작 시간 최소화
- 반환 시간(Turn Around Time) 최소화 : 제출한 시간부터 실행 완료 시간 최소화
- 대기 시간 최소화 : 준비상태 큐에서 대기하는 시간 최소화
- 균형 있는 자원의 사용 : 메모리, 입, 출력장치 등의 자원을 균형 있게 사용
- 무한 연기 회피 : 자원을 사용하기 위해 무한정 연기되는 상태 회피
  → CPU 이용률, 처리율, 반환 시간, 대기 시간, 응답 시간

프로세스 스케줄링의 기법

● **선점(Preemptive)** 스케줄링

- 하나의 프로세스가 CPU 를 할당받아 실행하고 있을 때 우선순위가 높은 다른 프로세스가 CPU 를 강제로 빼앗아 선점할 수 있는 기법
- 우선순위가 높은 프로세스 빠르게 처리 가능

- 빠른 응답 시간을 요구하는 **대화식 시분할 시스템**(Time Sharing System)에 사용됨
- **많은 오버헤드 발생**
- 선점이 가능하도록 일정 시간 배당에 대한 인터럽트용 타이머 클록 필요

 # RR, SRT, MLQ(Multi-Level Queue), MFQ

RR(Round-Robin)

- 시간 할당이 작아지면 프로세스-문맥 교환이 자주 일어난다.
- Time Sharing System 을 위해 고안된 방식이다.
- 시간 할당이 커지면 FCFS 스케줄링과 같은 효과를 얻을 수 있다.

SRT (Shortest Remaining Time)

- SRT 는 실행 시간을 추적해야 하므로, 오버헤드가 증가한다.
- ~~SRT 에서는 이미 할당된 CPU 를 다른 프로세스가 강제로 빼앗아 사용할 수 없다.~~

● 비선점(Non-Preemptive) 스케줄링

- 이미 할당된 CPU 를 다른 프로세스가 강제로 빼앗아 선점할 수 없는 기법
- CPU 를 할당 받으면 해당 프로세스가 완료될 때까지 CPU 사용
- 모든 프로세스에 대한 요구를 공정하게 처리 가능
- 프로세스 응답 시간의 예측 용이
- **일괄 처리 방식에 적합**
- 중요한 작업(짧은 작업)이 중요하지 않은 작업(긴 작업)을 기다리는 경우 발생 → **가뭄 현상**

 # 우선순위(Priority), 기한부(Deadline), FCFS(FIFO), SJF, HRN

SJF (Shortest Job First)

- 작업이 끝나기까지의 **실행시간** 추정치가 가장 작은 작업을 먼저 실행시킨다.

*다음과 같은 프로세스가 차례로 큐에 도착하였을 때, SJF(Shortest Job First) 정책을 사용할 경우 가장 먼저 처리되는 작업은? **P4**

프로세스 번호	실행시간
P1	6
P2	8
P3	4
P4	3

HRN (Highest Response-ratio Next)

- SJF 기법을 보완하기 위한 방식이다.
- 대기 시간이 긴 프로세스의 경우 우선 순위가 높아진다.
- 긴 작업과 짧은 작업 간의 지나친 불평등을 해소할 수 있다.
- 우선 순위를 계산하여 그 수치가 가장 **높은 것부터 낮은 순**으로 우선 순위가 부여된다.
- **HRN 우선순위 계산식 : (대기시간 + 서비스시간) / 서비스시간**

*HRN 방식으로 스케줄링 할 경우, 입력된 작업이 다음과 같을 때 처리되는 작업 순서 : **D→B→C→A**

작업	대기시간	서비스(실행)시간
A	5	20
B	40	20
C	15	45
D	20	2

(참고)

SSTF (Shortest Seek Time First)

*사용자가 요청한 디스크 입·출력 내용이 다음과 같은 순서로 큐에 들어 있을 때 SSTF 스케쥴링을 사용한 경우의 처리 순서는? (단, 현재 헤드 위치는 53 이고, 제일 안쪽이 1 번, 바깥쪽이 200 번 트랙이다.)

53-65-67-37-14-98-122-124-183

```
큐의 내용 : 98 183 37 122 14 124 65 67
```

현재 헤드위치에서 가장 가까운 것부터

11 기억장치 관리 전략

1) 반입(Fetch) 전략

- 보조기억장치에 보관중인 프로그램이나 데이터를 언제(When) 주기억장치로 적재할 것인지를 결정하는 전략
- 요구 반입(Demand Fetch) : 실행중인 프로그램이 특정 프로그램이나 데이터 등의 참조를 요구할 때 적재
- 예상 반입(Anticipatory Fetch) : 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재

2) 배치(Placement) 전략

- 새로 반입되는 프로그램이나 데이터를 주기억장치의 어디에(Where) 위치시킬 것인지를 결정하는 전략
- 최초 적합(First Fit): 빈 영역 중에서 첫 번째 분할 영역에 배치
- 최적 적합(Best Fit): 빈 영역 중에서 단편화를 가정 작게 남기는 분할 영역에 배치
- 최악 적합(Worst Fit): 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치

*기억공간이 15K, 23K, 22K, 21K 순으로 빈 공간이 있을 때 기억장치 배치 전력으로 "First Fit"을 사용하여 17K 의 프로그램을 적재할 경우 내부단편화의 크기는 얼마인가? 23k – 17k = **6K**

*빈 기억공간의 크기가 20K, 16K, 8K, 40K 일 때 기억장치 배치 전략으로 "Worst Fit"을 사용하여 17K 의 프로그램을 적재할 경우 내부 단편화의 크기는? 40k – 17k = **23K**

*메모리 관리 기법 중 Worst fit 방법을 사용할 경우 10K 크기의 프로그램 실행을 위해서는 어느 부분에 할당되는가? **NO.5**

영역번호	메모리크기	사용여부
NO.1	8K	FREE
NO.2	12K	FREE
NO.3	10K	IN USE
NO.4	20K	IN USE
NO.5	16K	FREE

3) 교체(Replacement) 전략

- 이미 사용되고 있는 영역 중에서 어느(Who) 영역을 교체할지 결정하는 전략
- FIFO, LRU, LFU, NUR, OPT, SCR

페이지 교체 알고리즘

● **FIFO(First In First Out) = FCFS(First Come First Serve)**

- 각 페이지가 주기억장치에 적재될 때마다 그때의 시간을 기억시켜 가장 먼저 들어와서 가장 오래 있었던 페이지를 교체하는 기법

*페이지 참조 열(Page reference string)에 대해 페이지 교체 기법으로 선입선출 알고리즘을 사용할 경우 페이지 부재(Page Fault) 횟수는? (단, 할당된 페이지 프레임 수는 3 이고, 처음에는 모든 프레임이 비어 있다.) **14**

<페이지 참조 열>

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0

*4 개의 프레임을 수용할 수 있는 주 기억장치가있으며, 초기에는 모두 비어 있다고 가정한다. 다음의 순서로 페이지 참조가 발생할 때, FIFO 페이지 교체 알고리즘을 사용할 경우 페이지 결함의 발생 횟수는? **6 회**

페이지 참조 순서 : 1, 2, 3, 1, 2, 4, 5, 1, 4

*다음과 같은 3 개의 작업에 대하여 FCFS 알고리즘을 사용할 때, 임의의 작업 순서로 얻을 수 있는 최대 평균 반환 시간을 T, 최소 평균 반환 시간을 t 라고 가정했을 경우 T-t 의 값은? **6**

프로세스	실행시간
P1	9
P2	3
P3	12

● **LRU(Least Recently Used)**

- 최근에 가장 오랫동안 사용하지 않은 페이지를 교체하는 기법
- 가장 오래 전에 사용된 페이지 교체

● **LFU**(Least Frequently Used)

- 사용 빈도가 가장 적은 페이지를 교체하는 기법

● **OPT(OPTimal** replacement, 최적 교체)

- 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체하는 기법
- 벨레이디(Belady)가 제안한 것으로, 페이지 부재 횟수가 가장 적게 발생하는 가장 효율적인 알고리즘

● **NUR**(Not Used Recently)

- LRU 와 비슷한 알고리즘으로, 최근에 사용하지 않은 페이지를 교체하는 기법
- 각 페이지마다 두 개의 비트, 즉 참조 비트와 변형 비트 사용

참조 비트	0	0	1	1
변형 비트	0	1	0	1
교체 순서	1	2	3	4

● **SCR**(Second Chance Replacement, 2 차 기회 교체)

- 가장 오랫동안 주기억장치에 있던 페이지 중 자주 사용되는 페이지의 교체를 방지하기 위한 것이다.
- FIFO 기법의 단점을 보완하는 기법

주기억장치 할당(Allocation)

- 프로그램이나 데이터를 실행시키기 위해 주기억장치에 어떻게(How) 할당할지 정함
- **연속 할당 기법** : 프로그램을 주기억장치에 연속으로 할당하는 기법
    - 단일 분할 할당 기법 : 오버레이, 스와핑
    - 다중 분할 할당 기법 : 고정(정적) 분할 할당 기법, 가변(동적) 분할 할당 기법
- **분산 할당 기법** : 프로그램을 특정 단위의 조각으로 나누어 할당하는 기법
    - 페이징(Paging) 기법 / 세그먼테이션(Segmentation) 기법

가상기억장치의 개요
- 보조기억장치(하드디스크)의 일부를 주기억장치처럼 사용하는 것으로, 용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용하는 기법
- 주기억장치의 용량보다 큰 프로그램을 실행하기 위해 사용
- 주기억장치의 이용률과 다중 프로그래밍 효율을 높일 수 있음
- 가상기억장치에 저장된 프로그램을 실행하려면 가상기억장치의 주소를 주기억장치의 주소로 바꾸는 주소 변환 작업 필요
- 블록 단위로 나누어 사용하므로 연속 할당 방식의 단편화 해결 가능

단편화(Fragmentation)
- 분할된 주기억장치에 프로그램을 할당하고 반납하는 과정을 반복하면서 사용되지 않고 남는 기억장치의 빈 공간 조각을 의미. 내부단편화와 외부단편화가 있음

12 페이징, 세그먼테이션

페이징(Paging) 기법
- 가상기억장치에 보관되어 있는 프로그램과 주기억장치의 영역을 **동일한** 크기로 나눈 후 나눠진 프로그램(페이지)을 동일하게 나눠진 주기억장치의 영역(페이지 프레임)에 적재시켜 실행하는 기법
- 페이지(page) : 일정한 크기로 나눈 단위
- 페이지 프레임(Page Frame) : 페이지 크기로 일정하게 나누어진 주기억장치의 단위
- **외부 단편화는 발생하지 않으나, 내부 단편화 발생**
- 주소 변환을 위해 페이지의 위치 정보를 갖고 있는 페이지 맵 테이블(Page Map Table) 필요
  → 페이지 맵 테이블 사용으로 비용 증가, 처리 속도 감소

페이지 크기가 작을 경우
- 기억장소 이용 효율이 증가한다.
- 입·출력 시간이 늘어난다.
- 내부 단편화가 감소한다.
- ~~페이지 맵 테이블의 크기가 감소한다.~~
- 페이지 단편화가 감소되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 줄어듬
- 효율적인 워킹 셋 유지 가능
- 페이지 정보를 갖는 페이지 맵(사상) 테이블의 크기가 커지고, 매핑 속도가 늦어짐

페이지 크기가 클 경우
- 페이지 단편화가 증가되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 늘어남
- 불필요한 내용까지도 주기억장치에 적재될 수 있음
- 페이지 정보를 갖는 페이지 맵(사상) 테이블의 크기가 작아지고, 매핑 속도가 빨라짐

- 디스크 접근 횟수가 줄어들어 전체적인 입, 출력 효율성이 증가됨
- 주기억 장치 공간 절약
- 참조되는 정보와 무관한 양의 정보가 주기억 장치에 남게 됨
- 테이블이 복잡하지 않아 관리 용이

세그먼테이션(Segmentation) 기법

- 가상기억장치에 보관되어 있는 프로그램을 **가변적인** 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 기억공간을 절약하기 위해서 사용하는 실행시키는 방법
- 기억장치의 사용자 관점을 보존하는 기억장치 관리 기법
- 세그먼트(Segment) : 논리적인 크기로 나눈 단위로, 각 세그먼트는 고유한 이름과 크기를 가짐
- 세그먼트 맵 테이블(Segment Map Table) : 주소 변환을 위해서 세그먼트가 존재하는 위치 정보를 갖고 있음
- 세그먼트가 주기억장치에 적재될 때 다른 세그먼트에게 할당된 영역을 침범할 수 없으며, 이를 위해 기억장치 보호키(Storage Protection Key)가 필요
- **내부 단편화는 발생하지 않으나, 외부 단편화 발생**

*다음과 같은 세그먼트 테이블을 가지는 시스템에서 논리 주소(2, 176)에 대한 물리 주소는? **398**

세그먼트번호	시작 주소	길이(바이트)
0	670	248
1	1752	422
2	222	198
3	996	604

- 논리주소 = (세그먼트번호, 변위값)
- 물리주소 = (세그먼트 시작주소 + 변위값)
- 즉 논리주소 (2,176)를 이용하여 물리주소를 구하면 (222+176) = 398

워킹 셋(Working Set)

- 운영체제의 가상기억장치 관리에서 프로세스가 일정 시간동안 자주 참조하는 페이지들의 집합을 의미한다.
- 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합

페이지 부재 빈도(PFF; Page Fault Frequency) 방식

- 페이지 부재율에 따라 주기억장치에 있는 페이지 프레임의 수를 늘리거나 줄여 페이지 부재율을 적정 수준으로 유지하는 방식
- 페이지 부재(Page Fault)는 프로세스 실행 시 참조할 페이지가 주기억장치에 없는 현상이며, 페이지 부재 빈도는 페이지 부재가 일어나는 횟수를 의미함

프리페이징(Prepaging)

- 처음의 과도한 페이지 부재를 방지하기 위해 필요할 것 같은 모든 페이지를 미리 한꺼번에 페이지 프레임에 적재하는 기법
- 기억장치에 들어온 페이지들 중에서 사용되지 않는 페이지가 많을 수도 있음

스레싱(Thrashing)

- 프로세스의 처리 시간보다 **페이지 교체**에 소요되는 시간이 더 많아지는 현상

14 시간지역성, 공간 지역성

지역성

- 프로세서들은 기억장치 내의 정보를 균일하게 접근하는 것이 아니라. 어느 한 순간에 특정부분을 집중적으로 참조한다.

시간 지역성(Temporal Locality)

- 하나의 기억장소가 가까운 장래에도 참조될 가능성이 높다.
- 시간 지역성의 예로 순환, 부프로그램, 스택 등이 있다.

 # Loop(루프), Stack(스택), Subroutine(서브루틴), Counting(카운팅), Totaling(집계)

공간 지역성(Spatial Locality)

- 프로세스가 어떤 페이지를 참조했다면 이후 그 페이지와 인접한 페이지들을 참조할 가능성이 높다.
- 어느 하나의 페이지를 참조하면 그 근처의 페이지를 계속 참조할 가능성이 높음
- 프로세서 실행시 일정 위치의 페이지를 집중적으로 액세스 한다.
- 공간 지역성의 대표적인 예로 순차적 코드의 실행이 있다.

