<4과목 프로그래밍 언어 활용>

1. 꼭 알아야 할 키워드 = \_\_\_\_\_(밑줄)

2. # = 두음 암기 or 한 칸 띄어 쓴 건 산출물

3. 시나공 + 수제비 정리 (페이지 참고)

4. “Ctrl+F” 탐색 → 제목 활용하기

**개발 환경 구축 ★★**

**p.534, 4-2**

**1) 개발 환경 구축의 개요**

- 응용 소프트웨어 개발을 위해 개발 프로젝트를 이해하고 하드웨어 및 소프트웨어 장비를 구축하는 것

- 하드웨어와 소프트웨어의 성능, 편의성, 라이선스 등의 비즈니스 환경에 적합한 제품들을 최종적으로 결정하여 구축함

**2) 하드웨어 환경 ★**

- 사용자와의 인터페이스 역할을 하는 클라이언트(Client)와 클라이언트와 통신하여 서비스를 제공하는 서버(Server)로 구성됨

- **클라이언트**: PC, 스마트폰 등

- **서버**: 웹 서버, 웹 애플리케이션 서버(WAS), 데이터베이스 서버, 파일 서버 등

**▶** **웹 서버**(Web Server) **\_\_ 4-96**

- 클라이언트로부터 직접 요청을 받아 처리하는 서버로, 저용량의 정적 파일들을 제공

# Apache HTTP Server, Microsoft Internet Service, Google Web Server 등

**▶** 웹 애플리케이션 서버(**WAS**; Web Application Server) **★**

- 정적인 콘텐츠를 처리하는 웹 서버(Web Server)와 반대됨

- 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어(=소프트웨어)

- 데이터 접근, 세션 관리, 트랜잭션 관리 등을 위한 라이브러리를 제공

# Apache Tomcat, IBM WebSphere, Oracle WebLogic, JEUS, JBoss, Jetty, Resin 등등

**▶** **데이터베이스 서버**(Database Server) \_\_ **4-98**

- 데이터베이스와 이를 관리하는 DBMS를 운영하는 서버  
# MySQL Server, IBM WebSphere, Oracle WebLogic 등

**▶** **파일 서버**(File Server)

- 데이터베이스에 저장하기에는 비효율적이거나, 서비스 제공을 목적으로 유지하는 파일들을 저장하는 서버

# AWS S3 등

**3) 웹 서버(Web Server)의 기능 \_\_ 4-97**

|  |  |
| --- | --- |
| **기능** | **설명** |
| HTTP/HTTPS 지원 | 브라우저로부터 요청을 받아 응답할 때 사용되는 프로토콜 |
| 통신 기록  (Communication Log) | 처리한 요청들을 로그 파일로 기록하는 기능 |
| 정적 파일 관리 **★**  (Managing Static Files) | HTML, CSS, 이미지 등의 정적 파일들을 저장하고 관리하는 기능 |
| 대역폭 제한 **★**  (Bandwidth Throttling) | 네트워크 트래픽의 포화를 방지하기 위해 응답 속도를 제한하는 기능 |
| 가상 호스팅 **★**  (Virtual Hosting) | 하나의 서버로 여러 개의 도메인 이름을 연결하는 기능 |
| 인증(Authentication) | 사용자가 합법적인 사용자인지를 확인하는 기능 |

**4) 소프트웨어 환경**

- 클라이언트와 서버 운영을 위한 시스템 소프트웨어와 개발에 사용되는 개발 소프트웨어로 구성됨

- **시스템 소프트웨어**: 운영체제(OS), 웹 서버 및 WAS 운용을 위한 서버 프로그램, DBMS

- **개발 소프트웨어**: **요**구사항 관리 도구, **설**계/모델링 도구, **빌**드 도구, **구**현 도구, **테**스트 도구, **형**상 관리 도구 등

#**요설 빌구테형**

**▶** **요**구사항 관리 도구 **★**

- 요구사항의 수집과 분석, 추적 등을 편리하게 도와주는 소프트웨어

# JIRA, IMB DOORS, inteGREAT, Reqtify, Trello 등

**▶** **설**계/모델링 도구

- UML을 지원하며, 개발의 전 과정에서 설계 및 모델링을 도와주는 소프트웨어

# DB Designer, PlantUML, ArgoUML 등

**▶** **빌**드 도구 **★**

- 구현 도구를 통해 작성된 소스의 빌드 및 배포, 라이브러리 관리를 지원하는 소프트웨어

# Ant, Maven, Gradle, Jenkins 등

**▶** **구**현 도구 **★**

- 개발 언어를 통해 애플리케이션의 실제 구현을 지원하는 소프트웨어

# Eclipse, IntelliJ IDEA, Visual Studio, Node.js 등

**▶** **테**스트 도구

- 모듈들이 요구사항에 적합하게 구현됐는지 테스트하는 소프트웨어

# CppUnit, JUnit, HttpUnit, NUnit, SprintTest 등

**▶** **형**상 관리 도구 **★**

- 산출물들을 버전별로 관리하여 품질 향상을 지원하는 스프트웨어

# CVS, SVN(Subversion), GIT 등

**5) 개발 언어의 선정 기준 ★★**

|  |  |
| --- | --- |
| **선정 기준** | **설명** |
| **적**정성 | 개발하려는 소프트웨어의 목적에 적합해야 함 |
| **효**율성 | 코드의 작성 및 구현이 효율적이어야 함 |
| **이**식성 | 다양한 시스템 및 환경에 적용 가능해야 함 |
| **친**밀성 | 개발 언어에 대한 개발자들의 이해도와 활용도가 높아야 함 |
| **범**용성 | 다른 개발 사례가 존재하고 여러 분야에서 활용되고 있어야 함 |

#**적효이친범**

**서버 개발 ★**

**p.537, 4-6**

**1) 서버 개발의 개요**

- 웹 애플리케이션의 로직을 구현할 서버 프로그램을 제작하여 웹 애플리케이션 서버(WAS)에 탑재하는 것을 의미함

- 서버 개발에 사용되는 프로그래밍 언어  
# JAVA, JavaScript, Python, PHP, Ruby등

**2) 서버 개발 프레임워크 ★**

\* **프레임워크(Framework)**: ‘뼈대’, ‘골조’를 의미하는데, 소프트웨어에서는 특정 기능을 수행하기 위해 필요한 클래스나 인터페이스 등을 모아둔 집합체를 뜻함

- 대부분 MVC(Model, View, Controller) 패턴을 기반으로 개발됨

|  |  |
| --- | --- |
| **프레임워크** | **특징** |
| Spring | JAVA 기반으로 만들어진 프레임워크,  전자정부 표준 프레임워크의 기반 기술로 사용됨 **★** |
| Node.js | JavaScript 기반으로 만들어진 프레임워크,  비동기 입, 출력 처리와 이벤트 위주의 높은 처리 성능을 갖고 있어 실시간으로 입, 출력이 빈번한 애플리케이션에 적합함 **★** |
| Django | Python 기반으로 만들어진 프레임워크,  컴포넌트의 재사용과 플러그인화를 강조하여 신속한 개발이 가능하도록 지원함 |
| Codeigniter | PHP 기반으로 만들어진 프레임워크,  인터페이스가 간편하며 서버 자원을 적게 사용함 |
| Ruby on Rails | Ruby 기반으로 만들어진 프레임워크,  테스트를 위한 웹 서버를 지원하며 데이터 베이스 작업을 단순화, 자동화시켜 개발 코드의 길이가 짧아 신속한 개발 가능 |

**3) 서버 프로그램 구현(효과적인 모듈 설계) \_\_ 4-9, 20년 3회 기출문제**

- 응용 소프트웨어와 동일하게 모듈 및 공통 모듈을 개발한 후, 모듈들을 통합하는 방식으로 구현, 유지보수가 용이해야 함

- 모듈의 독립성을 높이려면 모듈의 결합도(Coupling)를 약하게 하고,

응집도(Cohesion)를 강하게 하며 모듈의 크기를 작게 만들어야 함 **★★**

- 복잡도와 중복성을 줄이고 일관성 유지

- 공통 모듈은 여러 프로그램에서 재사용(Reuse) 할 수 있는 모듈을 의미함

→ 함수와 객체 재사용, 컴포넌트 재사용, 애플리케이션 재사용

▶ **재사용 프로그래밍 기법**

- **객**체지향 프로그래밍, **제**네릭 프로그래밍, **자**동 프로그래밍, **메**타 프로그래밍

#**객제자메**

**4) 프레임워크의 특성 ★★**

|  |  |
| --- | --- |
| **특성** | **설명** |
| **모**듈화  (Modularity) | 프레임워크는 캡슐화를 통해 모듈화를 강화하고 설계 및 구현의 변경에 따른 영향을 최소화함으로써 소프트웨어의 품질을 향상시킴 |
| **재**사용성  (Reusability) | 프레임워크는 재사용 가능한 모듈들을 제공함으로써 개발자의 생산성을 향상시킴 |
| **확**장성  (Extensibility) | 프레임워크는 다형성(Polymorphism)을 통한 인터페이스 확장이 가능하여 다양한 형태와 기능을 가진 애플리케이션 개발이 가능함 |
| 제어의 **역**흐름  (Inversion of Control) | 개발자가 관리하고 통제해야 하는 객체들의 제어를 프레임워크가 관리함으로써 생산성 향상시킴 **★** |

#**모재확역**

\

**5) 프레임워크의 구성요소 \_\_ 4-7**

- **개**발환경, **실**행환경, **운**영환경, **관**리환경

#**개실운관**

**보안 및 API ★★**

**p.540, 4-16**

**1) 소프트웨어 개발 보안의 개요**

- 소프트웨어 개발 과정에서 발생할 수 있는 보안 취약점을 최소화하여 보안 위협으로부터 안전한 소프트웨어를 개발하기 위한 일련의 보안 활동을 의미

# 시큐어 코딩(Secure Coding) **★**

- **기**밀성(Confidentiality), **무**결성(Integrity), **가**용성(Availability) 유지하는 것을 목표

#**기무가 ★**

- 정부에서 제공하는 소프트웨어 개발 보안 가이드를 참고하여 소프트웨어 개발 과정에서 점검해야 할 보안 항목들을 점검

**2) 소프트웨어 개발 보안 점검 항목**

|  |  |
| --- | --- |
| **점검 항목** | **설명** |
| **세**션 통제 | 세션의 연결과 연결로 인해 발생하는 정보를 관리하는 것  \* 세션: 서버와 클라이언트의 연결  **※ 보안 약점**: 불충분한 세션 관리, 잘못된 세션에 의한 정보 노출 등 |
| **입**력 데이터 검증 및 표현 | 입력 데이터에 대한 유효성 검증체계를 갖추고, 검증 실패 시 이를 처리할 수 있도록 코딩하는 것 ★  **※ 보안 약점**: SQL 삽입, 경로 조작 및 자원 삽입,  크로스사이트 스크립팅(XSS; Cross-Site Scripting) 등 |
| **보**안 기능 | 인증, 접근제어, 기밀성, 암호화 등의 기능 ★  **※ 보안 약점**: 적절한 인증 없는 중요기능 허용, 부적절한 인가,  사이트 간 요청 위조(CSRF, Cross-Site Request Forgery) 등 |
| **시**간 및 상태 | 동시 수행을 지원하는 병렬 처리 시스템이나 다수의 프로세스가 동작하는 환경에서 시간과 실행 상태를 관리하여 시스템이 원활히 동작되도록 코딩하는 것 ★  **※ 보안 약점**: 종료되지 않는 반복문 또는 재귀함수, 검사 시점과 사용 시점(TOCTOU; Time of Check Time of Use) 경쟁조건 등 |
| **에**러처리 | 소프트웨어 실행 중 발생할 수 있는 오류들을 사전에 정의하여 에러로 인해 발생할 수 있는 문제들을 예방하는 것  **※ 보안 약점**: 오류 메시지를 통한 정보 노출, 오류 상황 대응 부재 등 |
| **코**드 오류 | 개발자들이 코딩 중 실수하기 쉬운 타입 변환, 자원의 반환 등을 고려하며 코딩하는 것 ★  **※ 보안 약점**: 부적절한 자원 해제, 널 포인터(Null Pointer) 역참조 등 |
| **캡**슐화 | 데이터(속성)와 데이터를 처리하는 함수를 하나의 객체로 묶어 코딩하는 것 ★  **※ 보안 약점**: 제거되지 않고 남은 디버그 코드, 잘못된 세션에 의한 데이터 정보 노출 등 |
| **A**PI 오용 | API를 잘못 사용하거나 보안에 취약한 API를 사용하지 않도록 고려하여 코딩하는 것  **※ 보안 약점**: 취약한 API 사용, DNS lookup에 의존한 보안결정 등 |

