프로그래밍 일반

Section 1 운영체제 개요

1. 운영체제(OS; Operating System)의 기초 정리

① 운영체제는 컴퓨터 사용자와 컴퓨터 하드웨어 간의 인터페이스로서 동작하는 시스템 소프트웨어이다.
② 시스템 프로그램 : 프로그램을 메모리 적재, CPU관리, 메모리 관리, 디스크 관리 등)
③ 제한된 환경에서 자원을 효율적으로 관리하는 프로그램이다.
④ 자원 : CPU(프로세서, 처리기), 기억장치(ROM, RAM, 디스크), 주변장치, 프로세스, 응용프로그램(엑셀, 게임, 워드 등)
⑤ 사용자의 편이성을 제공한다.
⑥ 실행 : 과거 - ROM에 상주, 최근 - 디스크에 저장 -> 전원 ON -> RAM에 상주 (Bootstrapping Loader)
⑦ 운영체제는 스스로 어떤 유용한 기능도 수행하지 않고 다른 응용 프로그램이 유용한 작업을 할 수 있도록 환경을 마련하여 준다.
⑧ 운영체제의 종류로는 MS-DOS, Windows XP/Vista/7, LINUX, UNIX, OS/2, Mac OS 등이 있다.

2. 운영체제의 개념

1) 운영체제의 정의
① 하드웨어를 제어하는 소프트웨어
② 하드웨어를 활용할 수 있도록 펌웨어나 소프트웨어로 만들어진 프로그램
③ 컴퓨터 본체 및 각 주변 장치를 가장 능률적이고, 경제적으로 사용할 수 있도록 하는 프로그램
④ 컴퓨터를 편리하게 사용하고 하드웨어를 효율적으로 사용할 수 있도록 하는 프로그램
⑤ 컴퓨터 자원들인 기억 장치, 프로세스, 파일 및 정보, 네트워크 및 보호 등을 효율적으로 관리할 수 있는 프로그램의 집합

2) 운영체제의 목적 및 목표
① 컴퓨터 시스템의 처리량, 신뢰성을 최대화한다.
② 컴퓨터 시스템의 반환 시간, 응답 시간, 처리 시간, 대기 시간, 경과 시간을 최소화한다.
③ 컴퓨터를 구성하고 있는 자원을 효율적으로 운영하고 제어한다.
④ 사용자 인터페이스 제공
⑤ 자원 스케줄링(자원 공유가 목적)
⑥ 데이터 공유
⑦ 주변 장치 관리
⑧ 시스템의 이식성(호환성)을 높인다.

3) 로더
① 로더의 일반적인 기능
㉠ 할당(Allocation) ㉡ 연결(Linking)
㉢ 재배치(Relocation) ㉣ 적재(Loading)

② 절대 로더(Absolute Loader)
㉠ 할당 - 프로그래머 ㉡ 연결 - 프로그래머
㉢ 재배치 - 어셈블러 ㉣ 적재 - 로더

3. 운영체제의 발전

① 일괄 처리(Batch Processing) 시스템
② 다중 프로그래밍(Multi Programming) 시스템
③ 온라인(On-Line Processing System) 시스템
④ 시간 분할 처리(Time-Sharing Processing) 시스템
⑤ 실시간 처리(Real Time Processing) 시스템
⑥ 다중 모드 처리(Multi Mode Processing) 시스템
⑦ 분산(Distributed) 시스템
⑧ 병렬(Parallel) 시스템

4. 다중 시스템의 용어 정리

① 다중 프로그래밍(Multi Programming)
하나의 주기억 장치와 CPU로 구성된 컴퓨터 시스템에서 주기억 장치에 여러 개의 프로그램을 적재하여 처리하는 방식으로 단위 시간 내에 처리량을 최대로 한다.
② 다중 프로세싱(Multi Processing)
하나의 주기억 장치와 여러 개의 CPU로 구성된 컴퓨터 시스템에서 주기억 장치에 하나 또는 여러 개의 프로그램을 적재하여 처리하는 방식으로 보통 병렬 시스템을 말한다.
③ 다중 컴퓨터(Multi Computer)
여러 개의 독립적인 컴퓨터 시스템에서 하나의 작업을 공동으로 처리할 수 있는 시스템으로 보통 분산 컴퓨터 시스템이라고 한다.
④ 다중 태스킹(Multi Tasking)
하나의 주기억 장치와 CPU로 구성된 컴퓨터 시스템에서 여러 개의 프로그램을 동시에 처리할 수 있는 방식으로 사용자 관점에서의 다중 프로그래밍을 말한다. 예) 다운받으며 음악을 듣는다.