 # Array(배열), Sequential Code(순차적 코드)

프로세스 적재 정책과 관련한 설명

- 반복, 스택, 부프로그램은 **시간 지역성(Temporal Locality)**과 관련이 있다.
- **공간 지역성(Spatial Locality)**은 프로세스가 어떤 페이지를 참조했다면 이후 가상주소공간상 그 페이지와 인접한 페이지들을 참조할 가능성이 높음을 의미한다.
- 스레싱(Thrashing) 현상을 방지하기 위해서는 각 프로세스가 필요로 하는 프레임을 제공할 수 있어야 한다.
- ~~일반적으로 페이지 교환에 보내는 시간보다 프로세스 수행에 보내는 시간이 더 크면 스레싱(Thrashing)이 발생한다.~~

15 모듈

공통 모듈

1) **정확성(Correctness) :** 시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성
2) **명확성(Clarity) :** 해당 기능에 대해 일관되게 이해되고, 한 가지로 해석될 수 있도록 작성★
3) **완전성(Completeness) :** 시스템 구현을 위해 필요한 모든 것을 기술
4) **일관성(Consistency) :** 공통 기능들 간 상호 충돌이 발생하지 않도록 작성
5) **추적성(Traceability) :** 기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성

***공통모듈의 재사용 범위에 따른 분류가 아닌 것은?**

- 컴포넌트 재사용, 함수와 객체 재사용, 애플리케이션 재사용, ~~더미코드 재사용~~

모듈화(Modularity)

- 소프트웨어의 모듈은 프로그래밍 언어에서 Subroutine, Function 등으로 표현될 수 있다.
- 모듈화는 시스템을 지능적으로 관리할 수 있도록 해주며, 복잡도 문제를 해결하는 데 도움을 준다.
- 모듈화는 시스템의 유지보수와 수정을 용이하게 한다.
- **모듈의 수가 증가하면, 각 모듈의 크기가 작아진다. => 모듈간 통합 비용 적음, 모듈 하나의 개발비용 큼**
- **모듈의 수가 감소하면, 각 모듈의 크기가 커진다. => 모듈간 통합 비용 큼**

효과적인 모듈 설계

- 모듈간의 결합도를 약하게 하면 모듈 독립성이 향상된다.
- 복잡도와 중복성을 줄이고 일관성을 유지시킨다.
- 유지보수가 용이해야 한다.
- ~~모듈의 기능은 예측이 가능해야 하며 지나치게 제한적 이여야 한다.~~

16 라이브러리

- 라이브러리란 필요할 때 찾아서 쓸 수 있도록 모듈화되어 제공되는 프로그램을 말한다.
- 프로그래밍 언어에 따라 일반적으로 도움말, 설치 파일, 샘플 코드 등을 제공한다.
- 라이브러리는 모듈과 패키지를 총칭하며, 모듈이 개별 파일이라면 패키지는 파일들을 모아 놓은 폴더이다.
- 표준 라이브러리는 프로그래밍 언어가 기본적으로 가지고 있는 라이브러리를 의미한다.
- 외부 라이브러리는 별도의 파일 설치를 필요로 하는 라이브러리를 의미한다.

C 언어의 표준 라이브러리

- stdio.h : 표준 입출력 라이브러리
- math.h : 삼각 함수, 제곱근, 지수 등 수학적인 함수를 내장하고 있다.
- string.h : 문자열 처리 함수로, strlen()이 포함되어 있다.
- stdlib.h : 문자열을 수치 데이터로 바꾸는 문자 변환함수와 수치를 문자열로 바꿔주는 변환함수 등이 있다.
- time.h : 시간 처리에 사용되는 기능 제공

stdlib.h 라이브러리의 대표 함수

- atoi() : char to int : 문자열을 정수형으로 변한한다
- atof() : char to double : 문자열을 부동 소수점으로 변한한다
- itoa() : int to char : 정수형을 문자열로 변한한다
- ceil() : 소수점값이 나올 때 무조건 올림

JAVA 의 대표적인 표준 라이브러리

- java.lang : 기본적으로 필요한 인터페이스, 자료형, 예외 처리 등의 기능 제공. import 문 없이도 사용 가능
- java.util : 날짜 처리, 난수 발생, 복잡한 문자열 처리 등에 관련된 기능 제공
- java.io : 파일 입, 출력과 관련된 기능 및 프로토콜 제공
- java.net : 네트워크와 관련된 기능 제공
- java.awt : 사용자 인터페이스(UI)와 관련된 기능 제공

17 데이터 타입

*Java 프로그래밍 언어의 정수 데이터 타입 중 'long'의 크기는? **8byte**

C/C++의 데이터 타입 크기

~~- 문자 : char~~
~~- 정수 : short, int (4byte), long ()~~
~~- 실수: float, double, long double~~

18 변수 선언

- 변수는 어떤 값을 주기억 장치에 기억하기 위해서 사용하는 공간이다.
- 변수의 자료형에 따라 저장할 수 있는 값의 종류와 범위가 달라진다.
- char 자료형은 나열된 **하나의** 문자를 저장하고자 할 때 사용한다.
- boolean 자료형은 조건이 참인지 거짓인지 판단하고자 할 때 사용한다.

변수 작성 규칙

- 첫 자리에 숫자를 사용할 수 없다.
- 영문 대문자/소문자, 숫자, 밑줄(_)의 사용이 가능하다.
- 변수 이름의 중간에 공백을 사용할 수 **없다**.
- 이미 사용되고 있는 예약어는 사용할 수 없다.

*C 언어에서의 변수로 사용할 수 없는 것들

- else; (X) 예약어
- short (X) 예약어
- text-color (X) 특수기호 -

19 연산자

산술 연산자

+	덧셈
-	뺄셈
*	곱하기
**	제곱
/	나누기
//	나누기 연산 후 몫
%	나누기 연산 후 나머지
++	증감 연산자
--	감소 연산자

시프트 연산자

<<	왼쪽 시프트 (비트를 왼쪽으로 이동) ex. 00101 → 01010
>>	오른쪽 시프트 (비트를 오른쪽으로 이동) ex. 00101 → 00010

관계 연산자

==	같다
!=	같지 않다
>	크다
>=	크거나 같다
<	작다
<=	작거나 같다

비트 연산자

& and (모든 비트가 1 일 때만 1)
| or (모든 비트 중 한 비트라도 1 이면 1)
^ xor (모든 비트가 같으면 0, 하나라도 다르면 1)
~ not (각 비트의 부정, 0 이면 1, 1 이면 0)
*C 언어에서 비트 논리연산자에 해당하지 않는 것은? ^, ?, &, ~ (?는 조건 연산자)

논리 연산자
&& and (모두 참(1)이면 참(1))
|| or (하나라도 참(1)이면 참(1))
! not (부정)

조건 연산자(삼항 연산자)
조건 ? 긍정 수식 : 부정 수식;

대입 연산자
+= a += 1
-= a -= 1
***=** a *= 1
/= a /= 1
%= a %= 1
<<= a <<= 1
>>= a >>= 1

연산자 우선순위
단항 연산자 ! ~ ++ -- sizeof
산술 연산자 * / %
산술 연산자 + -
시프트 연산자 << >>
관계 연산자 < <= >= >
관계 연산자 == !=
비트 연산자 & ^ |
논리 연산자 && ||
조건 연산자 ?:
대입 연산자 = += -= *= /= %= <<= >>=
순서 연산자 ,

*다음중 JAVA 에서 우선순위가 가장 낮은 연산자는? -- % & =

*C 언어에서 연산자 우선순위가 높은 것에서 낮은 것은? ( ) / << < == ||

20 포인터

포인터, 포인터 변수
- 포인터는 **변수의 주소**를 말하며, C 언어에서는 주소를 제어할 수 있는 기능을 제공함

- 포인터 변수는 변수의 주소를 저장할 때 사용하는 변수이다.
- 포인터 변수는 필요에 의해 동적으로 할당되는 메모리 영역인 힙 영역에 접근하는 동적 변수다.
- 포인터 변수를 선언할 때는 자료의 형을 먼저 쓰고 변수명 앞에 간접 연산자 *를 붙임
  → int *a;
- 포인터 변수에 주소를 저장하기 위해 변수의 주소를 알아낼 때는 변수 앞에 번지 연산자 &를 붙임
  → a = &b;
- 실행문에서 포인터 변수에 간접 연산자 *를 붙이면 해당 포인터 변수가 가리키는 곳의 값을 말함
  → c = *a;

포인터, 배열

- 배열을 포인터 변수에 저장한 후 포인터를 이용해 배열의 요소에 접근할 수 있음
- 배열 요소에 대한 주소를 지정할 때는 일반 변수와 동일하게 & 연산자를 사용
- ex. int a[5], *b;
- b = a → 배열의 대표명을 적었으므로 a 배열의 시작 주소인 a[0]의 주소를 b 에 저장함
- b = &a[0] → a 배열의 첫 번째 요소인 a[0]의 주소(&)를 b 에 저장함

21 각종 문제 풀이

*다음 자바 프로그램 조건문에 대해 삼항 조건 연산자로 표현한 것은? **k = (i > j)?(i – j):(i + j);**

```
Int I = 7, j = 9;
int k;
if (i > j)
   k = i – j;
else
   k = i + j;
```

*다음 C 프로그램의 결과 값은? **25**

```
main(void) {
int i;
int sum = 0;
for(i =1; i<=10; i=i+2)
   sum = sum + i;
printf("%d", sum);
}
```

*다음은 사용자로부터 입력받은 문자열에서 처음과 끝의 3 글자를 추출한 후 합쳐서 출력하는 파이썬 코드에서 ㉠에 들어갈 내용은? **string[0:3] + string[-3:]**

```
String = input("7문자 이상 문자열을 입력하시오 :")
m = (    ㉠    )
print(m)
```

```
입력값 : Hello World
최종 출력 : HelrId
```

- [:] 처음부터 끝까지
- [start:] start 오프셋부터 끝까지
- [:end] 처음부터 end-1 오프셋까지
- [start : end] start 오프셋부터 end-1 오프셋까지

- [start : end : step] step 만큼 문자를 건너뛰면서, 위와 동일하게 추출

*다음 자바 코드를 실행한 결과는? **Unresolved compilation problem 오류 발생**

```
int x = 1, y = 6;
while (y--) {
    x++;
}
System.out.println("x = " + x + "y = " +y);
```

while 문의 조건식은 boolean 이어야 한다.

*다음 파이썬으로 구현된 프로그램의 실행 결과로 옳은 것은? **[0, 20, 40, 60]**

```
>>> a = [0,10,20,30,40,50,60,70,80,90]
>>> a[ : 7 : 2]
```

a[:7:2] = a[0]~a[6]라서 [0,10,20,30,40,50,60]에서 2 칸씩 띄운다.

*파이썬(Python) 프로그램이 실행되었을 때의 결과는? **66 (0~11 까지의 숫자의 합)**

```
def cs(n):
    s = 0
    for num in range(n+1):
        s += num
    return s

print(cs(11))
```

*C 언어 프로그램이 실행되었을 때의 결과는? **nationalter**

```
#include <stdio.h>
#include <string.h>
int main(void) {
        char str[50] = "nation";
        char *p2 = "alter";
        strcat(str, p2);
        printf("%s", str);
    return 0;
}
```

*C 언어 프로그램이 실행되었을 때의 결과는? 참(1) + 참(1) + 거짓(0) = **2**

```
#include <stdio.h>
int main(void) {
 int a = 3, b = 4, c = 2;
 int r1, r2, r3;

 r1 = b <= 4 || c == 2;
 r2 = (a > 0) && (b < 5);
 r3 = !c;

 printf("%d", r1+r2+r3);
 return 0;
}
```

*C 언어 프로그램이 실행되었을 때의 결과는? **8**

```
#include <stdio.h>
int main(void) {
 int n = 4;
 int* pt = NULL;
 pt=&n;

 printf("%d", &n + *pt - *&pt + n);
 return 0;
}
```

- **&n** = 변수 n 의 주소값
- ***pt** = 포인터 pt 가 가리키고 있는 주소에 저장된 값 = 변수 n
- ***&pt** = *(포인터 pt 의 주소값)= 포인터 pt 의 주소가 가리키고 있는 주소에 저장된 값 = 변수 n 의 주소
- &n + *pt - *&pt + n

= *pt + n = 4 + 4 = 8

* JAVA 프로그램이 실행되었을 때의 결과를 쓰시오. **34**

```
public class ovr {
 public static void main(String [] arge) {
  int arr[];
  int i = 0;
  arr = new int[10];
  arr[0] = 0;
  arr[1] = 1;
   while(i<8) {
       arr[i+2] = arr[i+1] + arr[i];
       i++;
   }
  System.out.println(arr[9]);
  }
}
```

*다음 C 언어 프로그램이 실행되었을 때의 결과는? 00000100 | 00000111 = 00000111 = **7**

```
#include <stdio.h>
int main(int argc, char *argv[]) {
         int a = 4;
         int b = 7;
         int c = a | b;

         printf("%d", c);
          return 0;
}
```

참고로 & 연산자일 경우 00000100 = 4 가 된다.

*다음 파이썬(Python) 프로그램이 실행되었을 때의 결과는? **4+2 = 6**

```
class FourCal:
    def setdata(sel, fir, sec):
        sel.fir = fir
        sel.sec = sec
    def add(sel):
        result = sel.fir + sel.sec
        return result
a = FourCal()
a.setdata(4, 2)
print(a.add())
```

*다음 JAVA 프로그램이 실행되었을 때의 결과는? **5, 5, 5**

```
public class Operator{
public static void main(String[] args) {
  int x=5, y=0, z=0;
  y = x++;
  z = --x;
System.out.print(x +"," + y + ","+ z);
  }
}
```

*다음 JAVA 프로그램이 실행되었을 때의 결과는? **2**

```
public class array1 {
 public static void main(String[] args) {
        int cnt = 0;
        do {
                cnt++;
        } while (cnt < 0);
        if(cnt==1)
                cnt++;
        else
                cnt = cnt + 3;
        System.out.printf("%d", cnt);
 }
}
```

*다음 C 언어 프로그램이 실행되었을 때의 결과는? **66**

```
#include <stdio.h>
int main(int argc, char *argv[]) {
        char a;
        a = 'A' + 1;
        printf("%d", a);
        return 0;
}
```

아스키코드로 'A'는 65, 'a'는 97

*다음 C 언어 프로그램이 실행되었을 때의 결과는? **121**

```
#include <stdio.h>
int main(int argc, char *argv[]) {
        int a[2][2] = {{11, 22), (44, 55}};
        int i, sum = 0;
        int *p;
        p= a[0];
        for(i = 1; i < 4; i++)
        sum += *(p + i);
        printf("%d", sum);
        return 0;
}
```

22+44+55 = 121

*다음 JAVA 코드 출력문의 결과는? **5 + 2 = 34<chal>5 + 2 = 7**

```
..생략..
System.out.println("5 + 2 = " + 3 + 4);
System.out.println("5 + 2 = " + (3 + 4));
..생략..
```

*다음은 파이썬으로 만들어진 반복문 코드이다. 이 코드의 결과는? **A, B, C 출력이 반복된다.**

```
>> while(True) :
    print('A')
    print('B')
    print('C')
    continue
    print('D')
```

22 각종 언어 / 프레임워크

절차적 프로그래밍 언어

- C : 컴파일러 방식의 언어로 이식성이 좋아 컴퓨터 기종에 관계없이 프로그램 작성 가능
- Algol(알골) : 수치계산이나 논리 연산을 위한 과학 기술 계산용 언어