#**세입보시 에코캡아**

**3) API(Application Programming Interface)** ★ \_\_ 4-18

- 응용 프로그램 개발 시 운영체제나 프로그래밍 언어 등에 있는 라이브러리를 이용할 수 있도록 함으로써 효율적인 소프트웨어 구현을 도와주는 인터페이스

- 개발에 필요한 여러 도구를 제공

- 누구나 무료로 사용할 수 있게 공개된 API를 Open API라고 함

# Windows API, 단일 유닉스 규격(SUS), Java API, 웹 API 등

**배치 프로그램 ★★**

**p.542, 4-21**

**1) 배치 프로그램(Batch Program)의 개요 ★ \_\_ 20년 3회 기출문제**

- 사용자와의 상호 작용 없이 여러 작업들을 미리 정해진 일련의 순서에 따라 일괄적으로 처리하는 것

|  |  |
| --- | --- |
| **종류** | **설명** |
| **이**벤트성 배치 | 특정 조건(이벤트)을 설정해두고 조건이 충족될 때만 수행 |
| **On**-Demand 배치 | 사용자 요청 시 수행 |
| **정**기 배치 | 일, 주, 월과 같이 정해진 기간에 정기 수행 |

#**이온정**

|  |  |
| --- | --- |
| **필수 요소** | **설명** |
| **대**용량 데이터 | 대량의 데이터를 가져오거나, 전달하거나, 계산 등의 처리가 가능해야 함 |
| **자**동화 | 심각한 오류가 발생하는 상황을 제외하고는 사용자의 개입 없이 수행돼야 함 |
| **견**고성 | 잘못된 데이터나 데이터 중복 등의 상황으로 중단되는 일 없이 수행돼야 함 |
| **안**정성/신뢰성 | 오류가 발생하면 오류의 발생 위치, 시간 등을 추적할 수 있어야 함 |
| **성**능 | 다른 응용 프로그램(애플리케이션)의 수행을 방해하지 않아야 하고, 지정된 시간 내에 처리가 완료돼야 함 |

#**대자견안성**

**2) 배치 스케줄러(Batch Scheduler), 잡 스케줄러(Job Scheduler)**

- 일괄 처리 작업이 설정된 주기에 맞춰 자동으로 수행되도록 지원해주는 도구

▶ **스프링 배치(Spring Batch)**: Spring 프레임워크의 특성을 그대로 가져와 스프링이 가지고 있는 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

**패키지 소프트웨어 ★**

**p.544, 4-101**

**1) 패키지 소프트웨어(Package Software)의 개요**

- 기업에서 일반적으로 사용하는 여러 기능들을 통합하여 제공하는 소프트웨어

- 기업에서 패키지 소프트웨어를 구입해 기업 환경에 적합하게 커스터마이징 후 사용

- 기능 요구사항을 70% 이상 충족시키는 패키지 소프트웨어 사용

**2) 패키지 소프트웨어 vs 전용 개발 소프트웨어**

|  |  |  |
| --- | --- | --- |
|  | **패키지 소프트웨어** | **전용 개발 소프트웨어** |
| **기능**  **요구사항** | 70% 이상 충족시키는 패키지 소프트웨어 | 모든 기능 요구사항 반영 가능 |
| **안정성** | 품질이 검증됐고, 업계 표준 준용 | 개발자의 역량에 따라 달라짐 |
| **라이선스** | 판매자 | 회사 |
| **생산성** | 개발을 위한 인력과 시간 절약 가능 | 개발을 위한 인력과 시간 필요 |
| **호환성** | 보장 안됨 | 설계 단계부터 고려하며 개발해서 호환성 좋음 |
| **유지보수** | 결함 발생 시 즉시 대응 어려움 | 결함 발생 시 즉시 대응 가능 |

**데이터 타입 ★★**

**p.552, 4-24, 20년 3회 기출문제**

- 변수(Variable)에 저장될 데이터의 형식을 나타내는 것으로, 변수에 값을 저장하기 전에 문자형, 정수형, 실수형 등 어떤 형식의 값으로 저장할지 선언하는 것

|  |  |  |
| --- | --- | --- |
| **유형** | **기능** | **예** |
| **불**린 타입  (Boolean Type, **bool**) | 조건의 참(True), 거짓(False)여부를 판단하여 저장할 때 사용  → 기본값은 거짓(False)임 | true, false |
| **문**자 타입  (Character Type, **char**) | 한 문자를 저장할 때 사용  → 작은따옴표(‘)안에 표시 | ‘A’, ‘a’, ‘1’, ‘\*’ |
| 문자**열** 타입  (Character String Type, **string**) | 문자열을 저장할 때 사용  → 큰따옴표(“)안에 표시 | “Hello!”, “1+2=3”  숫자도 “ ”안에 있으면 string |
| **정**수 타입  (Integer Type, **int**) | 정수, 소수점이 없는 숫자를 저장할 때 사용 | 1, -1, 10, -100 |
| 부동 **소**수점 타입  (Floating Point Type, **float**) | 실수, 소수점 이하가 있는 숫자를 저장할 때 사용 | 0.123, -1.6 |
| **배**열 타입  (Array Type, **array**) | 같은 타입의 데이터 집합을 만들어 저장할 때 사용  → 데이터는 중괄호{ } 안에 콤마(,)로 구분하여 값들을 나열함 | {1, 2, 3, 4, 5} |

#**불문열 정소배**

▶ **C/C++의 데이터 타입 크기**

|  |  |  |
| --- | --- | --- |
| **종류** | **데이터 타입** | **크기** |
| **문자**/부호없는 문자형 | **char**/unsigned char | 1Byte |
| **정수**/부호없는 정수형 | **short**/unsigned short | 2Byte |
| **int**/unsigned int | 4Byte |
| **long**/unsigned long | 4Byte |
| **long long** | 8Byte |
| **실수** | **float** | 4Byte |
| **double** | 8Byte |
| **long double** | 8Byte |

**변수 ★★★**

**p.555, 4-25**

**1) 변수(Variable)의 개요**

- 컴퓨터가 명령을 처리하는 도중 발생하는 값을 저장하기 위한 공간으로, 변할 수 있는 값을 의미

**2) 변수명 작성 규칙 ★ \_\_ 20년 3회 기출문제**

- 영문자, 숫자, \_(under bar) 사용 가능 **★** →ex) a, A, a1, \_, \_korea (O) / text-color (X)

- 첫 글자는 영문자(대, 소문자), \_(under bar)로 시작할 수 있으나, 숫자는 올 수 없음 **★**

- 글자 수에 제한이 없고, 대, 소문자 구분 → Kim, kim (O)

- 공백이나 \*, +, -, / 등의 특수문자를 사용할 수 없음 **★** →ex) my student, $a, <a (X)

- 예약어를 변수명으로 사용할 수 없음 **★** → if, for, while (X)

- 변수 선언 시 문장 끝에 반드시 세미콜론(;)을 붙여야 함

|  |  |
| --- | --- |
| **변수명** | **설명** |
| 2abc (X) → abc2 (O) | 변수명의 첫 글자를 숫자로 시작해서 변수로 사용할 수 없음 |
| sum\* (X) → sum (O) | 특수문자 ‘\*’를 변수명에 사용할 수 없음 |
| for (X) → For (O) | 예약어를 변수명으로 사용할 수 없음 |
| ha p (X) → ha\_p (O) | 변수명 중간에 공백을 사용할 수 없음 |
| Kim, kim (O) | C언어는 대, 소문자를 구분하기 때문에 Kim, kim은 서로 다른 변수로 사용 가능 |

**3) 예약어 – C언어**

|  |  |  |
| --- | --- | --- |
| **구분** | | **예약어** |
| 제어문 | 반복 | do, for, while |
| 선택 | case, default, else, if, switch |
| 분기 | break, continue, goto, return |
| 자료형 | | char, short, int, long, float, double, enum, signed, unsigned, union, void, struct, typedef |
| 기억 클래스 | | auto, register, static, extern |
| 기타 | | const, sizeof, volatile |

**4) 기억 클래스**

- 변수 선언 시 메모리 내에 변수의 값을 저장하기 위한 기억영역이 할당되는데, 할당 되는 기억영역에 따라 사용 범위에 제한이 있다. 이러한 기억영역을 결정하는 작업을 기억 클래스(Storage Class)라 함

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **종류** | **기억영역** | **예약어** | **생존기간** | **사용 범위** |
| **자**동 변수 | 메모리(스택) | auto | 일시적 | 지역적 |
| **레**지스터 변수 | 레지스터 | register |
| **정**적 변수(내부) | 메모리(데이터) | static | 영구적 |
| 정적 변수(외부) | 전역적 |
| **외**부 변수 | extern |

#**자레정외**

**5) 변수의 선언**

|  |  |
| --- | --- |
| **선언** | **설명** |
| 자료형 변수명 = 값; | ▶ **자료형**: 변수에 저장될 자료의 형식 지정  ▶ **변수명**: 사용자가 원하는 이름을 임의로 지정 (변수명 작성 규칙 준수)  ▶ **값**: 변수를 선언하면서 초기화할 값을 지정 |

**연산자 ★★★**

**p.560, 4-26**

#**산시관비논**

**1) 산술 연산자 ★**

|  |  |
| --- | --- |
| **연산자** | **의미** |
| + | 덧셈 |
| - | 뺄셈 |
| \* | 곱하기 |
| \*\* | 제곱 |
| / | 나누기 |
| // | 나누기 연산 후 소수점 이하의 수를 버리고, 정수 부분의 몫을 구함 |
| % | 나누기 연산 후 몫이 아닌 나머지를 구함 |
| ++ | 증감 연산자 (전치; 먼저 변수의 값을 증감시킨 후 연산에 사용) |
| -- | 감소 연산자 (후치; 먼저 변수를 연산에 사용한 후 값을 증감시킴) |

ex) ++a, --a (연산 전 ↑, ↓)

ex) a++, a-- (연산 후 ↑, ↓)

**2) 시프트 연산자**

|  |  |  |
| --- | --- | --- |
| **연산자** | **의미** | **비고** |
| << | 왼쪽 시프트 | 비트를 왼쪽으로 이동  ex) 00101 → 01010 |
| >> | 오른쪽 시프트 | 비트를 오른쪽으로 이동  ex) 00101 → 00010 |

**3) 관계 연산자 ★**

|  |  |
| --- | --- |
| **연산자** | **의미** |
| == | 같다 |
| != | 같지 않다 |
| > | 크다 |
| >= | 크거나 같다 |
| < | 작다 |
| <= | 작거나 같다 |

**4) 비트 연산자 ★ \_\_ 20년 1, 2회 기출문제**

- 비트별(0, 1)로 연산해 결과를 얻는 연산자

|  |  |  |
| --- | --- | --- |
| **연산자** | **의미** | **비고** |
| & | and | 모든 비트가 1일 때만 1 |
| | | or | 모든 비트 중 한 비트라도 1이면 1 |
| ^ | xor | 모든 비트가 같으면 0, 하나라도 다르면 1 |
| ~ | not | 각 비트의 부정, 0이면 1, 1이면 0 |

**5) 논리 연산자 ★**

- 두 개의 논리 값을 연산하여 참(true, 1) 또는 거짓(false, 0)을 결과로 얻는 연산자

|  |  |  |
| --- | --- | --- |
| **연산자** | **의미** | **비고** |
| && | and | 모두 참(1)이면 참(1) |
| || | or | 하나라도 참(1)이면 참(1) |
| ! | not | 부정 |

#**조대순**

**6) 조건 연산자(삼항 연산자) \_\_ 20년 3회 기출문제**

- 조건에 따라 서로 다른 수식 수행 / **if(?),** **else(:)**

# 조건 수식 **?** 수식1 **:** 수식2;

→ ‘조건 수식’이 **참**(true, 1)이면 ‘**수식1**’ 수행, **거짓**(false, 0)이면 ‘**수식2**’ 수행

**7) 대입 연산자**

|  |  |  |
| --- | --- | --- |
| **연산자** | **예** | **의미** |
| += | a += 1 | a = a+1 |
| -= | a -= 1 | a = a-1 |
| \*= | a \*= 1 | a = a\*1 |
| /= | a /= 1 | a = a/1 |
| %= | a %= 1 | a = a%1 |
| <<= | a <<= 1 | a = a<<1 |
| >>= | a >>= 1 | a = a>>1 |