5. 운영체제 시스템의 성능 평가

1) 성능 평가 요인 4가지
① 처리량(Throughput) : 동일한 시간(단위 시간) 내에서 얼마나 많은 작업량을 처리할 수 있는가의 요인
② 반환 시간(Turn around time) : 요청한 작업에 대하여 그 결과를 사용자에게 되돌려 줄 때까지 소요 되는 시간
③ 신뢰도(Reliability) : 작업의 결과를 얼마나 정확하고 믿을 수 있는가의 요인
④ 이용 가능도(Availability) : 시스템의 전체 운영 시간 중에서 실제 가동하여 사용 중인 시간의 비율(오류 없이 작동된 시간의 비율)

2) 다중 프로그래밍에서의 시간(Time)
① 응답 시간(Response time) : 작업이 처음 실행되기까지 걸린 시간으로 반응 시간이라고도 한다.
② 대기 시간(Waiting time) : 실제 작업을 CPU가 실행하지 않은 시간들을 더한 시간
③ 실행 시간(Running time) : CPU가 작업을 처리하는 시간을 더한 시간
④ 반환 시간(Turn around time) : 실행 시간과 대기 시간을 모두 더한 시간으로 작업이 완료될 때 까지 걸린 시간

3) Bench Mark Program
컴퓨터 시스템이나 CPU, 운영체제 등의 전반적인 성능을 측정, 비교하는 프로그램을 말한다. 운영체제의 성능을 평가한다면 실제 처리할 작업의 환경을 가상적으로 구축하고 가상 데이터를 입력하여 작업의 처리량, 반환 시간, 신뢰성 등을 비교 평가하게 된다.

6, 운영체제 구성 요소(기능에 따른 분류)

1) 운영체제 구성 요소(기능에 따른 분류)
① 제어 프로그램(Control Program)
- 감시 프로그램(Supervisor Program)
- 데이터 관리 프로그램 (Data Management Program)
- 작업 제어 프로그램(Job Control Program)
② 처리 프로그램(Processing Program)
- 언어 번역 프로그램(Language Translator Program)
- 서비스 프로그램(Service Program)
- 문제 프로그램(Problem Program)

2) 감시 프로그램(Supervisor Program)
중추적인 역할을 담당, 운영체제 제어 루틴의 호출을 인식, 해당 루틴의 동작을 감시 감독

3) 데이터 관리 프로그램 (Data Management Program)
자료 전송, 파일의 조작 및 처리, 입출력 자료와 프로그램의 논리적인 연결, 파일과 데이터를 표준화 처리

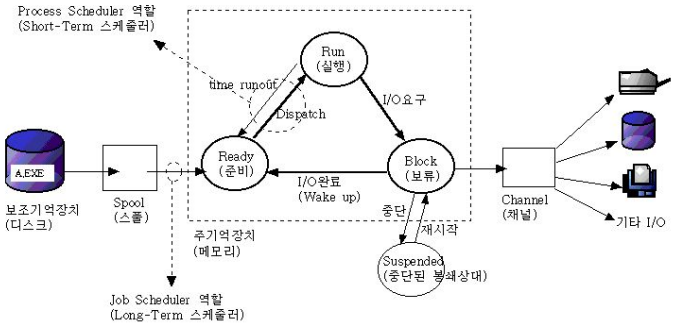
4) 작업 제어 프로그램(Job Control Program)
다른 업무로의 이행을 자동적으로 수행하기 위한 준비 및 그 처리 완료를 담당 작업의 연속 처리를 위한 스케줄 및 시스템 자원 할당 운영체제의 각종 제어 루틴(프로세스 관리, CPU 스케줄링,..)의 수행 순서를 관리 사용자의 명령어 및 프로그램에 따라 제어 루틴들의 작업 시기를 조절