- **Cobol(코볼)** : 사무 처리용 언어로 영어 문장 형식으로 구성되어 있어 이해와 사용이 쉬움
- Fortran(포트란) : 과학 기술 계산용 언어로 수학/공학 분야의 공식이나 수식과 같은 형태로 프로그래밍 가능
- Basic(베이직) : 교육용으로 개발되어 언어의 문법이 쉬움

객체지향 프로그래밍 언어

- C++ : C 언어에 객체지향 개념을 적용한 언어
- C# : Microsoft 에서 개발. JAVA 와 달리 불안전 코드(Unsafe Code) 기술을 통해 플랫폼 간 상호 운용성 확보
- JAVA : 분산 네트워크 환경에 적용이 가능하며, 멀티스레드 기능을 제공하므로 여러 작업을 동시에 처리 가능
- Delphi(델파이) : 기본적인 문법은 파스칼 문법에 여러 기능들이 추가되어 존재(높은 생산성과 간결한 코드)
- Smalltalk : 1 세대 객체지향 프로그래밍 언어, 최초로 GUI 를 제공한 언어

스크립트 언어

- **JavaScript(자바스크립트)**
  - 웹페이지의 동작을 제어하는 데 사용되는 **클라이언트용 스크립트 언어**
  - 프로토타입(Prototype)의 개념이 존재한다.
  - Prototype Link 와 Prototype Object 를 활용할 수 있다.
  - 객체지향 언어이다.
  - ~~클래스 기반으로 객체 상속을 지원한다. => 객체 기반으로 클래스 상속을 지원한다~~
- **ASP(Active Server Page)**
  - 서버 측에서 동적으로 수행되는 페이지를 만들기 위한 언어, Microsoft 제작
  - Windows 계열에서만 수행 가능
- **JSP(Java Server Page)**
  - JAVA 로 만들어진 서버용 스크립트. 다양한 운영체제에서 사용 가능
- **PHP(Professional Hypertext Preprocessor)**
  - 서버용 스크립트 언어로 C, JAVA 등과 문법이 유사. LINUX, UNIX, Windows 운영체제에서 사용 가능

  *PHP 에서 사용 가능한 연산자가 아닌 것은?
  - **@ # <> === 중에서 # 이 정답**
  - **PHP 연산자**

    @ : 함수 사용시 발생되는 오류메시지를 표시하지 않음

    <> : 값이 서로 같지 않을 때

    = : 값을 지정할 때 사용

    == : 두 값이 같은지 확인하기

    === : 두 값이 같고, 형식도 같은지 확인하기

    :: : 사용하는 시점에 객체가 생성되고 지정된 method 가 실행

- **Python(파이썬)**
  - 귀도 반 로섬(Guido van Rossum)이 발표한 언어
  - 인터프리터 방식이자 객체지향적이며, 배우기 쉽고 이식성이 좋은 것이 특징인 스크립트 언어

***스크립트 언어의 종류가 아닌 것은?**

- **PHP, ~~Cobol~~, Basic, Python**

선언형 언어

- **Haskell(하스켈)** : 함수형 프로그래밍 언어. 패턴 맞춤, 커링, 조건제시법, 가드, 연산자 정의 등 기능 존재

- **LISP(리스프)** : 함수형 프로그래밍 언어. 수학 표기법을 나타내기 위한 목적
- **PROLOG(프롤로그)** : 인공지능이나 계산 언어학 분야, 자연언어 처리 분야에서 사용
- **HTML** : 인터넷의 표준 문서인 하이퍼텍스트 문서를 만들기 위해 사용하는 언어
- **XML**: HTML 의 단점을 보완해 웹에서 구조화된 폭 넓고 다양한 문서들을 상호 교환할 수 있도록 설계된 언어

프레임워크

- **Spring** : JAVA 기반으로 만들어진 프레임워크. 전자정부 표준 프레임워크의 기반 기술로 사용됨
- **Node.js** : JavaScript 기반으로 만들어진 프레임워크, 실시간으로 입출력이 빈번한 애플리케이션에 적합
- **Django** : Python 기반으로 만들어진 프레임워크
- **Codeigniter** : PHP 기반으로 만들어진 프레임워크
- **Ruby on Rails** : Ruby 기반으로 만들어진 프레임워크

프레임워크의 특성

- 모듈화(Modularity)
  - 프레임워크는 캡슐화를 통해 모듈화를 강화하고 설계 및 구현의 변경에 따른 영향을 최소화함으로써 소프트웨어의 품질을 향상시킴
- 재사용성(Reusability)
  - 프레임워크는 재사용 가능한 모듈들을 제공함으로써 개발자의 생산성을 향상시킴
- 확장성(Extensibility)
  - 프레임워크는 다형성을 통한 인터페이스 확장이 가능하여 다양한 형태와 기능을 가진 애플리케이션 개발이 가능함
- 제어의 역흐름(Inversion of Control)
  - 개발자가 관리하고 통제해야 하는 객체들의 제어를 프레임워크가 관리함으로써 생산성 향상시킴

24 각종 규약

***X 시리즈** - 공중 데이터 교환망(PSDN)을 통한 DTE/DCE 접속 규격

- **X.20** : 비동기식 전송을 위한 DTE/DCE 접속 규격
- **X.21** : 동기식 전송을 위한 DTE/DCE 접속 규격
- **X.25** : 패킷 전송을 위한 DTE/DCE 접속 규격

***EEE 802 의 표준 규약**

- IEEE 802.3 : **CSMA/CD**
- IEEE 802.4 **: Token BUS**
- IEEE 802.5 : **Token RING = 토큰링에 대한 표준**
- IEEE 802.8 : Fiber optic LANS
- IEEE 802.9 : 음성/데이터 통합 LAN
- IEEE 802.11 : **무선 LAN(CSMA/CA)**

***CSMA/CD**

- IEEE 802.3 LAN 에서 사용되는 전송매체 접속제어(MAC) 방식
- Carrier-Sense Multiple Access with Collision Detection (충돌 감지)

***CSMA/CA**

- 무선 랜에서 데이터 전송시, 매체가 비어 있음을 확인한 뒤 충돌을 회피하기 위해 임의 시간을 기다린 후 데이터를 전송하는 방법이다.
- 네트워크에 데이터의 전송이 없는 경우라도 동시 전송에 의한 충돌에 대비하여 확인 신호를 전송한다.
- Carrier-Sense Multiple Access with Collision Avoidance (충돌 방지)

***IEEE 802.11e**

- IEEE 802.11 워킹 그룹의 무선 LAN 표준화 현황 중 **QoS**(Quality of Service) 강화를 위해 MAC 지원 기능을 채택한 것

***Collision Domain(충돌 도메인)**

- 충돌 발생을 검출할 수 있는 브리지 간 혹은 다른 계층 장치 간의 이더넷 세그먼트 범위

25 ARQ

***오류 제어에 사용되는 자동반복 요청방식(ARQ, Automatic Repeat reQuest)**

1) Stop-and-wait ARQ (정지-대기 ARQ)

- 한 개의 프레임을 전송하고, 수신 측으로부터 ACK 및 NAK 신호를 수신할 때까지 정보전송을 중지하고 기다리는 ARQ(Automatic Repeat Request) 방식

2) Go-back-N ARQ

- 여러 블록을 연속적으로 전송하고, 수신 측에서 부정 응답(NAK)을 보내오면 송신 측이 오류가 발생한 블록부터 **모두 재전송**

3) Selective-Repeat ARQ (선택적 재전송 ARQ)

- 여러 블록을 연속적으로 전송하고, 수신측에서 부정 응답(NAK)을 보내오면 송신 측이 **오류가 발생한 블록만을 재전송**

4) Adaptive ARQ (적응적 ARQ)

- 전송 효율을 최대로 하기 위해서 데이터 블록의 길이를 채널의 상태에 따라 동적으로 변경하는 방식

~~- Non-Acknowledge ARQ~~

26 배치 프로그램

배치 프로그램의 필수 요소

- 자동화는 심각한 오류 상황 외에는 사용자의 개입 없이 동작해야 한다.
- 안정성은 어떤 문제가 생겼는지, 언제 발생했는지 등을 추적할 수 있어야 한다.
- 대용량 데이터는 대용량의 데이터를 처리할 수 있어야 한다.

~~- 무결성은 주어진 시간 내에 처리를 완료할 수 있어야 하고, 동시에 동작하고 있는 다른 애플리케이션을 방해하지 말아야 한다.~~

- 대용량 데이터 : 대량의 데이터를 가져오거나, 전달하거나, 계산 등의 처리가 가능해야 함
- 자동화 : 심각한 오류가 발생하는 상황을 제외하고는 사용자의 개입 없이 수행돼야 함
- 견고성 : 잘못된 데이터나 데이터 중복 등의 상황으로 중단되는 일 없이 수행돼야 함
- 안정성/신뢰성 : 오류가 발생하면 오류의 발생 위치, 시간 등을 추적할 수 있어야 함
- 성능 : 다른 응용 프로그램의 수행을 방해하지 않아야 하고, 지정된 시간 내에 처리가 완료돼야 함

배치 스케줄러, 잡 스케줄러

- 일괄 처리 작업이 설정된 주기에 맞춰 자동으로 수행되도록 지원해주는 도구

- **스프링 배치(Spring Batch)**
  - 스프링이 가진 다양한 기능들을 모두 사용할 수 있는 오픈 소스 프레임워크
  - 주요 구성 요소 : Job, Job Launcher, Job Repository, Step
- **쿼츠(Quartz)**
  - Spring 프레임워크로 개발되는 응용 프로그램들의 일괄 처리를 위한 다양한 기능을 제공하는 오픈 소스 라이브러리
  - 주요 구성 요소 : Job, Job Detail, Trigger, Scheduler

27 기타

***보안유지기법**

- 외부보안 : 열감지, 음성, 지문 등의방법으로 천재지변이나 외부 침입으로부터 보호 (시설보안)
- 운용보안 : 액세스 권리를 부여
- 사용자 인터페이스 보안 : 운영체제가 사용자의 신원을 확인
- 내부보안 : 하드웨어나 운영체제에 내장된 보안 기능을 이용

***Garbage Collector**

- JAVA 에서 힙(Heap)에 남아있으나 변수가 가지고 있던 참조값을 잃거나 변수 자체가 없어짐으로써 더 이상 사용되지 않는 객체를 제거해주는 역할을 하는 모듈

***버퍼 오버플로**

- 메모리를 다루는 데 오류가 발생하여 잘못된 동작을 하는 프로그램 취약점

***linker**

- 언어번역프로그램이 생성한 목적프로그램들과 라이브러리, 또 다른 실행프로그램 등을 연결하여 실행 가능한 모듈을 만드는 것

***파일 디스크립터(File Descriptor)**

- 파일 관리를 위해 시스템이 필요로 하는 정보를 가지고 있다.
- 보조기억장치에 저장되어 있다가 파일이 개방(open)되면 주기억장치로 이동된다.
- 파일 제어 블록(File Control Block)이라고도 한다.
- 사용자가 파일 디스크립터를 직접 참조할 수 **없다**.

***파일 디스크립터(File Descroptor)의 정보에 포함 되지 않은 것은?**

- 파일 구조
- 파일 유형
- 파일 크기
- ~~파일 오류 처리 방법~~
- ~~파일의 내용~~
- ~~파일 작성자~~

<5 과목 정보시스템 구축 관리>

1 소프트웨어 개발 방법론

구조적 개발 방법론

- **정형화된 분석 절차에 따라 사용자 요구사항을 파악, 문서화하는 체계적 분석방법으로 자료흐름도, 자료사전, 소단위명세서의 특징을 갖는 것**

정보공학 방법론

- 정보 시스템의 개발을 위해 계획, 분석, 설계, 구축에 정형화된 기법들을 상호 연관성 있게 통합 및 적용하는 자료(Data) 중심의 방법론 → 대규모 정보 시스템 구축 적합

컴포넌트 기반 방법론 (CBD, Component Based Development)

- 생산성과 품질을 높이고, 유지보수 비용을 최소화할 수 있다.
- 컴포넌트 제작 기법을 통해 재사용성을 향상시킨다.
- 독립적인 컴포넌트 단위의 관리로 복잡성을 최소화할 수 있다.
- 개발 기간 단축으로 인한 생산성 향상
- 새로운 기능 추가가 쉬운 확장성
- 소프트웨어 재사용이 가능
- ~~모듈의 분할과 정복에 의한 하향식 설계방식이다.~~
- ~~1960 년대까지 가장 많이 적용되었던 소프트웨어 개발 방법~~

CBD 방법론의 SW 개발 표준 산출물

1. **요구파악 단계** → 요구사항 기술서, 용어 사전, 개념 모델, 유즈케이스 모델
2. **분석** → 객체 모델, UI 설계서, 아키텍처 기술서, 인터페이스 명세서, 컴포넌트 명세서, 컴포넌트 설계서, 데이터베이스 설계서, 사용자 요구사항 정의서
3. **구현** → 개발 표준 정의서, 플랫폼 종속적 코드
4. **테스트** → 테스트 계획서, 컴포넌트 테스트 보고서, 통합 테스트 보고서, 인수테스트 보고서

***CBD(Component Based Development) SW 개발 표준 산출물 중 '분석' 단계에 해당하는 것은?**

- ~~클래스 설계서~~
- ~~통합시험 결과서~~
- ~~프로그램 코드~~
- 사용자 요구사항 정의서

2 비용 산정 기법

소프트웨어 비용 결정 요소

1) 프로젝트 요소

- 제품 복잡도 : 소프트웨어의 종류에 따라 발생할 수 있는 문제점들의 난이도를 의미함
- 시스템 크기 : 소프트웨어의 규모에 따라 개발해야 할 시스템의 크기를 의미함
- 요구되는 신뢰도 : 일정 기간 내 주어진 조건하에서 프로그램이 필요한 기능을 수행하는 정도를 의미함

2) 자원 요소

- 인적 자원 : 소프트웨어 개발 관련자들이 갖춘 능력 혹은 자질을 의미함
- 하드웨어 자원 : 소프트웨어 개발 시 필요한 장비와 워드프로세서, 프린터 등의 보조 장비를 의미함
- 소프트웨어 자원 : 소프트웨어 개발 시 필요한 언어 분석기, 문서화 도구 등의 개발 지원 도구를 의미함

3) 생산성 요소

- 개발자 능력 : 개발자들이 갖춘 전문지식, 경험, 이해도, 책임감, 창의력 등을 의미함
- 개발 기간 : 소프트웨어를 개발하는 기간을 의미함

[3] 하향식 비용 산정 기법

- 과거의 유사한 경험을 바탕으로 전문 지식이 많은 개발자들이 참여한 회의를 통해 비용을 산정하는 비과학적인 방법

1) 전문가 감정 기법

- 조직 내에 있는 경험이 많은 두 명 이상의 전문가에게 비용 산정을 의뢰하는 기법
- 새로운 프로젝트에는 과거의 프로젝트와 다른 요소들이 있다는 것을 간과할 수 있음
- 새로운 프로젝트와 유사한 프로젝트에 대한 경험이 없을 수 있음
- 개인적이고 주관적일 수 있음