**8) 기타 연산자**

|  |  |
| --- | --- |
| **연산자** | **의미** |
| sizeof | 자료형의 크기 표시 |
| ,(콤마 comma) | 왼쪽에서 오른쪽으로 순서대로 수행되며, 순서를 콤마로 구분  **순**서 연산자 |
| (자료형) | 사용자가 자료형을 다른 자료형으로 변환할 때 사용하는 것  **캐스트(cast) 연산자**  ex) a = (int)1.3 + (int)1.4; → a = 2 |

**9) 연산자 우선순위 ★**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **대분류** | **중분류** | **연산자** | **결합규칙** | **우선 순위** |
| 단항 연산자 | 단항 연산자 | !(논리 not) ~(비트 not)  ++ -- sizeof | ← | **높음**  ↑  ↓  **낮음** |
| 이항 연산자 | **산**술 연산자 | \* / % | → |
| + - |
| **시**프트 연산자 | << >> |
| **관**계 연산자 | < <= >= > |
| == != |
| **비**트 연산자 | &  ^  | |
| **논**리 연산자 | &&  || |
| 삼항 연산자 | **조**건 연산자 | ? : | → |
| 대입 연산자 | **대**입 연산자 | = += -= \*= /= %= <<= >>= | ← |
| 순서 연산자 | **순**서 연산자 | , | → |

**#산시관비논 조대순**

**제어문, 반복문 ★★**

**p.572~579, 4-27**

**1) 제어문의 개념**

- 컴퓨터 프로그램은 명령어가 서술된 순서에 따라 무조건 위에서 아래로 실행되는데, 조건을 지정해서 진행 순서를 변경할 수 있다. 이렇게 프로그램의 순서를 변경할 때 사용하는 명령문을 제어문이라고 함

**2) 단순 if문**

▶ 형식 1: 조건이 참일 때만 실행

-조건이 참일 때 실행할 문장이 하나인 경우

if(조건)

실행할 문장;

-조건이 참일 때 실행할 문장이 두 문장 이상인 경우

if(조건)

{

실행할 문장1; // 실행할 문장이 두 문장 이상이면 { } 중괄호로 감싸기

실행할 문장2;

…

}

▶ 형식 2: 조건이 참일 때와 거짓일 때 실행할 문장이 다름

if(조건)

실행할 문장1; // 조건이 **참**일 경우 실행

else

실행할 문장2; // 조건이 **거짓**일 경우 실행

**3) 다중 if문 ★**

▶ 형식 1: 조건이 여러 개일 때 사용

if(조건1)

실행할 문장1; // 조건1이 참일 경우 실행

else if(조건2)

실행할 문장2; // 조건2가 참일 경우 실행

else if(조건3)

실행할 문장3; // 조건3이 참일 경우 실행

…

else

실행할 문장4; // 앞의 조건이 모두 거짓일 경우 실행

▶ 형식 2: if문 안에 if문이 포함됨

if(조건1)

{ // 조건1이 참일 경우 실행

if(조건2)

실행할 문장1; // 조건2가 참일 경우 실행

else(조건2)

실행할 문장2; // 조건2가 거짓일 경우 실행

}

else

실행할 문장3; // 조건1이 거짓일 경우 실행

**4) switch문 ★★**

▶ 조건에 따라 분기할 곳이 여러 곳인 경우 간단하게 처리할 수 있는 제어문

switch(수식)

{

case 레이블1:

실행할 문장1; // 수식의 결과가 레이블 1과 일치할 때 실행

break; // switch문 종료함

case 레이블 2:

실행할 문장2; // 수식의 결과가 레이블 2와 일치할 때 실행

break; // switch문 종료함

…

default: // 수식의 결과가 레이블 1~2와 일치하지 않을 때,

실행할 문장3; 혹은 break가 없을 땐 무조건 default 실행

}

**ex)**

switch(2)

{ case 3: printf(“1”); // 수식의 결과(2)가 레이블 1(3)과 일치하지 않음

break;

case 2: printf(“2”); // 수식의 결과(2)가 레이블 2(2)와 일치함 → printf(“2”); 실행

break; // switch문 종료함

case 1: printf(“3”); // 수식의 결과(2)가 레이블 3(1)과 일치하지 않음

break;

} → 결과 2 표시됨

**5) 반복문의 개념**

- 제어문의 한 종류로 일정한 횟수를 반복하는 명령문을 말한다. 보통 변수의 값을 일정하게 증가시키면서 정해진 수가 될 때까지 명령이나 명령 그룹을 반복 수행함

**6) for문**

- 초기값, 최종값, 증가값을 지정하는 수식을 이용해 정해진 횟수를 반복하는 제어문

- 초기값을 정한 다음 최종값에 대한 조건이 참이면 실행할 문장을 실행한 후 초기값을 증가값 만큼 증가시키면서 최종값에 대한 조건이 참인 동안 실행할 문장을 반복 수행함

▶ 형식

for(초기값; 최종값; 증가값)

실행할 문장; // 최종값이 참인 동안 실행, 두 문장 이상 → { } 입력

**7) while문**

- 조건이 참인 동안 실행할 문장을 반복 수행하는 제어문

- 조건이 참인 동안 실행할 문장을 반복 수행하다가 조건이 거짓이면 while문을 끝낸 후 다음 코드를 실행

- 조건이 처음부터 거짓이면 한 번도 실행하지 않음 **★**

▶ 형식

while(조건)

실행할 문장; // 조건이 참인 동안 실행, 두 문장 이상 → { } 입력

**8) do ~ while문 ★**

- 조건이 참인 동안 정해진 문장을 반복 수행하다가 조건이 거짓이면 반복문을 벗어나는 while문과 같은 동작을 함

- 그러나 조건이 처음부터 거짓이어도 실행할 문장을 무조건 한 번 실행함**,** 그리고 다음 조건을 판단하여 탈출 여부를 결정함

▶ 형식

do

실행할 문장; // 조건이 참인 동안 실행, 두 문장 이상 → { } 입력

while(조건);

**9) break, continue**

- switch문이나 반복문의 실행을 제어하기 위해 사용되는 예약어

▶ break **★**

-switch문이나 반복문 안에서 break가 나오면 블록을 벗어남

▶ continue

-continue 이후의 문장을 실행하지 않고 제어를 반복문의 처음으로 옮김

-반복문에서만 사용됨

**배열과 문자열 ★★**

**p.585**

**1) 배열의 개념**

- 동일한 데이터 유형을 여러 개 사용해야 할 경우 이를 손쉽게 처리하기 위해 여러 개의 변수들을 조합해서 하나의 이름으로 정의해 사용하는 것

- 개별적인 요소들의 위치는 첨자를 이용하여 지정

- 변수명 뒤에 대괄호 [ ]를 붙이고 그 안에 사용할 개수를 지정

- C언어에서 배열의 위치는 0부터 시작됨

**2) 1차원 배열**

- 1차원 배열은 변수들을 일직선상의 개념으로 조합한 배열

|  |  |
| --- | --- |
| **선언** | **설명** |
| **자료형 변수명[개수];** | ▶ **자료형**: 배열에 저장할 자료의 형 지정  ▶ **변수명**: 사용할 배열의 이름으로 사용자가 임의로 지정  ▶ **개수**: 배열의 크기를 지정하는 것으로 생략할 수 있음 |

첫 번째 두 번째 세 번째 네 번째 다섯 번째

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a[0] | a[1] | a[2] | a[3] | a[4] |

# a[3]: a는 배열의 이름이고, 3은 첨자로 배열 a에서의 위치를 나타냄

a[3]에 4를 저장시키려면 ‘a[3]=4’와 같이 작성

**3) 2차원 배열 ★**

- 2차원 배열은 변수들을 평면, 즉 행과 열로 조합한 배열

|  |  |
| --- | --- |
| **선언** | **설명** |
| **자료형 변수명[행개수][열개수];** | ▶ **자료형**: 배열에 저장할 자료의 형 지정  ▶ **변수명**: 사용할 배열의 이름으로 임의로 지정  ▶ **행개수**: 배열의 행 크기를 지정  ▶ **열개수**: 배열의 열 크기를 지정 |

첫 번째 두 번째 세 번째 네 번째 다섯 번째

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0, 0 | 0, 1 | 0, 2 | 0, 3 | 0, 4 |
| 1, 0 | 1, 1 | 1, 2 | 1, 3 | 1, 4 |
| 2, 0 | 2, 1 | 2, 2 | 2, 3 | 2, 4 |

# b[0][2]: b는 배열의 이름이고, 0은 행 첨자, 2는 열 첨자로서 배열 b의 위치를 나타냄

**4) 배열의 초기화**

- 배열 선언 시 초기값을 정할 수 있음

▶ 1차원 배열 초기화

char a[3] = {‘A’, ‘B’, ‘C’} / char a[ ] = {‘A’, ‘B’, ‘C’}

|  |  |  |
| --- | --- | --- |
| A | B | C |

a[0] a[1] a[2]

▶ 2차원 배열 초기화

int a[2][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} }; / = { 10, 20, 30, 40, 50, 60, 70, 80};

a[0][0] a[0][1] a[0][2] a[0][3]

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 20 | 30 | 40 |
| 50 | 60 | 70 | 80 |

a[1][0] a[1][1] a[1][2] a[1][3]

**5) 배열 형태의 문자열 변수**

→ char 배열이름[크기] = “문자열”

ex) char a[5] = “love”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| l | o | v | e | \0 |

\*문자열의 끝을 알리기 위한 널 문자(‘\0’)이 자동 삽입됨

**포인터 ★**

**p.594**

**1) 포인터와 포인터 변수**

- 포인터는 변수의 주소를 말하며, C언어에서는 주소를 제어할 수 있는 기능을 제공함

- C언어에서 변수의 주소를 저장할 때 사용하는 변수를 포인터 변수라고 함

- 포인터 변수는 필요에 의해 동적으로 할당되는 메모리 영역인 힙 영역에 접근하는 동적 변수임

▶ 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;

**2) 포인터와 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 b에 저장함

b = &a[0] → a 배열의 첫 번째 요소인 a[0]의 주소(&)를 b에 저장함

**절차적 프로그래밍 언어 ★★**

**p.600, 4-29**

**1) 절차적 프로그래밍 언어의 개요**

- 일련의 처리 절차를 정해진 문법에 따라 순서대로 기술해나가는 언어

- 프로그램이 실행되는 절차(Procedure)를 중요시 함

**2) 절차적 프로그래밍 언어의 장, 단점**

- 컴퓨터의 처리 구조와 유사하여 실행 속도가 빠름

- 같은 코드를 복사하지 않고 다른 위치에서 호출하여 사용할 수 있음(이식성🡩)

- 모듈 구성이 용이하며, 구조적인 프로그래밍이 가능함

- 프로그램을 분석하기 어렵고, 유지 보수나 코드의 수정이 어려움

**3) 절차적 프로그래밍 언어의 종류 ★**

|  |  |
| --- | --- |
| **언어** | **특징** |
| **C** | - 1972년 미국 벨 연구소의 데니스 리치에 의해 개발됨  - 시스템 프로그래밍 언어로 널리 사용됨  - 자료의 주소를 조작할 수 있는 포인터 제공  - 고급 프로그래밍 언어, 저급 프로그래밍 언어의 특징을 모두 갖춤  - UNIX의 일부가 C언어로 구현됨  - 컴파일러 방식의 언어  - 이식성이 좋아 컴퓨터 기종에 관계없이 프로그램 작성 가능 |
| **Algol**  (알골) | - 수치계산이나 논리 연산을 위한 과학 기술 계산용 언어  - PASCAL과 C언어의 모체 |
| **Cobol**  (코볼) | - 사무 처리용 언어  - 영어 문장 형식으로 구성되어 있어 이해와 사용이 쉬움  - 4개의 DIVISION으로 구성됨 |
| **Fortran**  (포트란) | - 과학 기술 계산용 언어  - 수학과 공학 분야의 공식이나 수식과 같은 형태로 프로그래밍 가능 |
| **Basic**  (베이직) | - 교육용으로 개발되어 언어의 문법이 쉬움  - 다양한 종류가 존재하며 서로 다른 종류 사이의 소스 코드는 호환되지 않음 |