2) 델파이 기법

- 산정 요원과 조정자에 의해 산정하는 방법
- 전문가 감정 기법의 주관적인 편견을 보완하기 위해 한 명의 조정자와 여러 전문가의 의견을 종합하여 산정하는 기법

[4] 상향식 비용 산정 기법

- 프로젝트의 세부적인 작업 단위별로 비용을 산정한 후 집계하여 전체 비용을 산정하는 방법

1) LOC 기법 (원시 코드 라인 수, source Line Of Code)

- S/W 각 기능의 원시 코드 라인수의 비관치, 낙관치, 기대치를 측정하여 예측치를 구하고 이를 이용하여 비용을 산정하는 기법

*LOC 기법에 의하여 예측된 총 라인수가 36,000 라인, 개발에 참여할 프로그래머가 6 명, 프로그래머들의 평균 생산성이 월간 300 라인일 때 개발에 소요되는 기간은? (36000 / 300) / 6 = **20 개월**

*LOC 기법에 의하여 예측된 총 라인 수가 50,000 라인, 프로그래머의 월 평균 생산성이 200 라인, 개발에 참여할 프로그래머가 10 인 일 때, 개발 소요 기간은? → (50,000 / 200) / 10 = **25 개월**

2) 개발 단계별 인월수(Effort Per Task) 기법

- LOC 기법을 보완하기 위한 기법
- 각 기능을 구현시키는 데 필요한 노력을 생명 주기의 각 단계별로 산정함, LOC 기법보다 더 정확함

3) COCOMO(Constructive Cost Model) 모형★

- 보헴(Boehm)이 제안한 것으로 원시코드 라인 수에 의한 비용 산정 기법이다.
- 비용견적의 유연성이 높아 소프트웨어 개발비 견적에 널리 통용되고 있다.
- 산정 결과는 프로젝트를 완성하는데 필요한 **man-month** 로 나타난다.
- ~~프로젝트 개발유형에 따라 object, dynamic, function 의 3 가지 모드로 구분한다.~~

COCOMO 의 소프트웨어 개발유형

- **조직형 Organic**
  - 기관 내부에서 개발된 중소규모의 소프트웨어로 일괄 자료 처리나 과학 기술 계산용, 비즈니스 자료 처리용으로 5 만(50KDSI) 라인 이하의 소프트웨어를 개발하는 유형
- **반분리형 Semi-Detached**
  - 트랜잭션 처리 시스템이나 운영체제, 데이터베이스 관리 시스템 등의 30 만(300KDSI) 라인 이하의 소프트웨어를 개발하는 유형
- **내장형 Embedded**
  - 최대형 규모의 트랜잭션 처리 시스템이나, 운영체제 등의 30 만(300KDSI) 라인 이상의 소프트웨어를 개발하는 유형

COCOMO 모형의 종류

- **기본형 COCOMO (Basic)**
  - 소프트웨어의 크기(생산 코드 라인 수)와 개발 유형만을 이용하여 비용을 산정하는 모형
- **중간형 COCOMO (Intermediate)**
  - 기본형 COCOMO 의 공식을 토대로 사용하나, 제품, 컴퓨터, 개발요원, 프로젝트 특성의 15 가지 요인에 의해 비용을 산정하는 모형
- **발전형 COCOMO (Detailed)**
  - 중간형 COCOMO 를 보완하여 만들어진 방법으로, 개발 공정별로 보다 자세하고 정확하게 노력을 산출하여 비용을 산정하는 모형

  → 소프트웨어 환경과 구성 요소가 사전에 정의되어 있어야 하며, 개발 과정의 후반부에 주로 적용함

4) Putnam 모형

- **Rayleigh-Norden** 곡선의 **노력 분포도**를 이용한 프로젝트 비용 산정기법
- 소프트웨어 생명 주기의 전 과정 동안에 사용될 노력의 분포를 가정해주는 모형
- 대형 프로젝트의 노력 분포 산정에 이용되는 기법
- 개발 기간이 늘어날수록 프로젝트 적용 인원의 노력이 감소함

*SLIM : **Putnam 모형을 기초로 해서 만든 자동화 추정 도구**

훈남(Putnam)이 노력(노력분포도)해서 슬림(SLIM)해졌네

5) 기능 점수(FP; Functional Point) 모형

- 알브레히트(Albrecht)가 제안한 것으로, 소프트웨어의 기능을 증대시키는 요인별로 가중치를 부여하고, 요인별 가중치를 합산하여 **총 기능점수를 산출**하며 총 기능점수와 영향도를 이용하여 기능점수(FP)를 구한 후 이를 이용해서 비용을 산정하는 기법

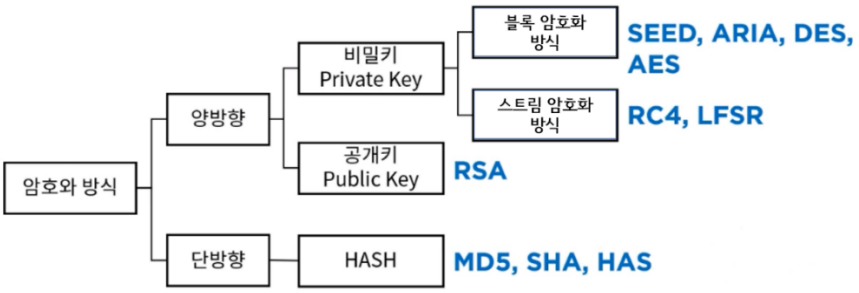
*ESTIMACS : 다양한 프로젝트와 개인별 요소를 수용하도록 FP 모형을 기초로 개발된 자동화 추정 도구

기능점수 모형에서 비용산정에 이용되는 요소

- 명령어(사용자 질의수)
- 데이터파일
- 출력보고서
- ~~클래스 인터페이스~~
- 자료 입력(입력 양식)
- 정보 출력(출력 보고서)
- 명령어(사용자 질의수)

- 데이터 파일
- 필요한 외부 루틴과의 인터페이스

5 암호 알고리즘



개인키 암호화 방식(Private Key Encryption) => 대칭키

- **암호화 키와 복호화 키가 동일하다**
- 비밀키는 DB 사용 권한이 있는 사용자만 나눠 가짐
- 암호화/복호화 속도가 빠름, 알고리즘이 단순함, 파일의 크기가 작음
- 관리해야 할 키의 수가 많음
- **블록 암호화 : DES, AES, SEED, ARIA**
- **스트림 암호화 : RC4, LFSR**

● **DES**

- 1975 년 미국 NBS 에서 발표한 개인키 암호화 알고리즘 (구 미국 표준)
- 블록 크기는 64 비트이며, 키 길이는 56 비트 키를 사용

● **AES**

- **암호화 키와 복호화 키가 동일한 암호화 알고리즘**
- DES 의 보안 취약점을 대체하기 위해 고안된 미국 표준 방식으로 현재 표준 대칭키 암호화 기법
- 블록 크기는 128 비트이며, 키 길이에 따라 128, 192, 256 으로 분류

● **SEED**

- 1999 년 한국인터넷진흥원(KISA)에서 개발한 블록 암호화 알고리즘
- 블록 크기는 128 비트이며, 키 길이에 따라 128, 256 으로 분류

● **ARIA**

- 2004 년 국가정보원과 산학연협회가 개발한 블록 암호화 알고리즘
- 블록 크기는 128 비트이며, 키 길이에 따라 128, 192, 256 으로 분류
- SEED 이후로 나온 대한민국의 국가 암호 표준(AES 와 동일)

공개키 암호화 방식(Public Key Encryption) => 비대칭키

- 비대칭 암호기법이라고도 한다.
- 대표적인 기법은 RSA 기법이 있다.
- 키 분배가 용이하고, 관리해야 할 키 개수가 적다.
- **암호와 해독에 다른 키를 사용한다.**
- 암호키는 공개되어 있어서 누구나 사용할 수 있다.
- 해독키를 가진 사람만이 해독할 수 있다.
- ~~공개키로 암호화된 메시지는 반드시 공개키로 복호화 해야 한다.~~

~~- 키분배가 비밀키 시스템(Private key system) 보다 어렵다.~~
- 암호화/복호화 속도가 느리고 알고리즘이 복잡하다. 파일의 크기가 큼
- 데이터를 암호화할 때 사용하는 키(공개키)는 DB 사용자에게 공개하고, 복호화 할 때의 키(비밀키)는 관리자가 관리하는 방법
- **RSA, Diffie-Hellman**

● **RSA**
- 비대칭 암호화 방식으로 소수를 활용한 암호화 알고리즘
- 큰 숫자를 소인수분해하기 어렵다는 기반 하에 1978 년 MIT 에 의해 제안된 공개키 암호화 알고리즘
- **소인수분해** 문제를 이용한 공개키 암호화 기법에 널리 사용되는 암호 알고리즘 기법

● ECC : **타원곡선** 함수를 이용한 암호화 기법

● EIGAMAI(이산대수)

***공개키 암호에 대한 설명으로 틀린 것은?**

~~1. 10 명이 공개키 암호를 사용할 경우 5 개의 키가 필요하다. => 암호키는 2N = 20 개 필요~~
2. 복호화키는 비공개 되어 있다.
3. 송신자는 수신자의 공개키로 문서를 암호화한다.
4. 공개키 암호로 널리 알려진 알고리즘은 RSA 가 있다.

***블록 암호화 방식이 아닌 것은?**

1. DES
~~2. RC4 => 스트림 암호 방식~~
3. AES
4. SEED

***다음 암호 알고리즘 중 성격이 다른 하나는?**

- MD4, MD5, SHA-1 = 해시 암호화 알고리즘
- AES = 대칭 키 암호화 알고리즘

***스트림 암호화 방식의 설명으로 옳지 않은 것은?**
- 비트/바이트/단어들을 순차적으로 암호화한다.
- RC4 는 스트림 암호화 방식에 해당한다.
- 대칭키 암호화 방식이다.
~~- 해쉬 함수를 이용한 해쉬 암호화 방식을 사용한다. => 단방향 암호화 방식~~

● **해시(Hash)**
- 임의의 길이의 입력 데이터나 메시지를 고정된 길이의 값이나 키로 변환하는 것
- 해시 알고리즘을 해시 함수라고 부르며, 해시 함수로 변환된 값이나 키를 해시값 또는 해시키라 부름

SHA 시리즈, **MD5**, N-NASH, SNEFRU
- 임의의 길이의 입력 데이터를 받아 고정된 길이의 해쉬 값으로 변환한다.
- 대표적인 해쉬 알고리즘으로 HAVAL, SHA-1 등이 있다.
- 해쉬 함수는 일방향 함수(One-way function)이다.
~~- 주로 공개키 암호화 방식에서 키 생성을 위해 사용한다.~~

***Salt :** 시스템에 저장되는 패스워드들은 Hash 또는 암호화 알고리즘의 결과 값으로 저장된다. 이때 암호공격을

막기 위해 똑같은 패스워드들이 다른 암호 값으로 저장되도록 추가되는 값을 의미한다.

6 각종 버전

국제 제품 품질 표준

● **ISO/IEC 9126 :** 소프트웨어 품질 특성 및 척도에 대한 표준화

***ISO/IEC 9126 의 소프트웨어 품질 특성**

- 기능성(Functionality) : **적합성**, **정확성**, 상호 운용성, **보안성**, 호환성
- 신뢰성(Reliability) : 성숙성, 결함 허용성, 회복성
- 사용성(Usability) : 이해성, **학습성**, 운용성, 친밀성
- 효율성(Efficiency) : 시간 효율성, 자원 효율성
- 유지 보수성(Maintainability) : 분석성, 변경성, 안정성, 시험성
- 이식성(Portability) : 적용성, 설치성, 대체성, 공존성

***ISO/IEC 9126 의 소프트웨어 품질 특성 중 기능성(Functionlity)의 하위 특성으로 옳지 않은 것은?**

- ~~학습성~~, 정합성, 정확성, 보안성

● **ISO/IEC 12119**

- **패키지 소프트웨어의 일반적인 제품 품질 요구사항 및 테스트를 위한 국제 표준**
- 패키지 소프트웨어 평가
- 패키지 소프트웨어 제품테스트 국제 표준
- 현재 ISO/IEC 12119 이 대체되어 ISO/ISE 25010 이 국제표준

● **ISO/IEC 14598 :** 소프트웨어 제품 평가

● **ISO/IEC 25000**

- SW 품질 평가 통합 모델
- SQuaRE 로도 불리며 위 3 개 표준을 통합

국제 프로세스 품질 표준

● **ISO/IEC 12207 - 표준 소프트웨어 생명 주기 프로세스**

- 소프트웨어의 개발, 운영, 유지보수 등을 체계적으로 관리하기 위한 소프트웨어 생명 주기 표준을 제공함
- **기본** 생명 주기 프로세스 : **획득**, 공급, **개발**, 운영, **유지보수**
- **지원** 생명 주기 프로세스 : 품질보증, 검증, 확인, 활동검토, 문제해결
- **조직** 생명 주기 프로세스 : 관리, 기반구조, 훈련, 개선

***ISO 12207 표준의 기본 생명주기의 주요 프로세스에 해당하지 않는 것은?**

- **획득, 개발, ~~성능평가~~, 유지보수**

● **ISO/IEC 15504 (SPICE)**

- **소프트웨어 프로세스**에 대한 개선 및 능력 측정 기준에 대한 국제 표준
- 불완전 → 수행 → 관리 → 확립 → 예측 → 최적화

***SPICE 의 프로세스 수행 능력 단계**

0 단계 – 불완전(Incomplete) : 프로세스가 구현되지 않았거나 목적을 달성하지 못함

1 단계 – 수행(Performed) : 프로세스가 수행되고 목적이 달성됨
2 단계 – 관리(Managed) : 정의된 자원의 한도 내에서 그 프로세스가 작업 산출물을 인도함
3 단계 – 확립(Established) : 소프트웨어 공학 원칙에 기반하여 정의된 프로세스가 수행됨
4 단계 – 예측(Predictable) : 프로세스가 목적 달성을 위해 통제되고, 양적인 측정을 통해서 일관되게 수행됨
5 단계 – 최적화(Optimizing) : 프로세스 수행을 최적화하고, 지속적인 개선을 통해 업무 목적을 만족시킴

***SPICE 의 목적**
- 프로세스 개선을 위해 개발 기관이 스스로 평가
- 기관에서 지정한 요구조건의 만족여부를 개발 조직이 스스로 평가
- 계약 체결을 위해 수탁 기관의 프로세스를 평가

● **CMMI(Capability Maturity Model Integration, 능력 성숙도 통합 모델)**
- 조직차원의 성숙도를 평가하는 단계별 표현과 프로세스 영역별 능력도를 평가하는 연속적 표현이 있음
- 소프트웨어 개발 조직의 업무 능력 및 조직의 성숙도를 평가하는 모델
- CMM 발전모형

***프로세스 성숙도 5 단계 # 초최정관**
- 초기(Initial) : 정의된 프로세스 없음 (작업자 능력에 따라 성공 여부 결정)
- 관리(Managed) : 규칙화된 프로세스 (특정한 프로젝트 내의 프로세스 정의 및 수행)
- 정의(Defined) : 표준화된 프로세스 (조직의 표준 프로세스를 활용하여 업무 수행)
- 정량적 관리(Quantitatively Managed) : 예측 가능한 프로세스 (프로젝트를 정량적으로 관리 및 통제))
- 최적화(Optimizing) : 지속적 개선 프로세스 (프로세스 역량 향상을 위해 지속적인 프로세스 개선

***CMM(Capability Maturity Model) 모델의 레벨이 아닌 것은?**
- 관리 단계
- 정의 단계
- 최적 단계
- ~~계획 단계~~

7 경로 제어, 트래픽 제어

경로 제어 (라우팅 프로토콜, 네트워크 3 계층)

● **RIP(Routing Information Protocol)**
- IGP(Interior Gateway Protocol)로 Bellman-Ford 알고리즘을 이용하여 최적의 경로를 설정한다.
- **거리 벡터 라우팅 프로토콜이라고 한다.**
- **소규모** 네트워크 환경에 적합하다.
- 최단경로탐색에는 **Bellman-Ford** 알고리즘을 사용한다.
- 최대 홉 카운트를 **15 홉 이하**로 한정하고 있다.
- 패킷을 목적지까지 전달하기 위해 사용

● **OSPF(Open Shortest Path First)**
- 네트워크 변화에 신속하게 대처할 수 있다.
- 멀티캐스팅을 지원한다.
- 최단 경로 탐색에 **Dijkstra** 알고리즘을 사용한다.
- **링크 상태 알고리즘**을 사용하며, **대규모** 네트워크에 적합한 것

- **Link-statc** 방식의 라우팅 프로토콜
- 네트워크에 변화가 있을 때에만 갱신
- RIP 한계를 극복

● **BGP(Border Gateway Protocol)**
- 자치 시스템 간의 라우팅 프로토콜로 EGP(Exterior Gateway Protocol)의 단점을 보완하기 위해 만들어짐
- 초기에 BGP 라우터들이 연결될 때는 전체 경로를 나타내는 라우팅 테이블을 교환하고, 이후에는 변화된 정보만 교환
- 라우터에 의해서 전체 경로 교환
- 루프 방지
- 179 번 포트 이용한 TCP 서비스 사용
- 오류제어나 흐름제어 필요하지않음

● **IGRP**
- Cisco System 의 고유의 프로토콜
- 홉 수를 기준으로 한 정보 전송
- 라우팅 경로 결정 시 회선의 전송능력 지연시간 사용률 신뢰도 바탕
- 독립적 네트워크 내에서만 사용

● **EIGRP**
- IGRP 의 Metric 구성 값에 256 을 곱하여 작동
- 프로토콜 종속 모듈
- 신뢰성 전송 프로토콜 (순차적 패킷 전달)
- 낮은 대역폭 및 빠른 수렴
- 업데이트 확산 알고리즘

트래픽 제어(Traffic Control)
- 네트워크의 보호, 성능 유지, 네트워크 자원의 효율적인 이용을 위해 전송되는 패킷의 흐름 또는 그 양을 조절하는 기능

1) 흐름 제어(Flow Control) : 송수신 측 사이에 전송되는 패킷의 수를 제어

● **정지 및 대기(Stop and Wait)**
- TCP 흐름제어기법 중 프레임이 손실되었을 때, 손실된 프레임 1 개를 전송하고 수신자의 응답을 기다리는 방식으로 한 번에 프레임 1 개만 전송할 수 있는 기법
- 수신 측의 확인 신호(ACK)를 받은 후에 다음 패킷을 전송하는 방식 → 한번에 하나의 패킷 전송

● **슬라이딩 윈도우(Sliding Window)**
- 수신 측의 확인 신호(ACK)를 받지 않더라도 미리 정해진 패킷의 수만큼 연속적으로 전송하는 방식
→ 한번에 여러 개 패킷 전송
- 수신 측으로부터 송신한 패킷에 대한 긍정 수신 응답(ACK)이 전달된 경우 윈도우 크기는 증가하고, 수신 측으로부터 부정 수신 응답(NAK)이 전달된 경우 윈도우 크기는 감소함
- 한 번에 여러 패킷(프레임)을 전송할 수 있어 전송 효율이 좋은 기법
- 수신 측으로부터 이전에 송신한 패킷에 대한 긍정 수신 응답(ACK)이 전달된 경우 윈도우 크기는 증가하고, 수신 측으로부터 이전에 송신한 패킷에 대한 부정 수신 응답(NAK)이 전달된 경우 윈도우 크기는 감소한다

2) 폭주(혼잡) 제어(Congestion Control) : 네트워크 내의 패킷 수를 조절하여 네트워크의 오버플로를 방지

● **느린 시작(Slow Start)**

- 윈도우의 크기를 1, 2, 4, 8, ... 같이 2 배씩 지수적으로 증가시켜 초기에는 느리지만 갈수록 빨라짐
- 전송 데이터의 크기가 임계 값에 도달하면 혼잡 회피 단계로 넘어감
- 패킷이 문제없이 도착하면 혼잡 윈도우 크기를 패킷마다 1 씩 증가시켜 한 주기가 지나면 혼잡 윈도우 크기가 2 배로 되지만, 혼잡 현상 발생시 혼잡 윈도우 크기를 1 로 줄여버리는 방식이다.

● **혼잡 회피(Congestion Avoidance)**

- 느린 시작의 지수적 증가가 임계 값에 도달하면 혼잡으로 간주하고 회피를 위해 윈도우의 크기를 1 씩 선형적으로 증가시켜 혼잡을 예방하는 방식
- 네트워크 내에서 패킷의 지연이 너무 높아지게 되어 트래픽이 붕괴되지 않도록 패킷의 흐름을 제어하는 트래픽 제어(종류 : AMID, Slow Start)

3) 교착 상태(Dead Lock) 방지

- 교환기 내에 패킷들을 축적하는 기억 공간이 꽉 차 있을 때 다음 패킷들이 기억 공간에 들어가기 위해 무한정 기다리는 형상

[8] 병행제어, 회복

병행제어(Concurrency Control)

- 다중 프로그램의 이점을 활용하여 동시에 여러 개의 트랜잭션을 병행수행할 때, 동시에 실행되는 트랜잭션들이 데이터베이스의 일관성을 파괴하지 않도록 트랜잭션 간의 상호 작용을 제어하는 것

병행제어의 목적

- 데이터베이스의 공유 최대화
- 시스템의 활용도 최대화
- 데이터베이스의 일관성 **유지**
- 사용자에 대한 응답시간 최소화

병행수행의 문제점

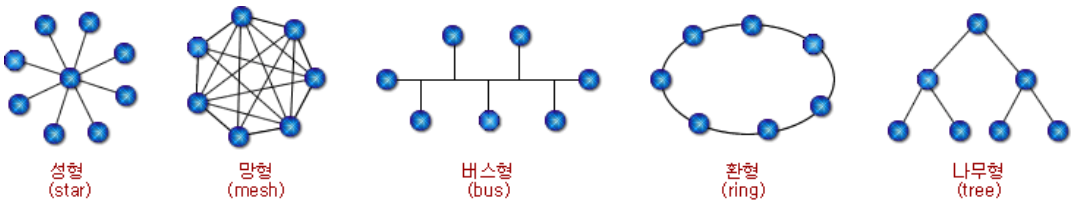
- 갱신 분실(Lost Update)
  - 2 개 이상의 트랜잭션이 같은 자료를 공유하여 갱신할 때 갱신 결과의 일부가 없어지는 현상 (덮어쓸 때)
- 비완료 의존성(Uncommitted Dependency)
  - 하나의 트랜잭션 수행이 실패한 후 회복되기 전에 다른 트랜잭션이 실패한 갱신 결과를 참조하는 현상
  - 임시 갱신이라고도 함

  → 현황파악 오류(Dirty Read)
- 모순성(Inconsistency)
  - 2 개의 트랜잭션이 병행수행될 때 원치 않는 자료를 이용함으로써 발생하는 문제
  - 불일치 분석이라고도 함 (일관성 결여)
- 연쇄 복귀(Cascading Rollback)
  - 병행수행되던 트랜잭션들 중 어느 하나에 문제가 생겨 Rollback 하는 경우 다른 트랜잭션도 함께 Rollback 되는 현상
  - 부분취소 불가능 현상

회복(Recovery)

- 트랜잭션들을 수행하는 도중 장애가 발생하여 데이터베이스가 손상되었을 때 손상되기 이전의 정상 상태로 복구하는 작업
- 장애의 유형
  - **트랜잭션 장애** : 트랜잭션 내부의 비정상적인 상황으로 인해 프로그램 실행이 중지되는 현상
  - **시스템 장애** : 데이터베이스에 손상을 입히지는 않으나 하드웨어 오동작, 소프트웨어의 손상, 교착상태 등에 의해 모든 트랜잭션의 연속적인 수행에 장애를 주는 현상
  - **미디어 장애** : 저장장치인 디스크 블록의 손상이나 디스크 헤드의 충돌 등에 의해 데이터베이스의 일부 또는 전부가 물리적으로 손상된 상태
- 회복 관리기(Recovery Management) : DMBS 의 구성 요소, 트랜잭션 실행이 성공적으로 완료되지 못하면 트랜잭션이 데이터 베이스에 생성했던 모든 변화를 취소(Undo)시키고, 트랜잭션 수행 이전의 원래 상태로 복구하는 역할 담당 - 메모리 덤프, 로그(Log)를 이용하여 회복 수행

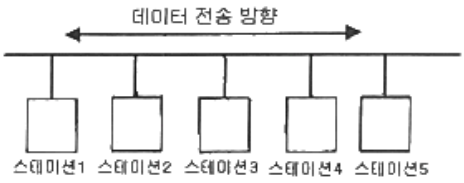
9 네트워크 구축



버스형(Bus)

- 한 개의 통신 회선에 여러 대의 단말장치가 연결되어 있는 형태 → **LAN** 에서 사용
- 물리적 구조가 간단하고, 단말장치의 추가와 제거가 용이
- 단말장치가 고장나더라도 통신망 전체에 영향을 주지 않기 때문에 신뢰성 향상
- 기밀 보장이 어렵고, 통신 회선의 길이에 제한이 있음

*다음 LAN 의 네트워크 토폴로지는 어떤 형인가? **버스형**



계층형(Tree, 트리형, 분산형)

- 중앙 컴퓨터와 일정 지역의 단말장치까지는 하나의 통신 회선으로 연결시키고, 이웃하는 단말장치는 일정 지역 내에 설치된 중간 단말장치로부터 다시 연결시키는 형태 → 분산 처리 시스템

링형(Ring, 환형, 루프형)

- 컴퓨터와 단말장치들을 서로 이웃하는 것끼리 포인트 투 포인트(Point-to-Point) 방식으로 연결시킨 형태 → **LAN** 에서 사용
- 분산 및 집중 제어 모두 가능하고 중계기 수가 많아짐
- 단말장치의 추가/제거 및 기밀 보호가 어려움
- 각 단말장치에서 전송 지연이 발생할 수 있음
- 데이터는 단방향 또는 양방향으로 전송할 수 있고, **단방향 링**의 경우 컴퓨터, 단말장치, 통신 회선 중 어느 하나라도 고장나면 전체 통신망에 영향을 미침

- 각 노드가 공평한 서비스를 받는다.
- 전송매체와 노드의 고장 발견이 쉽다.
- 새로운 노드를 추가할 경우 통신회선을 절단해야 한다.
- 목적지에 도달하는데 단방향인 경우 최대 n-1 개의 노드를 거쳐야 한다.

성형(Star, 중앙 집중형)
- 중앙에 중앙 컴퓨터가 있고, 이를 중심으로 단말장치들이 연결되는 중앙 집중식의 네트워크 구성 형태
- 포인트 투 포인트(Point-to-Point) 방식으로 회선을 연결
- 단말장치의 추가와 제거가 쉽지만, 중앙 컴퓨터가 고장나면 전체 통신망의 기능이 정지됨
- 중앙 집중식이므로 교환 노드의 수가 가장 적음

망형(Mesh, 네트워크형)
- 모든 지점의 컴퓨터와 단말장치를 서로 연결한 형태로, 노드의 연결성이 높음
- 많은 단말장치로부터 많은 양의 통신을 필요로 하는 경우 유리
- **공중 데이터 통신망**에서 사용되며, 통신 회선의 총 경로가 가장 길
- 통신 회선 장애 시 다른 경로를 통하여 데이터 전송 가능

10 Secure OS

- 컴퓨터 운영체제의 커널에 보안 기능을 추가한 것으로 운영체제의 보안상 결함으로 인하여 발생 가능한 각종 해킹으로부터 시스템을 보호하기 위하여 사용된다.

보호 방법
- 암호적 분리(Cryptographic Separation) : 내부 정보를 암호화하는 방법
- 논리적 분리(Logical Separation) : 프로세스의 논리적 구역을 지정하여 구역을 벗어나는 행위를 제한하는 방법
- 시간적 분리(Temporal Separation) : 동일 시간에 하나의 프로세스만 수행되도록 하여 동시 실행으로 발생하는 보안 취약점을 제거하는 방법
- 물리적 분리(Physical Separation) : 사용자별로 특정 장비만 사용하도록 제한하는 방법
- **#암논시물** → 구현하기 복잡한 순서 : 암 > 논 > 시 > 물

참조 모니터(Reference Monitor)
- 보호대상의 객체에 대한 접근통제를 수행하는 추상머신이며, 이를 실제로 구현한 것이 보안 커널임
- 3 가지 특징
  - 격리성(Isolation) : 부정 조작 불가능
  - 검증 가능성(Verifiability) : 적절히 구현됐다는 것 확인 가능
  - 완전성(Completeness) : 우회 불가능

Secure OS 의 보안 기능
- 식별 및 인증
- 임의적 접근통제(DAC)
- 강제적 접근통제(MAC)
- ~~고가용성 지원~~
- 계정관리
- 객체 재사용 방지

- 완전한 중재 및 조정
- 감사 및 감사기록 축소
- 안전한 경로
- 보안 커널 변경 방지
- 해킹 방지(Anti-Hacking)
- 통합 관리

11 용어 - 네트워크 관련

VPN(Virtual Private Network, 가상 사설 통신망)

- 이용자가 인터넷과 같은 공중망에 사설망을 구축하여 마치 전용망을 사용하는 효과를 가지는 보안 솔루션

Mesh Network(메시 네트워크)

- 기존 무선 랜의 한계 극복을 위해 등장하였으며, 대규모 디바이스의 네트워크 생성에 최적화되어 차세대 이동통신, 홈네트워킹, 공공 안전 등의 특수목적을 위한 새로운 방식의 네트워크 기술을 의미하는 것

스마트 그리드

- 전기 및 정보통신기술을 활용하여 전력망을 지능화, 고도화함으로써 고품질의 전력서비스를 제공하고 에너지 이용효율을 극대화하는 전력망

WI-SUN(와이선)

- 스마트 그리드와 같은 장거리 무선 통신을 필요로 하는 IoT 서비스를 위한 저전력 장거리(LPWA) 통신 기술

PICONET(피코넷)

- 여러 개의 독립된 통신장치가 UWB 통신 기술 또는 블루투스 기술을 사용하여 통신망을 형성하는 무선 네트워크 기술

LAN (근거리 통신망, Local Area Network)

- 비교적 근거리에 있는 컴퓨터, 프린터, 테이프 등의 자원을 연결하여 구성하며 주로 자원 공유의 목적이다.
- 사이트 간의 거리가 짧아 데이터의 전송 속도가 빠르고, 에러 발생율이 낮음
- 주로 버스형, 링형 구조 사용