**객체지향 프로그래밍 언어 ★★**

**p.602, 4-30**

**1) 객체지향 프로그래밍 언어의 개요**

- 현실 세계의 개체(Entity)를 기계의 부품처럼 하나의 객체로 만들어, 기계적인 부품들을 조립하여 제품을 만들 듯이 소프트웨어를 개발할 때도 객체들을 조립해서 프로그램을 작성할 수 있도록 한 프로그래밍 기법

**2) 객체지향 프로그래밍 언어의 장, 단점**

- 상속을 통한 재사용과 시스템의 확장이 용이하고, 코드의 재활용성이 높음

- 사용자와 개발자 사이의 이해를 쉽게 해주고, 대형 프로그램의 작성이 용이함

- 프로그래밍 구현을 지원해 주는 정형화된 분석 및 설계 방법이 없음

- 구현 시 처리 시간이 지연됨 → 실행 속도가 느림

**3) 객체지향 프로그래밍 언어의 종류 ★**

|  |  |
| --- | --- |
| **언어** | **특징** |
| **C++** | - C언어에 객체지향 개념을 적용한 언어  - 모든 문제를 객체로 모델링하여 표현함 |
| **C#** | - Microsoft에서 개발한 객체지향 프로그래밍 언어  - C++과 JAVA의 문법과 비슷함  - JAVA와 달리 불안전 코드(Unsafe Code)와 같은 기술을 통해  플랫폼 간 상호 운용성 확보 |
| **JAVA**  (자바) | - 분산 네트워크 환경에 적용이 가능하며, 멀티스레드 기능을 제공하므로 여러 작업을 동시에 처리할 수 있음  - 운영체제 및 하드웨어에 독립적이며, 이식성이 강함  - 캡슐화가 가능하고 재사용성이 높음 |
| **Delphi**  (델파이) | - 기본적인 문법은 파스칼 문법에 여러 기능들이 추가되어 존재  - Windows 운영체제에서 모든 부분을 프로그래밍할 수 있는 언어  - 높은 생산성과 간결한 코드가 대표적인 장점 |
| **Smalltalk** | - 1세대 객체지향 프로그래밍 언어  - 최초로 GUI를 제공한 언어 |

**4) 객체지향 프로그래밍 언어의 구성 요소 ★**

**▶ 객체(Object)**

- 독립적으로 식별 가능한 이름을 갖고 있음

- 객체가 가질 수 있는 조건인 상태(State)는 일반적으로 시간에 따라 변함

- 객체와 객체는 상호 연관성에 의한 관계가 형성됨

- 객체가 반응할 수 있는 메시지의 집합을 행위(연산, Method)라고 하며, 객체는 행위의 특징을 나타냄

- 객체는 일정한 기억장소를 갖고 있음

**▶ 클래스(Class)**

- 공통된 속성과 연산(행위)를 갖는 객체의 집합

- 객체지향 프로그램에서 데이터를 추상화하는 단위 ★

- 각각의 객체들이 갖는 속성과 연산(Method)을 정의하고 있는 틀

- 슈퍼 클래스(Super Class)는 특정 클래스의 상위(부모) 클래스

- 서브 클래스(Sub Class)는 특정 클래스의 하위(자식) 클래스

**▶ 인스턴스(Instance)**

- 클래스에 속한 각각의 객체

- 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화(Instantiation)라고 함

**▶ 메서드(Method)**

- 클래스로부터 생성된 객체를 사용하는 방법

- 전통적 시스템의 함수(Function) 또는 프로시저(Procedure)에 해당하는 연산

**▶ 메시지(Message)**

- 객체에게 어떤 행위를 하도록 지시하기 위한 방법

**5) 객체지향 프로그래밍 언어의 특징 ★★**

|  |  |
| --- | --- |
| **특징** | **설명** |
| **캡**슐화  (Encapsulation) **★** | ▶ 데이터(속성)와 데이터를 처리하는 함수를 하나로 묶는 것  - 캡슐화된 객체의 세부 내용이 외부에 은폐(정보 은닉)되어, 변경이 발생할 때 오류의 파급효과가 적음  - 캡슐화된 객체들은 재사용이 용이함 |
| **상**속성  (Inheritance) | 이미 정의된 상위 클래스(부모 클래스)의 모든 속성과 연산을 하위 클래스(자식 클래스)가 물려받는 것 |
| **다**형성  (Polymorphism) **★** | 메시지에 의해 객체(클래스)가 연산을 수행하게 될 때, 하나의 메시지에 대해 각 객체(클래스)가 가지고 있는 고유의 방법(특성)으로 응답할 수 있는 능력 |
| **추**상화  (Abstraction) | ▶ 불필요한 부분은 생략하고 객체의 속성 중 가장 중요한 것에만 중점을 두어 개략화, 모델화 하는 것  - 데이터의 공통된 성질을 추출하여 슈퍼 클래스를 선정하는 개념 |
| **정**보 은닉  (Information Hiding) | 캡슐화에서 가장 중요한 개념으로, 다른 객체에게 자신의 정보를 숨기고 자신의 연산만을 통하여 접근을 허용하는 것 |

#**캡상다추정**

**스크립트 언어 ★★**

**p.605, 4-31, 20년 1, 2회 기출문제**

**1) 스크립트 언어의 개요**

- HTML 문서 안에 직접 프로그래밍 언어를 삽입하여 사용하는 것으로, 기계어로 컴파일 되지 않고 별도의 번역기가 소스를 분석하여 동작하게 하는 언어

- 클라이언트의 웹 브라우저에서 해석되어 실행되는 클라이언트용 언어와, 서버에서 해석되어 실행된 후 결과만 클라이언트로 보내는 서버용 스크립트 언어

# **클라이언트용 언어**: JavaScript / **서버용 스크립트 언어**: ASP, JSP, PHP, Python

**2) 스크립트 언어의 장, 단점**

- 컴파일 없이 바로 실행하므로 결과를 바로 확인할 수 있음 **★**

- 개발 시간이 짧고, 소스 코드를 쉽고 빠르게 수정할 수 있음

- 코드를 읽고 해석해야 하므로 실행 속도가 느리고, 런타임 오류가 많이 발생함

**3) 스크립트 언어의 종류 ★**

|  |  |
| --- | --- |
| **언어** | **특징** |
| **JavaScript**  (자바스크립트) | 웹페이지의 동작을 제어하는 데 사용되는 클라이언트용 스크립트 언어로, 클래스가 존재하지 않으며 변수 선언도 필요 없음 **★** |
| **ASP**  (Active Server Page) | 서버 측에서 동적으로 수행되는 페이지를 만들기 위한 언어, Microsoft 제작 → Windows계열에서만 수행 가능 |
| **JSP**  (Java Server Page) | - JAVA로 만들어진 서버용 스크립트  - 다양한 운영체제에서 사용 가능 |
| **PHP**  (Professional Hypertext Preprocessor) | - 서버용 스크립트 언어로 C, JAVA 등과 문법이 유사함  - LINUX, UNIX, Windows 운영체제에서 사용 가능 |
| **Python**  (파이썬) | - 다양한 플랫폼에서 쓸 수 있고, 라이브러리(모듈) 풍부  - 유니 코드 문자열을 지원하여 다양한 언어의 문자 처리  - 들여쓰기를 사용하여 블록을 구분하는 문법 채용 **★**  - 다른 언어의 모듈들을 연결하는 대화형 인터프리터 언어 |
| **Perl**(펄) | 인터프리터 방식의 프로그래밍 언어 |

**선언형 언어 ★**

**p.607, 4-32**

**1) 선언형 언어의 개요**

- 명령형 언어가 문제를 해결하기 위한 방법을 기술한다면, 선언형 언어는 프로그램이 수행해야 하는 문제를 기술하는 언어

- 함수형 언어(적용형 언어)와 논리형 언어(선언적 언어)가 있음

- 목표를 명시하고 알고리즘은 명시하지 않음

cf) **명령형 언어**(절차적 언어, 객체지향 언어)

-알고리즘을 명시하고 목표는 명시하지 않음

-폰노이만 구조에 개념적인 기초를 두고 있음

-특정 구문의 연산을 이용하여 상태를 변경시키고 프로그램을 동작시킴

**2) 선언형 언어의 장, 단점**

- 가독성이나 재사용성이 좋고, 오류가 적음

- 프로그램 동작을 변경하지 않고도 관련 값을 대체할 수 있음

**3) 선언형 언어의 종류**

|  |  |
| --- | --- |
| **종류** | **특징** |
| **Haskell**  (하스켈) | - 함수형 프로그래밍 언어, 부작용이 없음  - 패턴 맞춤, 커링, 조건제시법, 가드, 연산자 정의 등 기능 존재 |
| **LISP**  (리스프) | - 함수형 프로그래밍 언어, 수학 표기법을 나타내기 위한 목적  - 함수 호출 시 함수 이름 혹은 연산자가 첫 번째로 위치하고 피연산자가 이어서 위치 **★** |
| **PROLOG**  (프롤로그) | - 논리식 기반으로 객체 간의 관계에 관한 문제를 해결하기 위해 사용  - 인공지능이나 계산 언어학 분야, 자연언어 처리 분야에서 사용 |
| **HTML** | 인터넷의 표준 문서인 하이퍼텍스트 문서를 만들기 위해 사용하는 언어 |
| **XML** | 기존 HTML의 단점을 보완해 웹에서 구조화된 폭 넓고 다양한 문서들을 상호 교환할 수 있도록 설계된 언어 |

**라이브러리 ★**

**p.609, 4-10, 4-36**

**1) 라이브러리(Library)의 개념**

- 프로그램을 효율적으로 개발할 수 있도록 자주 사용하는 함수나 데이터들을 미리 만들어 모아 놓은 집합체로 필요할 때 언제든지 호출하여 사용할 수 있음

▶ **표준 라이브러리**: 프로그래밍 언어에 기본적으로 포함되어 있는 라이브러리

▶ **외부 라이브러리**: 개발자들이 필요한 기능을 만들어 인터넷 등에 공유해 놓을 것으로, 다운받아 설치한 후 사용함

**2) C언어의 대표적인 표준 라이브러리**

|  |  |
| --- | --- |
| **헤더 파일** | **기능** |
| stdio.h | 데이터의 입, 출력(in, out)에 사용되는 기능 제공 |
| math.h | 수학 함수 제공 |
| string.h | 문자열 처리에 사용되는 기능 제공 |
| stdlib.h | 자료형 변환, 난수 발생, 메모리 할당에 사용되는 기능들 제공 |
| time.h | 시간 처리에 사용되는 기능 제공 |

**3) JAVA의 대표적인 표준 라이브러리**

- JAVA에서 패키지를 사용하려면 ‘import java.util’과 같이 import문을 이용해 선언한 후 사용해야 함

|  |  |
| --- | --- |
| **패키지** | **기능** |
| java.lang | 자바에 기본적으로 필요한 인터페이스, 자료형, 예외 처리 등에 관련된 기능 제공, import문 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 |

**데이터 입, 출력 ★★**

**p.611, 4-37**

**1) scanf( ) 함수**

- C언어의 표준 입력 함수, 키보드로 입력받아 변수에 저장하는 함수

|  |  |
| --- | --- |
| **형식** | **설명** |
| **scanf(서식 문자열, 변수의 주소);** | ▶ **서식 문자열**: 입력 받을 데이터의 자료형 지정  ▶ **변수의 주소**: 데이터를 입력 받을 변수를 적음,  변수의 주소로 입력 받아야 하기 때문에  변수에 주소 연산자 &를 붙임 |

ex) scanf(“%3d”, &a); → %: 서식 문자 / 3: 입력 자릿수 / d: 10진수 / &a: 변수 a의 주소

# 서식 문자열과 변수의 자료형은 일치해야 함, 한 번에 여러 개의 데이터 입력 가능

▶ **서식 문자열**

|  |  |
| --- | --- |
| **종류** | **의미** |
| %d | 정수형 10진수, decimal **★** |
| %u | 부호없는 정수형 10진수 |
| %o | 정수형 8진수 |
| %x | 정수형 16진수 |
| %c | 문자, character **★** |
| %s | 문자열, string **★** |
| %f | 소수점을 포함하는 실수, float **★** |
| %e | 지수형 실수 |
| %ld | long형 10진수 |
| %lo | long형 8진수 |
| %lx | long형 16진수 |
| %p | 주소 16진수 |

**2) printf( ) 함수**

- C언어의 표준 출력 함수, 인수로 주어진 값을 화면에 출력하는 함수

|  |  |
| --- | --- |
| **형식** | **설명** |
| **printf(서식 문자열, 변수);** | ▶ **서식 문자열**: 변수의 자료형에 맞게 지정  ▶ **변수**: 서식 문자열의 순서에 맞게 출력할 변수,  scanf( )와 달리 주소 연산자 &를 붙이지 않음 |

ex) printf(“%-8.2f”, 200.2); → 200.20VV(V는 빈 칸을 의미함)

%: 서식 문자 / -: 왼쪽부터 출력 / 8: 출력 자릿수 / 2: 소수점 이하 자리 / f: 실수 출력

▶ **주요 제어문자**

|  |  |  |
| --- | --- | --- |
| **문자** | **의미** | **기능** |
| \n | new line | 커서를 다음 줄 앞으로 이동 **★** |
| \b | backspace | 커서를 왼쪽으로 한 칸 이동 |
| \t | tab | 커서를 일정 간격 띄움 |
| \r | carriage return | 커서를 현재 줄 처음 이동 |
| \0 | null | 널 문자 출력 |
| \’ | single quote | 작은따옴표 출력 |
| \” | double quote | 큰따옴표 출력 |
| \a | alert | 스피커로 벨 소리 출력 |
| \\ | backslash | 역 슬래시 출력 |
| \f | form feed | 한 페이지 넘김 |

**3) 기타 표준 입, 출력 함수 ★**

|  |  |  |
| --- | --- | --- |
| **입력**  **(get)** | getchar( ) | 키보드로 한 문자를 입력받아 변수에 저장하는 함수 |
| gets( ) | 키보드로 문자열을 입력받아 변수에 저장하는 함수 |
| **출력**  **(put)** | putchar( ) | 인수로 주어진 한 문자를 화면에 출력하는 함수 |
| puts( ) | 인수로 주어진 문자열을 화면에 출력한 후,  커서를 자동으로 다음 줄 앞으로 이동하는 함수 |

**운영체제의 개념 ★★★**

**p.632~637, 4-40**

**1) 운영체제(OS; Operating System)의 정의**

- 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효과적으로 사용할 수 있도록 환경을 제공하는 여러 프로그램의 모임

# 사용자 > 응용 프로그램 > 유틸리티 > **운영체제(OS)** > 하드웨어

**2) 운영체제의 목적 ★**

|  |  |
| --- | --- |
| **목적** | **설명** |
| **처**리 능력  (Throughput) | 일정 시간 내에 시스템이 처리하는 일의 양 |
| **반**환 시간  (Turn Around Time) | 시스템에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간 |
| **사**용 가능도  (Availability) | 시스템을 사용할 필요가 있을 때 즉시 사용 가능한 정도 |
| **신**뢰도  (Reliability) | 시스템이 주어진 문제를 정확하게 해결하는 정도 |

#**처반사신**

**3) 운영체제의 기능 ★ \_\_ 20년 3회 기출문제**

- CPU, 메모리 공간, 프로세서(처리기, Processor), 기억장치(주기억장치, 보조 기억장치), 입, 출력 장치, 파일 및 정보 등의 자원 관리

- 다중 사용자와 다중 응용프로그램 환경 하에서 현재 상태를 파악하고 자원을 효율적으로 분배 및 관리하기 위해 스케줄링 기능 제공

- 사용자와 시스템 간의 편리한 인터페이스 제공, 입출력 장치와 사용자 프로그램 제어

- 데이터를 관리하고, 데이터 및 자원의 공유 기능을 제공

- 시스템의 오류 검사 및 복구, 자원 보호 기능 제공

- 입, 출력에 대한 보조 기능 제공 → 가상 계산기 기능 제공

**4) 운영체제의 주요 자원 관리**

|  |  |
| --- | --- |
| **자원** | **기능** |
| 프로세스 관리 | 프로세스 스케줄링 및 동기화 관리 담당 |
| 기억장치 관리 | 프로세스에게 메모리 할당 및 회수 관리 담당 |
| 주변장치 관리 | 입, 출력장치 스케줄링 및 전반적인 관리 담당 |
| 파일 관리 | 파일의 생성과 삭제, 변경, 유지 등의 관리 담당 |

**5) 운영체제의 종류 ★**

|  |  |  |
| --- | --- | --- |
| **운영체제** | **특징** | **인터페이스** |
| **Windows** | Microsoft사가 개발 | GUI |
| **UNIX** | AT&T 벨 연구소, MIT, Generic Electric이 공동 개발 | CLI |
| **LINUX** | 리누스 토발즈(Linus Torvalds)가 개발  UNIX와 호환이 가능한 커널(Kernel)  누구나 제한 없이 활용 및 재배포 가능한 오픈 소스 | CLI |
| **MacOS** | Apple 사가 UNIX를 기반으로 개발 | GUI |
| **MS-DOS** | Windows 이전에 사용되던 운영체제 | CLI |

**6) Windows의 주요 특징 ★**

▶ **GUI**(Graphic User Interface, 그래픽 사용자 인터페이스)

-키보드로 명령어를 직접 입력하지 않고, 마우스로 아이콘이나 메뉴를 선택하여 모든 작업을 수행하는 방식

→ 초보자도 쉽게 사용할 수 있게 GUI 채용

▶ **선점형 멀티태스킹**(Preemptive Mulit-Tasking)

-동시에 여러 개의 프로그램을 실행하면서 운영체제가 각 작업의 CPU 이용 시간을 제어하여 응용 프로그램 실행 중 문제가 발생하면 해당 프로그램을 강제 종료시키고 모든 시스템 자원을 반환하는 방식

→ 하나의 응용 프로그램이 CPU를 독점하는 것을 방지할 수 있어 시스템 다운 현상없이 더욱 안정적인 작업을 할 수 있음

▶ **PnP**(Plug and Play, 자동 감지 기능)

-컴퓨터 시스템에 프린터나 사운드 카드 등의 하드웨어를 설치했을 때, 해당 하드웨어를 사용하는 데 필요한 시스템 환경을 OS가 자동으로 구성해주는 기능

→ 운영체제가 하드웨어의 규격을 자동으로 인식하여 동작하게 해주므로 PC 주변장치를 연결할 때 사용자가 직접 환경을 설정하지 않아도 됌,

PnP기능을 활용하기 위해서는 하드웨어와 소프트웨어 모두 PnP를 지원해야 함

▶ **OLE**(Object Linking and Embedding)

-다른 여러 응용 프로그램에서 작성된 문자나 그림 등의 개체(Objects)를 현재 작성 중인 문서에 자유롭게 연결(Linking)하거나 삽입(Embedding)하여 편집할 수 있게 하는 기능

→ OLE로 연결된 이미지를 원본 프로그램에서 수정하거나 편집하면 그 내용이 그대로 해당 문서에 반영됨

▶ 255자의 긴 파일명

-VFAT(Virtual File Allocation Table)를 이용해 최대 255자까지 파일 이름 지정 가능

→ 파일 이름으로는 \ / : \* ? “ < > |를 제외한 모든 문자 및 공백을 사용할 수 있으며, 한글의 경우 127자까지 저장 가능

▶ 개인 사용자(Single-User) 시스템

-컴퓨터 한 대를 한 사람만이 독점해서 사용

**6) UNIX의 개요 및 특징 ★★**

- 소스가 공개된 개방형 시스템(Open System)

- 시분할 시스템(Time Sharing System)을 위해 설계된 **대**화식 운영체제

- **다**중 작업(Multi-Tasking, 멀티 태스킹) 지원

- 다중 **사**용자(Multi-User) 지원

- 대부분 C언어로 작성되어 있어 **이**식성이 높으며 장치, 프로세스 간의 호환성이 높음

- **계**층 구조(트리 구조)의 파일 시스템

#**대다사이계**

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

▶ **커널(Kernel)** **★**

-UNIX의 가장 핵심적인 부분

-컴퓨터가 부팅될 때 주기억장치에 적재된 후 상주하면서 실행됨

-하드웨어를 보호하고, 프로그램과 하드웨어 간의 인터페이스 역할을 담당

-프로세스 관리, 기억장치 관리, 파일 관리, 입, 출력 관리 등 여러 가지 기능 수행

**▶ 쉘(Shell)** **★ \_\_ 20년 1, 2회 기출문제**

-사용자의 명령어를 인식하여 프로그램을 호출하고, 명령을 수행하는 명령어 해석기

-주기억장치에 상주하지 않고, 명령어가 포함된 파일 형태로 존재하며 보조 기억장치에서 교체 처리가 가능

-시스템과 사용자 간의 인터페이스 역할을 담당

-파이프라인 기능 지원 및 입, 출력 재지정을 통해 입, 출력의 방향 변경 가능

-여러 종류의 쉘이 있음

-DOS의 COMMAND.COM과 같은 기능 수행

**7) UNIX에서의 프로세스 간 통신**

- 각 프로세스는 시스템 호출을 통해 커널의 기능을 사용하며, 프로세스 간 통신은 시그널(Signal), 파이프(Pipe), 소켓(Socket) 사용

▶ **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

**8) LINUX의 개요 및 특징**

- 1991년 리누스 토발즈(Linus Torvalds)가 UNIX를 기반으로 개발한 운영체제

- 대부분의 특징이 UNIX와 동일하며 UNIX와 완벽하게 호환됨

- 프로그램 소스 코드가 무료로 공개되어 있음

**9) MacOS의 개요 및 특징**

- 1980년대 Apple사가 UNIX를 기반으로 개발한 운영체제

- 아이맥(iMAC)과 맥북(MacBook) 등 애플 사에서 생산하는 제품에서만 사용 가능

- 드라이버 설치 및 install과 uninstall의 과정이 단순

**기억장치 관리 ★★★**

**p.640, 4-43**

**1) 기억장치의 관리 전략의 개요 ★**

- 보조기억장치의 프로그램이나 데이터를 주기억장치에 적재시키는 시기(When), 적재 위치(Where) 등을 지정하여 한정된 주기억장치의 공간을 효율적으로 사용하기 위함

# **반**입(Fetch), **배**치(Placement), **할**당(Allocation), **교**체(Replacemnet)

#**반배할교**

**2) 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 적재하는 방법

▶ 예상 반입(Anticipatory Fetch): 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재하는 방법

**3) 배치(Placement) 전략 \_\_ 20년 3회 기출문제**

- 새로 반입되는 프로그램이나 데이터를 주기억장치의 어디에(Where) 위치시킬 것인지를 결정하는 전략

▶ **최**초 적합(First Fit): 빈 영역 중에서 첫 번째 분할 영역에 배치

▶ 최**적** 적합(Best Fit): 빈 영역 중에서 단편화를 가정 작게 남기는 분할 영역에 배치

▶ 최**악** 적합(Worst Fit): 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치

#**최적악**

**4) 교체(Replacement) 전략 \_\_ 4-46**

- 이미 사용되고 있는 영역 중에서 어느(Who) 영역을 교체할지 결정하는 전략

# FIFO, LRU, LFU, NUR, OPT, SCR

**5) 주기억장치 할당(Allocation)의 개념 ★**

- 프로그램이나 데이터를 실행시키기 위해 주기억장치에 어떻게(How) 할당할지 정함

▶ **연**속 할당 기법: 프로그램을 주기억장치에 연속으로 할당하는 기법  
# **단**일 분할 할당 기법: 오버레이, 스와핑

# **다**중 분할 할당 기법: 고정(정적) 분할 할당 기법, 가변(동적) 분할 할당 기법

▶ **분**산 할당 기법: 프로그램을 특정 단위의 조각으로 나누어 할당하는 기법  
# **페**이징(Paging) 기법 / **세**그먼테이션(Segmentation) 기법

#**연단다 분페세**

**6) 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

**7) 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

- 일정한 크기로 나눈 단위를 페이지(Page)라 하고, 페이지 크기로 일정하게 나누어진 주기억장치의 단위를 페이지 프레임(Page Frame)이라 함