*현재 많이 사용되고 있는 LAN 방식인 "10BASE-T"에서 "10"의 의미는? **데이터 전송속도가 10Mbps**

WAN (원거리 통신망, Wide Area Network)

- 대륙과 대륙 같이 멀리 떨어진 사이트들을 연결하여 구성
- 사이트 간의 거리가 멀기 때문에 통신 속도가 느리고, 에러 발생율이 높음

NFC (근거리 무선 통신, Near Field Communication)

- 고주파(HF)를 이용한 근거리 무선 통신 기술
- 아주 가까운 거리에서 양방향 통신을 지원하는 RFID(Radio Frequency Identification) 기술의 일종

IoT(Internet of Things, 사물 인터넷)

- 사람과 사물, 사물과 사물 간에 지능 통신을 할 수 있는 M2M 의 개념을 인터넷으로 확장하여 사물은 물론, 현실과 가상 세계의 모든 정보와 상호 작용하는 개념

- 정보 통신 기술 기반 실세계와 가상세계의 사물을 인터넷으로 연결하여 서비스 제공하는 기술

M2M(Machine to Machine)
- 무선 통신을 이용한 기계와 기계 사이의 통신

Mobile Computing(모바일 컴퓨팅)
- 휴대형 기기로 이동하면서 자유로이 네트워크에 접속하여 업무를 처리할 수 있는 환경

Cloud Computing(클라우드 컴퓨팅)
- 각종 컴퓨팅 자원을 중앙 컴퓨터에 두고 인터넷 기능을 갖는 단말기로 언제 어디서나 인터넷을 통해 컴퓨터 작업을 수행할 수 있는 환경(사설 클라우드, 공용 클라우드, 하이브리드 클라우드)

Grid Computing(그리드 컴퓨팅)
- 수 많은 컴퓨터를 하나의 컴퓨터처럼 묶어 분산 처리하는 방식

Mobile Cloud Computing(MCC; 모바일 클라우드 컴퓨팅)
- 클라우드 서비스를 이용하여 소비자와 소비자의 파트너가 모바일 기기로 클라우드 컴퓨팅 인프라를 구성하여 여러 가지 정보와 자원을 공유하는 ICT 기술

Inter-Cloud Computing(인터클라우드 컴퓨팅)
- 여러 클라우드 서비스 제공자들이 제공하는 클라우드 서비스나 자원을 연결하는 기술

NDN(Named Data Networking)
- 콘텐츠 자체의 정보와 라우터 기능만으로 데이터 전송을 수행하는 기술
- 콘텐츠 중심 네트워킹(CNN; Content Centric Networking)과 같은 개념으로 기존의 IP 망을 대체할 새로운 인터넷 아키텍처

NGN(Next Generation Network, 차세대 통신망)
- ITU-T 에서 개발하고 있는 유선망 기반의 차세대 통신망으로, 하나의 망이 인터넷처럼 모든 정보와 서비스를 패킷으로 압축하여 전송
- 유선망 기반의 차세대 통신망 유선망뿐만 아니라 이동 사용자를 목표로 함

SDN(Software Defined Networking, 소프트웨어 정의 네트워킹)
- 네트워크를 컴퓨터처럼 모델링하여 여러 사용자가 각각의 소프트웨어들로 네트워킹을 가상화하여 제어하고 관리하는 네트워크

지능형 초연결망
- 국가망에 소프트웨어 정의 기술을 적용하는 방법(정부 주관 사업)

UWB(Ultra WideBand, 초광대역)
- 짧은 거리에서 많은 양의 디지털 데이터를 낮은 전력으로 전송하기 위한 무선 기술

WBAN(Wireless Body Area Network)
- 웨어러블 또는 몸에 심는 형태의 센서나 기기를 무선으로 연결하는 개인 영역 네트워킹 기술

GIS (Geographic Information System, 지리 정보 시스템)
- 지리적인 자료를 위성을 이용해 모든 사물의 위치 정보를 제공해주는 시스템

USN(Ubiquitous Sensor Network, 유비쿼터스 센서 네트워크)
- 필요한 모든 곳에 RFID 태그를 부착하고 사물의 인식 정보는 물론 주변의 환경정보까지 탐지하여 이를 네트워크에 연결해 정보를 관리하는 것
- 각종 센서로 수집한 정보를 무선으로 수집할 수 있도록 구성한 네트워크

SON(Self Organizing Network, 자동 구성 네트워크)
- 주변 상황에 맞추어 스스로 망을 구성하는 네트워크

Ad-hoc Network(애드 혹 네트워크)
- 재난 현장과 같이 별도의 고정된 유선망을 구축할 수 없는 장소에서 구성한 네트워크

Network Slicing(네트워크 슬라이싱)
- 5G 네트워크를 구현하는 중요한 핵심 기술로, 하나의 물리적인 코어 네트워크 인프라를 독립된 다수의 가상 네트워크로 분리하는 네트워크 기술

저전력 블루투스 기술(BLE; Bluetooth Low Energy)
- 일반 블루투스와 동일한 2.4GHz 주파수 대역을 사용하지만 연결되지 않은 대기 상태에서는 절전모드를 유지하는 기술

비콘
- 블루투스 기반의 근거리 무선통신 장치

포스퀘어(Foursquare)
- 위치 기반 소셜 네트워크 서비스

12 용어 – 소프트웨어 관련

서비스 지향 아키텍처(SOA; Service Oriented Architecture)
- 기업의 소프트웨어 인프라인 정보시스템을 공유와 재사용이 가능한 서비스 단위나 컴포넌트 중심으로 구축하는 정보기술 아키텍처
- 정보를 누구나 이용 가능한 서비스로 간주하고 연동과 통합을 전제로 아키텍처를 구축

***서비스 지향 아키텍처 기반 애플리케이션을 구성하는 층이 아닌 것은?** ★
- 표현 층(Presentation Layer)
- 프로세스 층(Process Layer)
- 비즈니스 층(Business Layer)
- ~~제어 클래스층~~
- 서비스 층(Service Layer)
- 영속 층(Persistency Layer)

Digital Twin(디지털 트윈)

- 물리적인 사물과 컴퓨터에 동일하게 표현되는 **가상의 모델**로 실제 물리적인 자산 대신 소프트웨어로 가상화함으로써 실제 자산의 특성에 대한 정확한 정보를 얻을 수 있고, 자산 최적화, 돌발사고 최소화, 생산성 증가 등 설계부터 제조, 서비스에 이르는 모든 과정의 효율성을 향상시킬 수 있는 모델

Mashup(매시업)

- 웹에서 제공하는 정보 및 서비스를 이용하여 새로운 소프트웨어나 서비스, 데이터베이스 등을 만드는 기술

텐서플로(Tensorflow)

- 구글의 구글 브레인 팀이 제작하여 공개한 기계 학습(Machine Leaming)을 위한 오픈소스 소프트웨어 라이브러리

Baas

- 블록체인 개발환경을 클라우드로 서비스하는 개념
- 블록체인 네트워크에 노드의 추가 및 제거가 용이
- 블록체인의 기본 인프라를 추상화하여 블록체인 응용프로그램을 만들 수 있는 클라우드 컴퓨팅 플랫폼

Grayware(그레이웨어)

- 바이러스, 트로잔등 악성프로그램과는 다르게 사용자 동의를 받아 설치하는 프로그
- 사용자 입장에서는 유용 혹은 악의적일 수 있는 애드웨어(광고), 트랙웨어(스파이웨어), 악성 공유웨어

인공지능(AI; Artificial Intelligence)

- 인간의 두뇌와 같이 컴퓨터 스스로 추론, 학습, 판단 등 인간지능적인 작업을 수행하는 시스템
- 인공지능 개발언어 : 리스프(LISP), 프롤로그(PROLOG)

Neuralink(뉴럴링크)

- 사람이 인공지능에 대항할 수 있는 더 높은 수준의 기능에 도달하도록 컴퓨터와 뇌를 연결한다는 개념

Deep Learning(딥러닝)

- 인간의 두뇌를 모델로 만들어진 인공 신경망(ANN; Artificial Neural Network)을 기반으로 하는 기계 학습 기술

Expert System(전문가 시스템)

- 의료 진단 등과 같은 특정 분야의 전문가가 수행하는 고도의 업무를 지원하기 위한 컴퓨터 응용 프로그램

Blockchain(블록체인)

- P2P(Peer-to-Peer) 네트워크를 이용하여 온라인 금융 거래 정보를 온라인 네트워크 참여자(Peer)의 디지털 장비에 분산 저장하는 기술 (비트 코인)

분산 원장 기술(DLT; Distributed Ledger Technology)

- 중앙 관리자나 중앙 데이터 저장소가 존재하지 않고 P2P 망 내의 참여자들에게 모든 거래 목록이 분산 저장되어 거래가 발생할 때마다 지속적으로 갱신되는 디지털 원장

Hash(해시)

- 임의의 길이의 입력 데이터나 메시지를 고정된 길이의 값이나 키로 변환하는 것

양자 암호키 분배(QKD; Quantum Key Distribution)

- 양자 통신을 위해 비밀키를 분배하여 관리하는 기술로, 두 시스템이 암호 알고리즘 동작을 위한 비밀키를 안전하게 공유하기 위해 양자 암호키 분배 시스템을 설치하여 운용하는 방식으로 활용

프라이버시 강화 기술(PET; Privacy Enhancing Technology)

- 개인정보 위험 관리 기술로, 다양한 사용자 프라이버시 보호 기술을 통칭함

디지털 저작권 관리(DRM; Digital Rights Management)

- 인터넷이나 기타 디지털 매체를 통해 유통되는 데이터의 저작권을 보호하기 위해 데이터의 안전한 배포를 활성화하거나 불법 배포를 방지하기 위한 시스템

공통 평가 기준(CC; Common Criteria)

- 정보화 순기능 역할을 보장하기 위해 정보화 제품의 정보보호 기능과 이에 대한 사용 환경 등급을 정한 기준

개인정보 영향평가 제도(PIA; Privacy Impact Assessment)

- 개인 정보를 활용하는 새로운 정보시스템의 도입 및 기존 정보시스템의 중요한 변경 시 시스템의 구축, 운영이 기업의 고객은 물론 국민의 사생활에 미칠 영향에 대해 미리 조사, 분석, 평가하는 제도

리치 인터넷 애플리케이션(RIA; Rich Internet Application)

- 플래시 애니메이션 기술과 웹 서버 애플리케이션 기술을 통합하여 기존 HTML 보다 역동적인 웹페이지를 제공하는 신개념의 플래시 웹페이지 제작 기술

Semantic Web(시맨틱 웹)

- 컴퓨터가 사람을 대신하여 정보를 읽고 이해하고 가공하여 새로운 정보를 만들어 낼 수 있도록 이해하기 쉬운 의미를 가진 차세대 지능형 웹

Vaporware(증발품)

- 판매 계획 또는 배포 계획은 발표되었으나 실제로 고객에게 판매되거나 배포되지 않고 있는 소프트웨어

오픈 그리드 서비스 아키텍처(OGSA; Open Grid Service Architecture)

- 애플리케이션 공유를 위한 웹 서비스를 그리드 상에서 제공하기 위해 만든 개방형 표준

서비스형 소프트웨어(SaaS; Software as a Service)

- 소프트웨어의 여러 기능 중에서 사용자가 필요로 하는 서비스만 이용할 수 있도록 한 소프트웨어
- cf) 서비스형 인프라(IaaS), 서비스형 플랫폼(PaaS)

Software Escrow(소프트웨어 에스크로, 임치)

- 소프트웨어 개발자의 지식재산권을 보호하고 사용자는 저렴한 비용으로 소프트웨어를 안정적으로 사용 및 유지보수 받을 수 있도록 소스 프로그램과 기술 정보 등을 제 3 의 기관에 보관하는 것

복잡 이벤트 처리(CEP; Complex Event Processing)

- 실시간으로 발생하는 많은 사건들 중 의미가 있는 것만을 추출할 수 있도록 사건 발생 조건을 정의하는 데이터 처리 방법

증강 현실(AR; Augmented Reality)

- 실제 촬영한 화면에 가상의 정보를 부가하여 보여주는 기술, 혼합현실(MR; Mixed Reality)이라고도 부름

가상 현실(VR; Virtual Reality)

- 컴퓨터 등을 사용한 인공적인 기술로 만들어낸 실제와 유사하지만 실제가 아닌 어떤 특정한 환경이나 상황 혹은 그 기술 자체를 의미함

13 용어 – DB 관련

Data Mining(데이터 마이닝)

- 빅데이터 분석 기술 중 대량의 데이터를 분석하여 데이터 속에 내재되어 있는 변수 사이의 상호관례를 규명하여 일정한 패턴을 찾아내는 기법

Hadoop(하둡)

- 오픈 소스를 기반으로 한 분산 컴퓨팅 플랫폼이다.
- 일반 PC 급 컴퓨터들로 가상화된 대형 스토리지를 형성한다.
- 다양한 소스를 통해 생성된 빅데이터를 효율적으로 저장하고 처리한다.

스쿱(Sqoop)

- 하둡(Hadoop)과 관계형 데이터베이스간에 데이터를 전송할 수 있도록 설계된 도구

Data Warehouse(데이터 웨어하우스)

- 조직이나 기업체의 중심이 되는 업무시스템에서 모아진 정보를 일관된 스키마로 저장한 저장소

Tajo (타조)

- 하둡(Hadoop) 기반 데이터웨어하우스 시스템

Big Data(빅데이터)

- 기존의 관리 방법이나 분석 체계로는 처리하기 어려운 막대한 양의 정형 또는 비정형 데이터 집합
- 데이터의 양, 데이터의 다양성, 데이터의 속도

Broad Data(브로드 데이터)

- 다양한 채널에서 소비자와 상호 작용을 통해 생성된, 이전에 사용하지 않거나 알지 못했던 새로운 데이터나, 기존 데이터에 새로운 가치가 더해진 데이터

Meta Data(메타 데이터)

- 일련의 데이터를 정의하고 설명해 주는 데이터
- 데이터를 빠르게 검색하거나 내용을 간략하고 체계적으로 하기 위해 사용

Smart Data(스마트 데이터)

- 실제로 가치를 창출할 수 있는 검증된 고품질의 데이터

Digital Archiving(디지털 아카이빙)

- 디지털 정보 자원을 장기적으로 보존하기 위한 작업
- 아날로그 콘텐츠는 디지털로 변환한 후 압축해서 저장하고, 디지털 콘텐츠도 체계적으로 분류하고 메타 데이터를 만들어 DB 화하는 작업

Data Diet(데이터 다이어트)

- 데이터를 삭제하는 것이 아니라 압축하고, 중복된 정보는 중복을 배제하고, 새로운 기준에 따라 나누어 저장하는 작업

Map Reduce(맵리듀스)

- 대용량 데이터를 분산 처리하기 위한 목적으로 개발된 프로그래밍 모델이다.
- Google 에 의해 고안된 기술로써 대표적인 대용량 데이터 처리를 위한 병렬 처리 기법을 제공한다.
- 임의의 순서로 정렬된 데이터를 분산 처리하고 이를 다시 합치는 과정을 거친다.