- 외부 단편화는 발생하지 않으나, **내부 단편화 발생 \_\_ 4-48**

- 주소 변환을 위해 페이지의 위치 정보를 갖고 있는 페이지 맵 테이블(Page Map Table) 필요 → 페이지 맵 테이블 사용으로 비용 증가, 처리 속도 감소

**8) 세그먼테이션(Segmentation) 기법 ★**

- 가상기억장치에 보관되어 있는 프로그램을 **다양한** 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 기억공간을 절약하기 위해서 사용하는 실행시키는 방법

- 논리적인 크기로 나눈 단위를 세그먼트(Segment)라고 하며, 각 세그먼트는 고유한 이름과 크기를 가짐

- 기억장치의 사용자 관점을 보존하는 기억장치 관리 기법

- 주소 변환을 위해서 세그먼트가 존재하는 위치 정보를 갖고 있는 세그먼트 맵 테이블(Segment Map Table) 필요

- 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 \_\_ 4-48**

**8) 페이지 교체 알고리즘 ★**

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

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

▶ **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)

-LRU와 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, 2차 기회 교체)

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

**9) 페이지 크기 ★★**

▶ **페이지 크기가 작을 경우**

-페이지 단편화가 감소되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 줄어듬

-불필요한 내용이 주기억장치에 적재될 확률이 적으므료 효율적인 워킹 셋 유지 가능

-Locality에 더 일치할 수 있기 때문에 기억장치 효율 높아짐

-페이지 정보를 갖는 페이지 맵(사상) 테이블의 크기가 커지고, 매핑 속도가 늦어짐

-디스크 접근 횟수가 많아져서 전체적인 입, 출력 시간은 늘어남

▶ **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

**10) Locality(지역성, 구역성) ★★**

- 프로세스가 실행되는 동안 주기억장치를 참조할 때 일부 페이지만 집중적으로 참조하는 성질이 있다는 이론

- 스래싱(Thrashing)을 방지하기 위한 워킹 셋 이론의 기반

- 데닝(Denning) 교수에 의해 구역성의 개념이 증명됐으며, 캐시 메모리 시스템의 이론적 근거

|  |  |
| --- | --- |
| **종류** | **설명** |
| **시**간적 구역성  (Temporal Locality) | 한 번 참조한 페이지는 가까운 시간 내에 계속 참조할 가능성이 높음  # Loop(**루**프), Stack(**스**택), Subroutine(**서**브루틴),  Counting(**카**운팅), Totaling(**집**계) **★**  #**루스서카집** |
| **공**간적 구역성  (Spatial  Locality) | 어느 하나의 페이지를 참조하면 그 근처의 페이지를 계속 참조할 가능성이 높음  # Array(배열), Sequential Code(순차적 코드) **★** |

**11) 워킹 셋(Working Set) ★ \_\_ 4-50**

- 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합

- 주기억장치에 상주시킴으로써 페이지 부재 및 페이지 교체 현상이 줄어들어 프로세스의 기억장치 사용이 안정됨

- 시간이 지남에 따라 자주 참조하는 페이지들의 집합이 변화하기 때문에 워킹 셋은 시간에 따라 변경됨

**12) 페이지 부재 빈도(PFF; Page Fault Frequency) 방식 \_\_ 4-50**

- 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

**13) 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

**14) 스래싱(Thrashing) ★ \_\_ 4-49**

- 프로세스의 처리 시간보다 페이지 교체에 소요되는 시간이 더 많아지는 현상

→ 전체 시스템 성능 저하

- 다중 프로그래밍(멀티 태스킹)의 정도가 높아짐에 따라 CPU의 이용률은 어느 특정 시점까지는 높아지지만, 다중 프로그래밍의 정도가 더욱 커지면 스래싱이 나타나고, CPU의 이용률은 급격히 감소됨

▶ **스래싱 현상 방지 방법**

-다중 프로그래밍의 정도를 적정 수준으로 유지 **★**

-페이지 부재 빈도(Page Fault Frequency)를 조절해 사용

-워킹 셋(Working Set)을 유지함

-부족한 자원을 증설하고, 일부 프로세스를 중단시킴

**프로세스 및 스케줄링 ★★**

**p.657~661, 4-51**

**1) 프로세스(Process)의 정의**

- 일반적으로 프로세서(처리기, Processor), 즉 CPU에 의해 처리되는 사용자 프로그램, 시스템 프로그램인 실행중인 프로그램을 의미하며 작업(Job), 태스크(Task)라고도 함

- 프로세서(Processor) → 프로세스(Process) → 프로시저(Procedure, 절차)

▶ 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

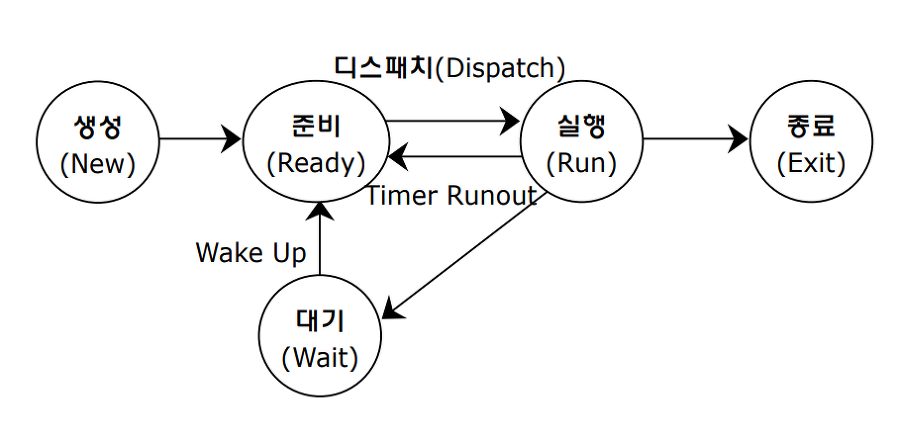
-프로세서가 할당되는 실체로서, 디스패치가 가능한 단위

**2) PCB(Process Control Block, 프로세스 제어 블록) ★**

|  |  |
| --- | --- |
| **저장 정보** | **설명** |
| 프로세스 고유 **식**별자 | 프로세스를 구분할 수 있는 고유의 번호 |
| 프로세스의 현재 **상**태 | 준비, 대기, 실행 등의 프로세스 상태 |
| 프로그램 **카**운터 | 실행될 명령어의 주소를 가지고 있는 레지스터 |
| CPU **레**지스터 정보 | 누산기, 인덱스 레지스터, 범용 레지스터 등에 대한 정보 |
| **스**케줄링 및  프로세스의 우선순위 | 스케줄링 정보 및 프로세스가 실행될 우선 순위 |
| **계**정 정보 | CPU 사용 시간, 실제 사용 시간, 한정된 시간 |
| **입**, 출력 상태 정보 | 입, 출력장치, 개방된 파일 목록 |
| **메**모리장치 관리 정보 | 기준 레지스터, 페이지 테이블에 대한 정보 |
| 포인터 | 프로세스가 위치한 메모리 및 할당된 자원에 대한 포인터 |

#**식상카레 스계입메**

**3) 프로세스 상태 전이 ★ \_\_ 20년 1, 2회 기출문제**



**4) 프로세스 상태 전이 관련 용어**

|  |  |
| --- | --- |
| **관련 용어** | **설명** |
| **디스패치**  **(Dispatch)** | 준비 상태에서 대기하고 있는 프로세스 중 하나가 프로세서를 할당받아 실행 상태로 전이되는 과정 **★**  # 준비(Ready) → 실행(Run) |
| **Wake Up** | 프로세스가 대기 상태에서 준비 상태로 전이되는 과정  # 대기(Wait) → 준비(Ready) |
| **스풀링(Spooling)** | 나중에 한꺼번에 입, 출력하기 위해 디스크에 저장하는 과정 |

**5) 스레드(Thread) ★ \_\_ 20년 1, 2회 기출문제**

- 프로세스 내에서의 작업 단위로서 시스템의 여러 자원을 할당받아 실행하는 단위

▶ **단일 스레드**: 하나의 프로세스에 하나의 스레드가 존재하는 경우

▶ **다중 스레드**: 하나의 프로세스에 하나 이상의 스레드가 존재하는 경우 **★**

- 프로세스의 일부 특성을 갖고 있기 떄문에 경량 프로세스라고도 함

- 동일 프로세스 환경에서 서로 독립적인 다중 수행 가능

→ 하나의 프로세스를 여러 개의 스레드로 생성해 병행성 증진 및 성능과 처리율 향상

프로그램 응답 시간 단축과 기억장소의 낭비가 줄어들고 통신이 향상됨 **★**

**6) 스케줄링(Scheduling)의 개요**

- 프로세스가 생성되어 실행될 떄 필요한 시스템의 여러 자원을 해당 프로세스에게 할당하는 작업

|  |  |
| --- | --- |
| **종류** | **설명** |
| **장기 스케줄링**  (작업 스케줄링,  상위 스케줄링) | 어떤 프로세스가 시스템의 자원을 차지할 수 있도록 할 것인가를 결정하여 준비상태 큐로 보내는 작업  → 작업 스케줄러에 의해 수행됨 |
| **중기 스케줄링** | 어떤 프로세스들이 CPU를 할당받을 것인지 결정하는 작업 |
| **단기 스케줄링**  (프로세서 스케줄링,  하위 스케줄링) | 프로세스가 실행되기 위해 CPU를 할당받는 시기와 특정 프로세스를 지정하는 작업  → 프로세서 스케줄링 및 문맥 교환은 프로세서 스케줄러에 의해 수행됨 |

# **문맥 교환(Context Switching)**: 하나의 프로세스에서 다른 프로세스로 CPU가 할당되는 과정에서 발생되는 것 **★**

**7) 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

→ CPU이용률, 처리율, 반환 시간, 대기 시간, 응답 시간 **★**

**8) 프로세스 스케줄링의 기법 ★ \_\_ 4-54**

▶ 선점(Preemptive) 스케줄링: 하나의 프로세스가 CPU를 할당받아 실행하고 있을 때 우선순위가 높은 다른 프로세스가 CPU를 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

# **R**ound Robin, SR**T**(Shortest Remaining Time), **M**LQ(Multi-Level Queue), M**F**Q

#**RTMF**

▶ 비선점(Non-Preemptive) 스케줄링: 이미 할당된 CPU를 다른 프로세스가 강제로 빼앗아 선점할 수 없는 기법

- CPU를 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(**P**riority), 기한부(**D**eadline), **F**CFS(FIFO), S**J**F(Shortest Job First), **H**RN

#**PDF JH**

**\* HRN(Highest Response-ratio Next) \_\_ 20년 1, 2, 3회 기출문제**

SJF 기법의 가뭄 현상을 보완하기 위한 방식으로, 대기 시간이 긴 프로세스일 경우 우선순위가 높아지고, 우선순위 계산식의 수치가 **가장 높은 것부터 낮은 순으로 우선순위를 부여**해 긴 작업과 짧은 작업 간의 지나친 불평등을 해소함

→ **HRN 우선순위 계산식**: (대기시간 + 서비스시간) / 서비스시간 **★**

**인터넷 및 OSI 참조 모델 ★★★**

**p.669~672, 4-70**

**1) IP 주소(Internet Protocol Address) \_\_ 4-78**

- 인터넷에 연결된 모든 컴퓨터 자원을 구분하기 위한 고유한 주소

- 숫자로 8비트씩 4부분, 총 32비트로 구성됨

|  |  |
| --- | --- |
| **클래스** | **설명 / 서브넷 마스크** |
| **A** Class | 국가나 대형 통신망에 사용(0~127) / 255.0.0.0 |
| **B** Class | 중대형 통신망에 사용(128~191) / 255.255.0.0 |
| **C** Class | 소규모 통신망에 사용(192~223) / 255.255.255.0 |
| **D** Class | 멀티캐스트용으로 사용(224~239) / 255.255.255.255 |
| **E** Class | 실험적 주소이며 공용되지 않음(240~255) **★** |

**2) 서브네팅(Subnetting)**

- 할당된 네트워크 주소를 다시 여러 개의 작은 네트워크로 나누어 사용하는 것

- 4바이트의 IP 주소 중 네트워크 주소와 호스트 주소를 구분하기 위한 비트를 서브넷 마스크(Subnet Mask)라 하며, 이를 변경해 네트워크 주소를 여러 개로 분할해 사용