14 용어 - 서비스 공격 관련

서비스 공격 관련

DDoS(분산 서비스 거부, Distributed Denial of Service)

- 여러 곳에 분산된 공격 지점에서 한 곳의 서버에 대해 분산 서비스 공격을 수행하는 공격 방법
- 공격 종류 : **Trinoo, Tribe Flood Network, Stacheldraht**

***DDoS 공격과 연관이 있는 공격 방법은?**

- **Tribe Flood Network**

DoS(서비스 거부, Denial of Service)

- 표적이 되는 서버의 자원을 고갈시킬 목적으로 다수의 공격자 또는 시스템에서 대량의 데이터를 한 곳의 서버에 집중적으로 전송함으로써 표적이 되는 서버의 정상적인 기능을 방해하는 것

Ping Flood

- 특정 사이트에 매우 많은 ICMP Echo 를 보내면, 이에 대한 응답(Respond)을 하기 위해 시스템 자원을 모두 사용해버려 시스템이 정상적으로 동작하지 못하도록 하는 공격방법

SYN Flood

- 막대한 양의 TCP SYN 패킷을 대상 시스템으로 보내서 시스템을 마비 시키는 공격 방법

Ping of Death(죽음의 핑)

- Ping 명령을 전송할 때 패킷의 크기를 인터넷 프로토콜 허용 범위 이상으로 전송하여 공격 대상의 네트워크를 마비시키는 공격 방법

TearDrop

- Offset 값을 변경시켜 수신 측에서 **패킷을 재조립할 때 오류로 인한 과부하**를 발생 시킴

LAND Attack

- 패킷을 전송할 때 송신 IP 주소와 수신 IP 주소를 모두 공격 대상의 IP 주소로 하여 공격 대상 자신에게 전송하는 것으로, **자신에 대해 무한 응답**하게 하는 공격 방법

네트워크 침해 공격 관련

SQL 삽입 공격(SQL Injection)

- SQL Injection 은 임의로 작성한 SQL 구문을 애플리케이션에 삽입하는 공격방식이다.
- SQL Injection 취약점이 발생하는 곳은 주로 웹 애플리케이션과 데이터베이스가 연동되는 부분이다.
- 로그인과 같이 웹에서 사용자의 입력 값을 받아 데이터베이스 SQL 문으로 데이터를 요청하는 경우 SQL Injection 을 수행할 수 있다.
- ~~DBMS 의 종류와 관계없이 SQL Injection 공격 기법은 모두 동일하다.~~

Evil Twin Attack

- 소셜 네트워크에서 악의적인 사용자가 지인 또는 특정 유명인으로 가장하여 활동하는 공격 기법

Phishing(피싱)

- 메일 등으로 공공기관이나 금융기관에서 보낸 것처럼 위장하여 사용자의 개인정보를 빼내는 기법
- 대표적으로 스미싱이 있다.

Smishing(스미싱)

- 문자 메시지(SMS)에 링크를 거는 등 문자 메시지를 이용해 사용자의 개인 신용 정보를 빼내는 수법
- SMS + 피싱 즉 SMS 를 이용하는 피싱 사기

Spear Phishing(스피어 피싱)

- 인간 상호 작용의 깊은 신뢰를 바탕으로 특정 대상을 선정한 후 메일의 링크나 파일을 클릭하도록 유도한 뒤 개인 정보를 탈취하는 수법

Qshing(큐싱)

- QR 코드와 개인정보 및 금융정보를 낚는다(Fishing)의 합성 신조어

지능형 지속 위협(APT; Advanced Persistent Threats)

- 조직적으로 특정 기업이나 조직 네트워크에 침투해 활동 거점을 마련한 뒤 때를 기다리면서 보안을 무력화시키고 정보를 수집한 다음 외부로 빼돌리는 형태의 공격

무차별 대입 공격(Brute Force Attack)

- 암호화된 문서의 암호키를 찾아내기 위해 적용 가능한 모든 값을 대입하여 공격하는 방식

크로스 사이트 스크립팅(XSS; Cross-Site Scripting)

- 웹페이지에 악의적인 스크립트를 포함시켜 사용자 측에서 실행되게 유도함으로써, 정보유출 등의 공격을 유발할 수 있는 취약점

정보보안 침해 공격 관련

Ransomware(랜섬웨어)

- 개인과 기업, 국가적으로 큰 위협이 되고 있는 주요 사이비 범죄 중 하나로 Snake, Darkside 등 시스템을 잠그거나 데이터를 암호화해 사용할 수 없도록 하고 이를 인질로 금전을 요구하는 데 사용되는 악성 프로그램

- 인터넷 사용자의 컴퓨터에 침입해 내부 문서 파일 등을 암호화해 사용자가 열지 못하게 하는 공격으로, 암호 해독용 프로그램의 전달을 조건으로 사용자에게 돈을 요구하기도 한다.

Key Logger Attack(키로거 공격)

- 컴퓨터 사용자의 키보드 움직임을 탐지해 ID, 패스워드 등 개인의 중요한 정보를 몰래 빼가는 해킹 공격

Smurfing(스머핑)

- IP 또는 ICMP 의 특성을 악용하여 엄청난 양의 데이터를 한 사이트에 집중적으로 보냄으로써 네트워크를 불능 상태로 만드는 공격 방법

Zombie(좀비) PC

- 악성코드에 감염되어 다른 프로그램이나 컴퓨터를 조종하도록 만들어진 컴퓨터
- C&C(Command & Control) 서버의 제어를 받아 주로 DdoS 공격 등에 이용됨

C&C 서버

- 해커가 원격지에서 감염된 좀비 PC 에 명령을 내리고 악성코드를 제어하기 위한 용도로 사용하는 서버

Botnet(봇넷)

- 악성 프로그램에 감염되어 악의적인 의도로 사용될 수 있는 다수의 컴퓨터들이 네트워크로 연결된 형태

Worm(웜)

- 네트워크를 통해 연속적으로 자신을 복제하여 시스템의 부하를 높임으로써 결국 시스템을 다운시키는 바이러스의 일종

Zero Day Attack(제로 데이 공격)

- 보안 취약점이 발견되었을 때 발견된 취약점의 존재 자체가 널리 공표되기도 전에 해당 취약점을 통하여 이루어지는 보안 공격

Back Door(백도어, Trap Door)

- 시스템 설계자가 서비스 기술자나 프로그래머의 액세스 편의를 위해 시스템 보안을 제거하여 만들어 놓은 비밀 통로로, 컴퓨터 범죄에 악용되기도 함

Trojan Horse(트로이 목마)

- 정상적인 기능을 하는 프로그램으로 위장하여 프로그램 내에 숨어 있다가 해당 프로그램이 동작할 때 활성화되어 부작용을 일으키는 것으로, 자기 복제 능력은 없음
- 정상적인 기능인 척하는 악성 프로그램

Pharming(파밍)

- 홈페이지 주소를 바꿔 사용자가 진짜 사이트로 오인하게 하여 접속하게 한 다음 개인정보를 탈취하는 기법.

15 용어 – 보안 관련

SSH(Secure Shell)

- SSH 의 기본 네트워크 포트는 22 번을 사용한다

- 전송되는 데이터는 암호화 된다.
- 키를 통한 인증은 클라이언트의 공개키를 서버에 등록해야 한다.
- 서로 연결되어 있는 컴퓨터 간 원격 명령실행이나 셸 서비스 등을 수행한다.

Bell-Lapadula Model

- 군대의 보안 레벨처럼 정보의 기밀성에 따라 상하 관계가 구분된 정보를 보호하기 위해 사용
- 자신의 권한보다 낮은 보안 레벨권한을 가진 경우에는 높은 보안 레벨의 문서를 읽을 수 없고 자신의 권한보다 낮은 수준의 문서만 읽을 수 있다.
- 자신의 권한보다 높은 보안 레벨의 문서에는 쓰기가 가능하지만 보안 레벨이 낮은 문서의 쓰기 권한은 제한한다.

Seven Touchpoints

- 실무적으로 검증된 개발보안 방법론 중 하나로써 SW 보안의 모범 사례를 SDLC(Software Development Life Cycle)에 통합한 소프트웨어 개발 보안 생명주기 방법론

방화벽(Firewall)

- 내부 네트워크에서 외부로 나가는 패킷은 그대로 통과시키고, 외부에서 내부 네트워크로 들어오는 패킷은 내용을 엄밀히 체크하여 인증된 패킷만 통과시키는 구조

웹 방화벽(Web Firewall)

- 일반 방화벽이 탐지하지 못하는 SQL 삽입 공격, XSS(Cross-Site Scripting) 등의 웹 기반 공격을 방어할 목적으로 만들어진 웹 서버에 특화된 방화벽

침입 방지 시스템(IPS; Intrusion Prevention System)

- 방화벽과 침임 탐지 시스템을 결합한 것으로, 비정상적인 트래픽을 능동적으로 차단하고 격리하는 등의 방어 조치를 취하는 보안 솔루션

데이터 유출 방지(DLP; Data Loss Prevention)

- 내부 정보의 외부 유출을 방지하는 보안 솔루션으로, 사내 직원이 사용하는 PC 와 네트워크상의 모든 정보를 검색하고 사용자 행위를 탐지, 통제해 사전 유출 방지

NAC(Network Access Control)

- 네트워크에 접속하는 내부 PC 의 MAC 주소를 IP 관리 시스템에 등록한 후 일관된 보안 관리 기능을 제공하는 보안 솔루션
- 내부 PC 의 소프트웨어 사용 현황을 관리하여 불법적인 소프트웨어 설치를 방지

ESM(Enterprise Security Management)

- 방화벽, IDS, IPS, 웹 방화벽, VPN 등에서 발생한 로그 및 보안 이벤트를 통합하여 관리하는 보안 솔루션
- 보안 솔루션 간의 상호 연동을 통해 종합적인 보완 관리 체계를 수립할 수 있음

SDP(Software Defined Perimeter)

- '블랙 클라우드'라고도 불리며, 2007 년경 GIG 의 네트워크 우선권에 따라 DISA 에서 수행한 작업에서 발전한 컴퓨터 보안 접근 방식

16 용어 - 하드웨어 관련/기타

N-Screen(앤 스크린)

- PC, TV, 휴대폰에서 원하는 콘텐츠를 끊김없이 자유롭게 이용할 수 있는 서비스

NFT (Non-Fungible Token)

- 대체 불가능한 토큰으로, 희소성을 갖는 디지털 자산을 대표하는 토큰
- 블록체인 기술을 활용하지만, 기존의 가상자산과 달리 디지털 자산에 별도의 고유한 인식 값을 부여하고 있어 상호교환이 불가능하다는 특징이 있다.

고가용성(HA; High Availability)

- 긴 시간동안 안정적인 서비스 운영을 위해 장애 발생 시 즉시 다른 시스템으로 대체 가능한 환경을 구축하는 메커니즘
- Hot Standby(상시 대기 방식), Mutual Take-Over(상호 인수), Concurrent Access(동시적 접근)

3D Printing

- 대상을 평면에 출력하는 것이 아니라 손으로 만질 수 있는 실제 물체로 만들어 내는 것

4D Printing

- 특정 시간이나 환경 조건이 갖추어지면 스스로 형태를 변화시키거나 제조되는 자가 조립(Self-Assembly) 기술이 적용된 제품을 3D Printing 하는 기술 의미

RAID(Redundant Array of I Disk)

- 여러 개의 하드디스크로 디스크 배열을 구성하여 파일을 구성하고 있는 데이터 블록들을 서로 다른 디스크들에 분산 저장해 디스크의 속도를 향상시키는 것

4K 해상도

- 차세대 고화질 모니터의 해상도를 지칭하는 용어

Companion Screen(컴패니언 스크린)

- N Screen 의 한 종류로, TV 방송 시청 시 방송 내용을 SNS 를 통해 공유하며 추가적인 기능을 수행할 수 있는 스마트폰, 태블릿 PC 등을 의미

→ 세컨드 스크린(Second Screen) 이라고도 불림

Thin Client PC(신 클라이언트 PC)

- 하드디스크나 주변 장치 없이 기본적인 메모리만 갖추고 서버와 네트워크로 운용되는 개인용 컴퓨터
- 기억장치를 따로 두지 않기 때문에 PC 를 분실하더라도 정보가 유출될 우려가 없음

Phablet(패블릿)

- 폰(Phone)과 태블릿(Tablet)의 합성어로, 태블릿 기능을 포함한 5 인치 이상의 대화면 스마트폰

C 형 USB(Universal Serial Bus Type-C)

- 기존 A 형 USB 에 비하여 크기가 작고, 24 핀으로 위아래의 구분이 없어 어느 방향으로든 연결 가능

MEMS(멤스; Micro-Electro Mechanical Systems)

- 초정밀 반도체 제조 기술을 바탕으로 센서, 액추에이터(Actuator) 등 기계 구조를 다양한 기술로 미세 가공하여 전기계적 동작을 할 수 있도록 한 초미세 장치

Trust-Zone Technology(트러스트존 기술)

- 하나의 프로세서 내에 일반 애플리케이션을 처리하는 일반 구역과 보안이 필요한 애플리케이션을 처리하는 보안 구역으로 분할하여 관리하는 하드웨어 기반의 보안 기술

M-DISC(엠디스크; Millennial DISC)

- 한 번의 기록만으로 자료를 영구 보관할 수 있는 광 저장 장치

Memristor(멤리스터)

- 메모리(Memory)와 레지스터(Register)의 합성어, 전류의 방향과 양 등 기존 경험을 모두 기억하는 특별한 소자
- 전원 공급이 끊어졌을 때도 직전에 통과한 전류의 방향과 양을 기억하기 때문에 다시 전원이 공급되면 기존의 상태가 그대로 복원됨

원 세그(One Seg)

- 일본과 브라질에서 상용 중인 디지털 TV 방송 기술의 일종 (주로 모바일 기기를 대상으로)

PERT

- 프로젝트 일정 관리 기법
- 계획 평가 및 재검토 기술, 프로젝트 관리를 분석하거나 주어진 완성 프로젝트를 포함한 일을 묘사하는데 쓰이는 모델

ZIGBEE

- 저전력 라디오를 이용한 개인 통신망

Cyberbullying

- 사이버 상에서 특정인을 집단적으로 따돌리거나 집요하게 괴롭히는 행위

OTT(오버더탑)

- 개방된 인터넷을 통해 방송프로그램, 영화 등 미디어 콘텐츠를 제공하는 서비스

스택가드(Stack Guard)

- 메모리상에서 프로그램의 복귀 주소와 변수사이에 특정 값을 저장해 두었다가 그 값이 변경되었을 경우 오버플로우 상태로 가정하여 프로그램 실행을 중단하는 기술.

tripwire

- 크래커가 침입하여 백도어를 만들어 놓거나, 설정 파일을 변경했을 때 분석하는 도구

cron

- 작업 예약 스케줄러
- 스케쥴러를 실행시키기 위해 작업이 실행되는 시간, 주기 등 설정하할 때 ()표현식을 통해 배치 수행시간 설정

VAN

- 통신사업자의 회선을 임차하여 단순한 전송 기능 이상의 **부가가치**를 부여한 데이터 등 복합적인 서비스를 제공하는 정보통신망

NS chart (Nassi-Schneiderman chart)