**3) IPv6(Internet Protocol version 6) ★ \_\_ 20년 1, 2, 3회 기출문제**

- 현재 사용하고 있는 IP 주소 체계인 IPv4의 주소 부족 문제를 해결하기 위해 개발됨

- 128비트의 긴 주소를 사용하고, IPv4에 비해 자료 전송 속도가 빠름

- 인증성, 기밀성, 데이터 무결성의 지원으로 보안 문제 해결 가능

- IPv4와 호환성이 뛰어나고, IPv6 확장 헤더로 네트워크 기능 확장이 용이함

- Traffic Class, Flow Label을 이용하여 등급별, 서비스별로 패킷을 구분할 수 있어 품질 보장(**QoS**; Quality of Service)이 용이

# **유**니캐스트(Unicast), **멀**티캐스트(Multicast), **애**니캐스트(Anycast)

#**유멀애 ★**

cf) IPv4: **유**니캐스트, **멀**티캐스트, **브**로드캐스트(Broadcast) #**유멀브**

**4) 도메인 네임(Domain Name)**

- 숫자로 된 IP 주소를 사람이 이해하기 쉬운 문자 형태로 표현한 것

- 호스트 컴퓨터 이름(www), 소속 기관 이름(hankook), 소속 기관의 종류(co), 소속 국가명(kr) → www.hankook.co.kr

- 문자로 된 도메인 네임을 컴퓨터가 이해할 수 있는 IP 주소로 변환하는 역할을 하는 시스템을 DNS(Domain Name System)라고 하며, 이런 역할을 하는 서버를 DNS 서버라 함 **★**

**5) OSI(Open System Interconnection) 참조 모델 ★★ \_\_ 4-73, 20년 1, 2, 3회 기출문제**

|  |  |  |
| --- | --- | --- |
| **계층** | **설명** | **주요 프로토콜** |
| 응용 계층  (**A**pplication Layer, 7) | 사용자와 네트워크 간 응용서비스 연결, 데이터 생성 | HTTP, FTP, TELNET, SMTP / SNTP, DNS |
| 표현 계층  (**P**resentation Layer, 6) | 구문 검색, 코드 변환, 암/복호화, 데이터 압축, 문맥 관리 기능 | JPEG, MPEG |
| 세션 계층  (**S**ession Layer, 5) | 연결 접속(유지), 동기 제어, 동기점(대화) | SSH, TLS |
| 전송 계층  (**T**ransport Layer, 4) | 종단간(End to End) 신뢰성 있는 데이터 전송, 흐름 제어(슬라이딩 윈도우), 오류 및 혼잡 제어 **★** | TCP / UDP, RTCP  → 세그먼트(Segment) |
| 네트워크 계층  (**Ne**twork Layer, 3) | 단말기 간 데이터 전송을 위한 최적화된 경로(라우팅) 제공 **★** | IP, ICMP, IGMP, ARP, RARP, RIP, OSPF  → 패킷(Packet) |
| 데이터 링크 계층  (**Da**ta Link Layer, 2) | 인접 시스템(노드) 간 물리적 연결을 이용해 데이터 전송, 동기화, 오류 및 흐름제어, 오류검출 및 재전송 **★** | HDLC, PPP, LLC, MAC  → 프레임(Frame) |
| 물리 계층  (**Phy**sical Layer, 1) | 매체 간의 전기적, 기능적, 절차적 기능 정의 | RS-232C, X.21  → 비트(Bit) |

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

**네트워크 관련 장비 및 프로토콜 ★★**

**p.676~679**

**1) 네트워크 관련 장비**

|  |  |
| --- | --- |
| **장비** | **설명** |
| **게**이트웨이  (Gateway) | 전 계층(1~7계층)의 프로토콜 구조가 다른 네트워크의 연결 수행 **★** |
| **라**우터  (Router) | 브리지와 같이 LAN과 LAN의 연결 기능에 데이터 전송의 최적 경로를 선택할 수 있는 기능이 추가된 것 → 네트워크 계층(Ne) **★** |
| **스**위치  (Switch) | 브리지와 같이 LAN과 LAN을 연결하여 훨신 더 큰 LAN을 만드는 장치,  하드웨어 기반으로 처리해서 전송 속도가 빠름 → 데이터 링크 계층(Da) |
| **브**리지  (Bridge) | LAN과 LAN을 연결하거나 LAN안에서의 컴퓨터 그룹을 연결하는 기능 수행, MAC 브리지라고도 함 → 데이터 링크 계층(Da) |
| **리**피터  (Repeater) | 신호가 왜곡되거나 약해질 경우 원래의 신호 형태로 재생하여 다시 전송하는 역할 수행 → 물리 계층(Phy) |
| **허**브(Hub) | 한 사무실이나 가까운 거리의 컴퓨터들을 연결하는 장치 → 물리 계층(Phy) |

#**게라스 브리허**

**2) 프로토콜(Protocol)의 정의 \_\_ 4-72**

- 서로 다른 기기들 간의 데이터 교환을 원활하게 수행할 수 있도록 표준화시켜 놓은 통신 규약

**3) 프로토콜의 기본 요소 ★★**

|  |  |
| --- | --- |
| **기본 요소** | **설명** |
| **구**문(Syntax) | 전송하고자 하는 데이터의 형식, 부호화, 신호 레벨 등을 규정 |
| **의**미(Semantics) | 두 기기 간의 효율적이고 정확한 정보 전송을 위한 협조 사항과 오류 관리를 위한 제어 정보를 규정 |
| **타**이밍(Timing) | 두 기기 간의 통신 속도, 메시지의 순서 제어 등을 규정 |

#**구의타**

**TCP/IP ★★★**

**p.681**

**1) TCP/IP(Transmission Control Protocol/Internet Protocol)의 개요 \_\_ 3회 기출문제**

- 인터넷에 연결된 서로 다른 기종의 컴퓨터들이 데이터를 주고받을 수 있도록 하는 표준 프로토콜

|  |  |
| --- | --- |
| **TCP** | ▶ OSI 7계층의 전송 계층(4계층)에 해당  - 신뢰성 있는 연결형 서비스 제공 **★**  - 패킷의 다중화, 순서 제어, 오류 제어, 흐름 제어 기능 제공  - 스트림(Stream) 전송 기능 제공 **★** |
| **IP** | ▶ OSI 7계층의 네트워크 계층(3계층)에 해당  - 데이터 그램을 기반으로 하는 비연결형 서비스 제공 **★**  - 패킷의 분해/조립, 주소 지정, 경로 선택 기능(Routing) 제공 **★** |

**2) TCP/IP의 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  (TCP를 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 |

**3) 응용 계층의 주요 프로토콜 ★**

|  |  |
| --- | --- |
| **HTTP**  **(Hypertext Transfer Protocol)** | HTML 문서를 송, 수신하기 위한 표준 프로토콜 |
| **FTP**  **(File Transfer Protocol)** | 파일을 주고받을 수 있는 원격 파일 전송 프로토콜 |
| **TELNET** | 멀리 떨어져 있는 컴퓨터에 접속하여 자신의 컴퓨터처럼 사용할 수 있도록 해주는 서비스  → 가상의 터미널 기능 수행 |
| **SMTP**  **(Simple Mail**  **Transfer Protocol)** | 전자 우편을 교환하는 서비스 |
| **SNTP**  **(Simple Network**  **Management Protocol)** | TCP/IP의 네트워크 관리 프로토콜로, 라우터(Ne), 허브(Phy) 등 네트워크 정보를 네트워크 관리 시스템에 보내는 데 사용되는 표준 통신 규약 |
| **DNS**  **(Domain Name System)** | 도메인 네임을 IP 주소로 매핑(Mapping, 연결)하는 시스템 |

**4) 전송 계층의 주요 프로토콜 ★**

|  |  |
| --- | --- |
| **TCP**  **(Transmission**  **Control**  **Protocol)** | - 신뢰성 있는 연결형 서비스 제공 **★**  - 순서 제어, 오류 제어, 흐름 제어 기능 제공 → 투명성  - 스트림(Stream) 전송 기능 제공 **★**  - 양방향 연결(Full Duplex Connection)형 서비스 제공  - 가상 회선 연결 형태의 서비스 제공 |
| **UDP**  **(User**  **Datagram**  **Protocol)** | - 신뢰성보다는 속도가 중요시되는 네트워크에서 사용  - 실시간 전송에 유리함 **★**  - 데이터 전송 전에 연결을 설정하지 않는 비연결형 서비스 제공 **★**  - TCP에 비해 단순한 헤더 구조를 가지므로, 오버헤드 적음 **★** \* UDP 헤더: **So**urce Port, **De**stination Port, **Len**gth, **Che**cksum, **Da**ta  #**소데랭체데** |
| **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 32비트의 경계로 끝남 |

**5) 인터넷 계층의 주요 프로토콜 ★ \_\_ 20년 1, 2회 기출문제**

|  |  |
| --- | --- |
| **IP**  **(Internet Protocol)** | - 전송할 데이터에 주소를 지정하고, 경로 설정 기능을 함  - 비연결형인 데이터그램 방식을 사용해 신뢰성 보장 X |
| **ICMP**  **(Internet Control**  **Message Protocol)** | IP와 조합하여 통신중에 발생하는 오류의 처리와 전송 경로 변경 등을 위한 제어 메시지를 관리하는 역할을 하며, 헤더는 8Byte로 구성됨 |
| **IGMP**  **(Internet Group**  **Management Protocol)** | 멀티캐스트를 지원하는 호스트나 라우터 사이에서 멀티캐스트 그룹 유지를 위해 사용됨 |
| **ARP**  **(Address Resolution**  **Protocol)** | 호스트의 IP 주소를 호스트와 연결된 네트워크 접속 장치의 물리적 주소(MAC Address)로 바꿈  # IP 주소 → MAC 주소 **★** |
| **RARP**  **(Reverse Address**  **Resolution Protocol)** | ARP와 반대로 물리적 주소(MAC Address)를 IP 주소로 변환하는 기능을 함  # MAC 주소 → IP 주소 **★** |

**6) 네트워크 액세스 계층의 주요 프로토콜 ★**

|  |  |
| --- | --- |
| **Ethernet(IEEE 802.3)** | CSMA/CD 방식의 LAN |
| **IEEE 802** | LAN을 위한 표준 프로토콜 |
| **HDLC** | 비트 위주의 데이터 링크 제어 프로토콜 |
| **X.25** | 패킷 교환망을 통한  DTE와 DCE 간의 인터페이스를 제공하는 프로토콜 |
| **RS-232C** | 공중 전화 교환망(PSTN)을 통한  DTE와 DCE 간의 인터페이스를 제공하는 프로토콜 |

**추가 정리, 수제비 및 기출문제** ★★★

**1) 형상 관리 절차 \_\_ 4-5**

▶ 형상 **식**별: 형상 관리 대상에 이름과 관리 번호를 부여하고, 계층(Tree) 구조로 구분하여 수정 및 추적이 용이하도록 하는 작업

▶ 형상 **통**제(변경 관리): 식별된 형상 항목에 대한 변경 요구를 검토하여 현재의 기준선(베이스 라인, Base line)이 잘 반영될 수 있도록 조정하는 작업

▶ 형상 **감**사: 기준선(베이스 라인)의 무결성을 평가하기 위해 확인, 검증, 검열 과정을 통해 공식적으로 승인하는 작업

▶ 형상 **기**록(상태 보고): 형상의 식별, 통제, 감사 작업의 결과를 기록, 관리하고 보고서를 작성하는 작업

#**식통감기**

**2) 모듈화 \_\_ 4-11**

|  |  |
| --- | --- |
| **원리** | **설명** |
| **정**보 은닉  (Information Hiding) | 어렵거나 변경 가능성이 있는 모듈을 타 모듈로부터 은폐 |
| **분**할과 정복  (Divide & Conquer) | 복잡한 문제를 분해, 모듈 단위로 문제 해결 |
| 데이터 **추**상화  (Data Abstraction) | 각 모듈 자료 구조를 액세스하고 수정하는 함수내에 자료 구조의 표현 내역을 은폐 |
| 모듈 **독**립성  (Module Inpendency) | 낮은 결합도와 높은 응집도 |