- 논리의 기술에 중점을 둔 도형식 표현 방법이다.
- 연속, 선택 및 다중 선택, 반복 등의 제어논리 구조로 표현한다.
- 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는데 적합하다.
- 논리의 기술에 중점을 두고 도형을 이용한 표현 방법이다.
- 이해하기 쉽고 코드 변환이 용이하다.
- ~~주로 화살표를 사용하여 논리적인 제어구조로 흐름을 표현한다. => 직사각형을 포개어가는 것으로 나타낸다.~~
- ~~화살표나 GOTO 를 사용하여 이해하기 쉽다. => 화살표가 없고 입구와 출구가 하나~~

Wavelength Division Multiplexing★

- 광섬유를 이용한 통신기술의 하나를 의미함
- 파장이 서로 다른 복수의 광신호를 동시에 이용하는 것으로 광섬유를 다중화하는 방식임
- 빛의 파장 축과 파장이 다른 광선은 서로 간섭을 일으키지 않는 성질을 이용함

nmap★

- 서버에 열린 포트 정보를 스캐닝해서 보안취약점을 찾는데 사용하는 도구

PaaS-TA

- 국내 IT 서비스 경쟁력 강화를 목표로 개발되었으며 인프라 제어 및 관리 환경, 실행 환경, 개발 환경, 서비스 환경, 운영환경으로 구성되어 있는 개방형 클라우드 컴퓨팅 플랫폼

VLAN

- 물리적 배치와 상관없이 논리적으로 LAN 을구성하여 Broadcast Domain 을 구분할 수있게 해주는 기술로 접속된 장비들의 성능향상 및 보안성 증대 효과가 있는 것

Software Defined Storage

- IT 스토리지 기술
- 가상화를 적용하여 필요한 공간만큼 나눠 사용할 수 있도록 하며 서버 가상화와 유사함
- '컴퓨팅 소프트웨어로 규정'하는 데이터 스토리지 체계이며, dlfjwd 조직 내 여러 스토리지를 하나처럼 관리하고 운용하는 컴퓨터 이용 환경
- 스토리지 자원을 효율적으로 나누어 쓰는 방법으로 이해할 수 있음

SSO (Single Sign On)

- 시스템이 몇 대가 되어도 하나의 시스템에서 인증에 성공하면 다른 시스템에 대한 접근권한도 얻는 시스템

OWASP

- 오픈소스 웹 애플리케이션 보안 프로젝트로서 주로 웹을 통한 정보 유출, 악성 파일 및 스크립트, 보안 취약점 등을 연구하는 곳

DAS(Direct Attached Storage)

- 하드디스크와 같은 데이터 저장장치를 호스트 버스 어댑터에 직접 연결하는 방식
- 저장장치와 호스트 기기 사이에 네트워크 디바이스가 있지 말아야 하고 직접 연결하는 방식으로 구성

NAS(Network Attached Storage)
- 서버와 저장장치를 네트워크를 통해 연결하는 방식
- 장소에 구애받지 않고 저장장치에 쉽게 접근, 확장성 및 유연성 우수

SAN(Storage Area Network)
- 네트워크상에 광채널 스위치의 이점인 고속 전송과 장거리 연결 및 멀티 프로토콜 기능을 활용
- 각기 다른 운영체제를 가진 여러 기종들이 네트워크 상에서 동일 저장장치의 데이터를 공유하게 함으로써, 여러 개의 저장장치나 백업 장비를 단일화시킨 시스템

IPSec (IP Security)
- ESP 는 발신지 인증, 데이터 무결성, 기밀성 모두를 보장한다.
- 운영 모드는 Tunnel 모드와 Transport 모드로 분류된다.
- AH 는 발신지 호스트를 인증하고, IP 패킷의 무결성을 보장한다.
- ~~암호화 수행시 일방향 암호화만 지원한다. => 해시 암호화~~

17 정보 보안 / 접근 통제

정보 보안의 3 요소

1) **기밀성** : 시스템 내에는 인가된 사용자만 **접근**이 허용. 정보가 전송 중에 노출되더라도 데이터를 읽을 수 없음
2) **무결성** : 시스템 내의 정보는 오직 인가된 사용자만 **수정**할 수 있는 보안 요소
3) **가용성**

정보 보안을 위한 접근 제어(Access Control)
- 적절한 권한을 가진 인가자만 특정 시스템이나 정보에 접근할 수 있도록 통제하는 것이다.
- 시스템 및 네트워크에 대한 접근 제어의 가장 기본적인 수단은 IP 와 서비스 포트로 볼 수 있다.
- 네트워크 장비에서 수행하는 IP 에 대한 접근 제어로는 관리 인터페이스의 접근제어와 List 등이 있다.
- ~~DBMS 에 보안 정책을 적용하는 도구인 8XDMCP 를 통해 데이터베이스에 대한 접근제어를 수행할 수 있다.~~

정보 보안을 위한 접근통제 정책 종류 (MAC/ DAC/ RBAC)

1) 임의 접근 통제 (DAC; Discretionary Access Control)
-데이터에 접근하는 사용자의 신원에 따라 접근 권한 부여
접근통제 권한=주체

2) 강제 접근 통제 (MAC; Mandatory Access Control)
- 정보 시스템 내에서 어떤 주체가 특정 개체에 접근하려 할 때 양쪽의 보안 레이블(Security Label)에 기초하여 높은 보안 수준을 요구하는 정보(객체)가 낮은 보안 수준의 주체에게 노출되지 않도록 하는 접근 제어 방법
접근통제 권한=제 3 자

3) 역할 기반 접근 통제 (RBAC ; Role-based Access Control)

*다음은 정보의 접근통제 정책에 대한 설명이다. (ㄱ)에 들어갈 내용으로 옳은 것은? MAC

정책	(ㄱ)	DAC	RBAC
권한 부여	시스템	데이터 소유자	중앙 관리자
접근 결정	보안등급 (Label)	신분 (Identity)	역할 (Role)
정책 변경	고정적 (변경 어려움)	변경 용이	변경 용이
장점	안정적 중앙 집중적	구현 용이 유연함	관리 용이

21 IDS(침입 탐지 시스템)

침임 탐지 시스템(IDS; Intrusion Detection System)

- 컴퓨터 시스템의 비정상적인 사용, 오용, 남용 등을 실시간으로 **탐지**하는 시스템
- **HIDS**(Host-Based Intrusion Detection)는 운영체제에 설정된 사용자 계정에 따라 어떤 사용자가 어떤 접근을 시도하고 어떤 작업을 했는지에 대한 기록을 남기고 추적한다.
- **NIDS**(Network-Based Intrusion Detection System)로는 대표적으로 Snort 가 있다.
- 외부 인터넷에 서비스를 제공하는 서버가 위치하는 네트워크인 **DMZ**(Demilitarized Zone)에는 IDS 가 설치될 수 있다.
- ~~이상 탐지 기법(Anomaly Detection)은 Signature Base 나 Knowledge Base 라고도 불리며 이미 발견되고 정립된 공격 패턴을 입력해두었다가 탐지 및 차단한다.~~

18 소프트웨어 생명주기 모형

1) 폭포수 모형(Waterfall Model)

- 가장 오래된 모형, 고전적 생명주기 모형, 선형 순차적 모델
- 많은 적용 사례가 있지만 요구사항 변경이 어렵고 각 단계의 결과가 확인되어야 다음 단계로 갈 수 있다.
- 단계별 정의 및 산출물이 명확하지만, 개발 도중의 요구사항의 변경이 어려움
- 타당성 검토, 계획, 요구사항 분석, 구현, 테스트, 유지보수의 단계를 통해 소프트웨어를 개발한다.
- 순차적인 접근방법을 이용한다.
- 단계적 정의와 산출물이 명확하다.
- ~~개발 중 발생한 요구사항을 쉽게 반영할 수 있다.~~

2) 프로토타입 모형(Prototype Model, 원형 모형)

- 견본(시제)품을 만들어 최종 결과물을 예측하는 모형
- 인터페이스 중점을 두어 개발
- 개발 중간에 요구사항의 변경이 쉬움
- **요구사항의 충실 반영**
- **개발 단계 안에서 유지보수가 이루어지는 것으로 볼 수 있다.**
- 발주자나 개발자 모두에게 공동의 참조모델을 제공한다.
- 사용자나 요구사항을 충실히 반영할 수 있다.
- ~~최종 결과물이 만들어지는 소프트웨어 개발 완료시점에 최초로 오류 발견이 가능하다.~~

3) 나선형 모형(Spiral Model, 점진적 모형)

- 폭포수 모형 + 프로토타입 모형 에서 위험 분석 기능 추가
- 점진적 개발 과정 반복으로 요구사항 추가 가능하고, 정밀하고 **유지보수 과정 필요 없음**

- 프로토타입을 지속적으로 발전시켜 최종 소프트웨어 개발까지 이르는 개발방법으로 위험관리가 중심이다.
- 비교적 대규모 시스템에 적합하다.
- **계획 수립 → 위험 분석 → 개발 및 검증 → 고객 평가**
- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것을 목적으로 한다.
- ~~계획, 설계, 개발, 평가의 개발 주기가 한번만 수행된다~~.

***나선형(Spiral) 모형의 주요 태스크가 아닌 것은?**

- 위험 분석
- 개발
- 평가
- ~~버전 관리~~

19 테일러링

소프트웨어 개발 방법론 테일러링의 개요

- 프로젝트 상황 및 특성에 맞도록 정의된 소프트웨어 개발 방법론의 절차, 사용기법 등을 수정 보완하는 작업
- 수행절차 : 프로젝트 특징 정의 → 표준 프로세스 선정 및 검증 → 상위 수준의 커스터마이징 → 세부 커스터마이징 → 테일러링 문서화

테일러링(Tailoring) 개발 방법론

- **내부적 요건 (내부 기준)**
  - 목표 환경
  - 요구사항
  - 프로젝트 규모
  - 보유 기술
  - 납기/비용
  - 기술환경
  - 구성원 능력
- **외부적 요건 (외부 기준)**
  - 법적 제약사항
  - 국제표준 품질기준

24 기타

세션 하이재킹을 탐지하는 방법

- **비동기화 상태 탐지** : 서버와 시퀀스 넘버를 주기적으로 탐지, 비동기 상태 탐지
- **ACK STORM 탐지** : 급격한 ACK 비율 증가시 탐지
- **패킷의 유실 및 재전송 증가 탐지** : 공격자가 중간에 끼어서 작동하므로 패킷의 유실과 서버와의 응답이 길어짐
- ~~FTP SYN SEGNENT 탐지~~

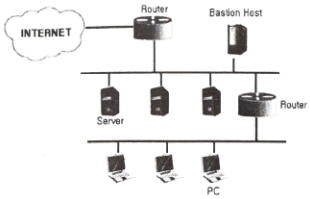
각종 오류

- 생략 오류(omission error) : 입력 시 한 자리를 빼놓고 기록한 경우 (1234 → 123)
- 필사 오류(Transcription error) : 입력 시 임의의 한 자리를 잘못 기록한 경우 (1234 → 1235)

- '12536'으로 기입되어야 하는데 '12936'으로 표기되었을 경우, 어떤 코드 오류에 해당하는가?
- 전위 오류(Transposition error) : 입력 시 좌우 자리를 바꾸어 기록한 경우 (1234 → 1243)
- 이중 오류(Double Transposition error) : 전위 오류가 두 가지 이상 발생한 경우 (1234 → 2143)
- 추가 오류(Addition error) : 입력 시 한 자리 추가로 기록한 경우 (1234 → 12345)
- 임의 오류(Random error) : 위의 오류가 두 가지 이상 결합하여 발생한 경우 (1234 → 12367)

*침입차단 시스템(방화벽) 중 다음과 같은 형태의 구축 유형은? **Screen Subnet**

- 스크린 서브넷(Screen Subnet) : 외부 네트워크와 내부 네트워크 사이에 두는 완충적인 통신망



SDDC(Software-Defined Data Center, 소프트웨어 정의 데이터센터)

- 컴퓨팅, 네트워킹, 스토리지, 관리 등을 모두 소프트웨어로 정의한다.
- 인력 개입 없이 소프트웨어 조작만으로 자동 제어 관리한다.
- 데이터센터 내 모든 자원을 가상화하여 서비스한다.
- ~~특정 하드웨어 에 종속되어 특화된 업무를 서비스하기에 적합하다.~~

***백도어 탐지 방법**

- 무결성 검사
- **열린 포트 확인**
- 로그 분석
- SetUID 파일 검사
- 바이러스 및 백도어 탐지 툴 이용

***합성 중심**

- 전자 칩과 같은 소프트웨어 부품, 즉 블록(모듈)을 만들어서 끼워 맞추는 방법으로 소프트웨어를 완성시키는 재사용 방법(블록 구성 방법)

***생성 중심**

- 추상화 형태로 쓰여진 명세를 구체화하여 프로그램을 만드는 방법 (패턴 구성 방법)

ICMP(Internet Control Message Protocol)

- IP 동작에서 네트워크 진단이나 제어 목적으로 사용

*인터넷 제어 메시지 프로토콜(ICMP)에 관한 설명으로 옳지 않은 것은? 2

1. 에코 메시지는 호스트가 정상적으로 동작하는 지를 결정하는 데 사용할 수 있다.
2. ~~**물리계층 프로토콜이다. => 인터넷계층**~~
3. 메시지 형식은 8 바이트의 헤더와 가변길이의 데이터 영역으로 분리된다.
4. 수신지 도달 불가 메시지는 수신지 또는 서비스에 도달할 수 없는 호스트를 통지하는데 사용된다.

개발보안 방법론

MS-SDL(Microsoft-Secure Development Lifecycle)

- Microsoft 에서 보안수준이 높은 안전한 소프트웨어를 개발하기 위해 자체수립한 SDL
- 방법론이 적용되기 전 버전보다 50% 이상 취약점이 감소함

Seven Touchpoints

- SW 보안의 모범 사례를 SDLC(Software Development Life Cycle)에 통합한 소프트웨어 개발 보안 생명주기 방법론

CLASP(Comprehensive, Lightweight Application Security Process)

- '개념 관점, 역할기반 관점, 활동평가 관점, 활동구현 관점, 취약성 관점'등의 활동중심, 역할 기반의 프로세스로 구성된 집합체로서 이미 운영중인 시스템에 적용하기 적당한 소프트웨어 개발 보안 방법론

CWE(Common Weakness Enumeration)

- 소프트웨어 취약점 및 취약점에 대한 범주 시스템으로, 소프트웨어의 결함을 이해하고 이러한 결함을 식별, 수정 및 방지하는데 사용할 수 있는 자동화된 도구를 작성함

요구사항 분석 자동화 도구

- **SREM** : TRW 사가 우주 국방 시스템 그룹에 의해 실시간 처리 소프트웨어 시스템에서 요구사항을 명확히 기술하도록 할 목적으로 개발한 것으로, RSL 과 REVS 를 사용하는 자동화 도구

PSL/PSA : 미시간 대학에서 개발한 것으로 PSL 과 PSA 를 사용하는 자동화 도구

- **HIPO** : 시스템의 분석 및 설계나 문서화할 때 사용되는 기법으로 시스템 실행 과정의 입력, 처리, 출력의 기능을 나타내고, 종류로는 가시적 도표, 총체적 도표, 세부적 도표가 있음
- **SADT** : SoftTech 사에서 개발된 것으로 구조적 요구분석을 하기 위해 블록 다이어그램을 채택한 자동화 도구★
- **TAGS** : 시스템 공학 방법 응용에 대한 자동 접근 방법으로, 개발 주기의 전 과정에 이용할 수 있음