#**정분추독**

**3) 예외 처리 구성 \_\_ 4-38**

|  |  |
| --- | --- |
| **구성** | **설명** |
| **throw** | - 프로그램이 정상적으로 실행될 수 없는 상황일 때 예외를 던짐  - 강제로 예외를 발생시키는 경우에 사용하는 명령어 |
| **try** | - 예외가 발생할 만한 코드 블록을 저장  - try { } 괄호 안에 예외 처리 대상 코드를 작성  - 블록 안에서 예외가 발생했을 때 throw 명령으로 예외를 던짐 |
| **catch** | - if-else문처럼 try-catch문으로 한 쌍으로 쓰임  - try안에서 throw한 예외 객체에 대한 예외 처리  - catch 블록을 예외 핸들러(Exception Handler)라고 부름 |

**4) 프로토타입(Prototype) \_\_ 4-39**

- 속성과 메서드를 다른 클래스의 인스턴스 또는 빈 객체에 복제, 생성하는 작업을 덜 수 있는 프로그래밍 스타일

- 객체지향 프로그래밍과 달리 클래스를 명확히 정의하지 않아도 됨

**5) 라우팅 프로토콜(Network, 3계층) ★ \_\_ 4-85**

|  |  |
| --- | --- |
| **프로토콜** | **설명** |
| **RIP**  (Routing Information Protocol) | ▶ IGP(Interior Gateway Protocol)로 **Bellman-Ford 알고리즘**을 이용하여 최적의 경로를 설정하는 **소규모** 프로토콜  - **최대 홉(Hop) 수를 15로 제한**  - **거리 벡터 프로토콜**이라고도 함 |
| **OSPF**  (Open Shortest Path First) | ▶ IGP(Interior Gateway Protocol)로 RIP의 단점 개선을 위해 **daijkstra 알고리즘** 및 **Link Static 기반**으로 최단경로를 찾는 **대규모** 프로토콜 |
| **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 교환**하고, 이후에는 변화된 정보만 교환 |

**6) 은행가 알고리즘(Banker’s Algorithm) \_\_ 개정 전 기출문제, 20년 1, 2회 기출문제**

- 교착상태의 해결 방법 중 Avoidance(회피) 사용

**7) 교착 상태 발생의 필요 충분 조건 \_\_ 개정 전 기출문제, 20년 1, 2회 기출문제**

- **상**호 배제(Mutual Exclusion), **점**유와 대기(Hold and Wait), **환**형 대기(Circular Wait), **비**선점(Non-Preemption)

#**상점환비**

**8) IEEE 802.11e \_\_ 개정 전 기출문제, 20년 1, 2회 기출문제**

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

**9) JAVA 언어의 접근제한자 ★★ \_\_ 개정 전 기출문제, 20년 1, 2회 기출문제**

|  |  |
| --- | --- |
| **종류** | **설명** |
| **public** | 모든 접근 허용 |
| **protected** | 같은 패키지(폴더)에 있는 객체와 상속관계의 객체들만 접근 허용 |
| **default** | 같은 패키지(폴더)에 있는 객체들만 접근 허용 |
| **private** | 현재 객체 내에서만 접근 허용 |

→ **public > protected > default > private**

**10) 배열의 초기화 \_\_ p.589, 기출문제**

78. C 언어에서 배열 b[5]의 값은? (2020년 제1, 2회차 필기시험, B형)

static int b[9]={1, 2, 3};

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |

b[0] b[1] b[2] b[3] b[4] **b[5]** b[6] b[7] b[8]

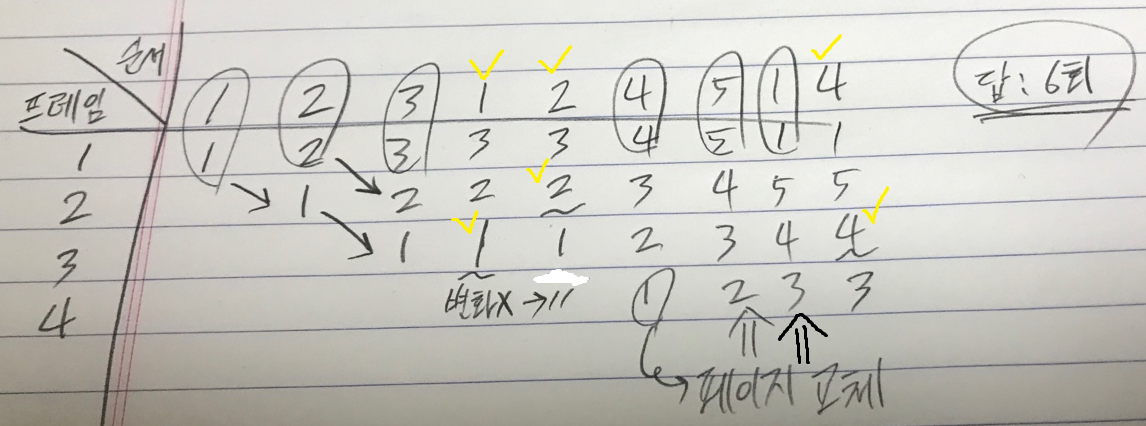
→ **b[5] = 0;**

**11) 페이지 결함(Page Fault) 횟수 \_\_ 19년 2회 기출문제, 20년 1, 2회 기출문제**

46. **4개의 프레임**을 수용할 수 있는 주 기억장치가있으며, 초기에는 모두 비어 있다고 가정한다. 다음의 순서로 페이지 참조가 발생할 때, **FIFO 페이지 교체 알고리즘**을 사용할 경우 **페이지 결함의 발생 횟수**는?



 ① 4회 ② 5회 **③ 6회** ④ 7회



**12) UNIX 명령어 \_\_ p. 667, 20년 3회, 21년 1회 기출문제**

|  |  |
| --- | --- |
| **명령어** | **설명** |
| **cat** | 파일 내용 화면 표시, 커널 버전 확인 = **TYPE** (Windows 명령어) |
| chdir | 현재 사용할 디렉터리의 위치 변경 = **CD** (Windows 명령어) |
| **chmod** | 파일의 사용 허가 지정, 파일의 속성 변경 = ATTRIB (Windows) |
| **chown** | 소유자 변경, **ch**ange **own** |
| cp | 파일 복사, **c**o**p**y |
| rm | 파일 삭제, **r**e**m**ove |
| exec | 새로운 프로세스 수행, **exec**ute |
| find | 파일 찾기 |
| **fork** | 새로운 프로세스 생성, 하위 프로세스 호출 및 프로세스 복제 명령 |
| **fsck** | 파일 시스템 검사 및 보수, **f**ile**s**ystem **c**hec**k** |
| **ls** | 현재 디렉터리 내의 파일 목록 확인, **l**i**s**t = **DIR** (Windows 명령어) |
| mount/unmount | 파일 시스템 마운팅/마운팅 해제 |

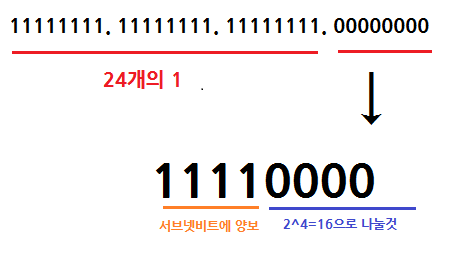
**13) 서브넷(Subnet) \_\_ 17년 2회 기출문제, 20년 3회 기출문제**

89. **200.1.1.0/24 네트워크**를 **FLSM 방식**을 이용하여 **10개의 subnet**으로 나누고 **ip subnet -zero**를 적용했다. 이때 서브네팅된 네트워크 중 **10번째 네트워크의 broadcast IP 주소**는?

**① 200.1.1.159** ② 201.1.5.175

  ③ 202.1.11.191 ④ 203.1.255.245

→ “**10**”개의 subnet으로 나눠야 함으로 2^3=”**8**”로는 부족하고, 2^4=”**16**”으로 나눔



200.1.1.0.0~200.1.1.0.15 (0~15)  
200.1.1.16.0~200.1.1.0.31 (16~31)  
200.1.1.32.0~200.1.1.0.47 (32~47)  
200.1.1.48.0~200.1.1.0.63 (48~63)  
200.1.1.64.0~200.1.1.0.79 (64~79)  
200.1.1.80.0~200.1.1.0.95 (80~95)  
200.1.1.96.0~200.1.1.0.111 (96~111)  
200.1.1.112.0~200.1.1.0.127 (112~127)  
200.1.1.128.0~200.1.1.0.143 (128~143)  
200.1.1.144.0~**200.1.1.0.159** (144~159) - 10번째

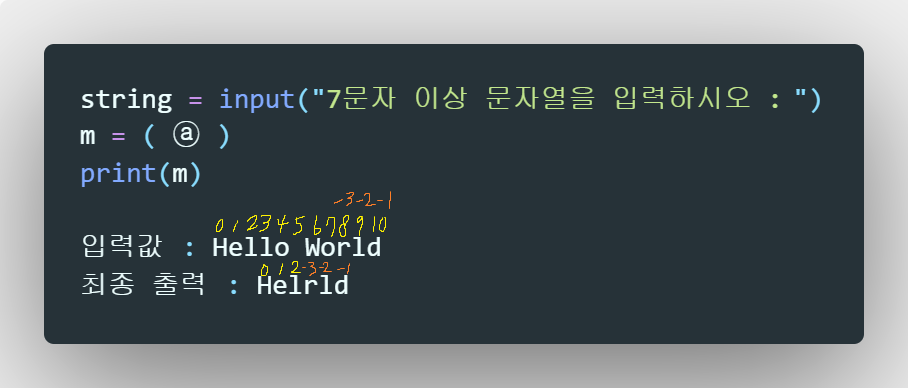
**14) 소프트웨어 취약점 \_\_ 20년 3회 기출문제**

- 메모리를 다루는 데 오류가 발생하여 잘못된 동작을 하는 프로그램 취약점

→ **버퍼 오버플로**

**15) Python List [ ] \_\_ 20년 3회 기출문제**

74. 다음은 사용자로부터 **입력받은 문자열에서 처리음**과 **끝의 3글자를 추출**한 후 합쳐서 출력하는 파이썬 코드에서 ⓐ에 들어갈 내용은?



① string[1:3] + string[-3:] ② string[:3] + string[-3:-1]

**③ string[0:3] + string[-3:]** ④ string[0:] + string[:-1]

① 최종 출력: elrld → [1, 2] + [-3, -2, -1]

② 최종 출력: Helrl → [0, 1, 2] + [-3, -2]

**③ 최종 출력: Helrld → [0, 1, 2] + [-3, -2, -1]**

④ 최종 출력: Hello WorldHello Worl → [0 ~ 10] + [-11 ~ -2]

\***정리**: string[ ] 안에서 **:을 기준**으로 **앞에 값은 시작**, **뒤의 값은 해당하는 값의 전**까지 **list를 출력**함

**16) Java 출력함수 \_\_ 20년 4회 기출문제**

|  |  |
| --- | --- |
| **명령어** | **설명** |
| **system.out.print()** | 기본 출력 (줄바꿈 X → \n 써서 줄바꿈 가능) |
| **system.out.printIn()** | 출력 시 자동으로 줄바꿈 |
| **system.out.printf()** | 연산도 출력할 수 있음 (가장 다양한 표현 가능) |

**17) C언어 데이터 처리 – 열거체, 구조체, 공용체 \_\_ 20년 4회 기출문제**

|  |  |
| --- | --- |
| **종류** | **설명** |
| **열거체** | 서로 연관된 정수형 상수들의 집합으로, 정수형 상수에 이름을 붙여 코드를 이해하기 쉽게 하고, **enum**으로 선언해 사용하는 사용자 정의 자료형 |
| **구조체** | 각 변수가 다른 메모리에 할당되어 있고 C, C++에서 **struct**로 선언하여 사용자가 기본 타입을 가지고 새롭게 정의할 수 있는 사용자 정의 자료형 |
| **공용체** | 모든 멤버 변수가 하나의 메모리 공간을 공유하며 C, C++에서 **union**으로 선언하여 사용하는 사용자 정의 자료형 |

**18) PHP 연산자 \_\_ 20년 4회 기출문제**

|  |  |
| --- | --- |
| **종류** | **설명** |
| **@** | 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 (좀 더 깐깐) |
| **::** | new 지시자로 class를 미리 객체화 시켜놓지 않고, 사용하는 시점에서 객체가 생성되고 지정된 method가 실행되도록 하는 접근자 |