Section 2 프로세스 관리

1. 프로세스 개념

1) 프로세스의 정의
① CPU에 의해서 현재 실행되고 있는 프로그램
② PCB(프로세스 제어 블록)의 존재로서 명시되는 것
③ 프로세서가 할당되는 개체로서 디스패치가 가능한 단위
④ 지정된 결과를 얻기 위한 일련의 계통적 동작
⑤ 목적 또는 결과에 따라 발생되는 사건들의 과정
⑥ 비 동기적 행위를 일으키는 주체
⑦ 프로시저가 활동 중인 것
⑧ 실행 중인 프로시저의 제어 궤적
⑨ CPU가 할당되는 실체

2) 프로세스 상태 전이도



3) 주요 프로세스 상태

① 준비(Read) 상태 : 준비 상태에 있는 여러 프로세스 중 프로세스를 선정하여 CPU를 할당하는 시점
② 실행(Run) 상태 : 프로세스가 CPU를 차지하고 있는 상태
③ 대기(Block) 상태 : 처리 속도가 느린 I/O(입 출력) 작업 중인 상태

4) 용어 정리
① Dispatch : 준비 상태에서 대기하고 있는 프로세스 중 하나가 CPU를 할당받아 실행 상태로 변하는 과정
② time run out : 자신에 할당된 시간만큼 CPU를 사용하고 준비 상태로 변하는 시점
③ I/O 요구 : 프로세스가 CPU를 사용 중에 I/O 행위가 필요하여 보류 상태로 이동하는 시점
④ Wake up : I/O(입출력) 작업이 완료되어 준비 상태로 이동하는 시점
⑤ Spool : 입출력(I/O) 장치와의 속도 차를 극복하기 위한 장치로 대부분 하드디스크가 중재
⑥ Buffering : CPU와 입 출력 장치와의 속도 차이를 줄이기 위해 메모리가 중재

2. 인터럽트 처리(Interrupt Processing)

1) 인터럽트의 처리를 위한 작업 순서
① 인터럽트가 발생하면 운영체제가 제어권을 받는다.
② 운영체제는 인터럽트 받은 현재의 프로세스 상태를 기억한다.
③ 운영체제는 인터럽트의 정보를 분석하여 지정되어 있는 루틴으로 제어권을 넘겨준다.
④ 인터럽트 처리 루틴이 인터럽트를 처리한다.
⑤ 인터럽트가 걸렸던 이전 프로세스의 상태로 복구된다.
⑥ 인터럽트가 걸렸던 시점 이후부터 프로세스가 실행된다.

2) 인터럽트의 종류
① SVC 인터럽트 : 입/출력 수행 루틴 호출, 기억장치 할당 루틴, 오퍼레이터와의 대화
② 입 출력 인터럽트 : 하드웨어적 인터럽트로 입 출력 채널 확인, 준비, 할당, 완료 시에 발생
③ 외부 인터럽트 : 인터럽트 시계에 의해 프로세스가 시간 할당량이 종료된 경우
④ 재시작 인터럽트 : 사용자가 Ctrl+Alt+Del 키를 입력하거나 Reset 키를 이용하여 시스템을 재부팅하는 경우
⑤ 프로그램 검사 인터럽트 : Overflow나 Underflow 상태 시, 나눗셈에서 분모가 0인 경우
⑥ 기계 검사 인터럽트 : 컴퓨터 시스템이 고장으로 발생

4) 문맥 교환(Context Switching)
다중 프로그래밍 시스템에서 운영체제에 의하여 CPU가 할당되는 프로세스를 변경하기 위하여 현재 CPU가 사용하여 실행되고 있는 프로세스의 상태 정보를 저장하고, 앞으로 실행될 프로세스의 상태정보를 설정한 다음에 CPU를 할당하여 실행되도록 하는 작업

3. 프로세스 제어 블록(PCB : Process Control Block)

1) PCB 정의
PCB는 운영체제 내에서 한 프로세스의 존재를 정의한다. 즉, 여러개의 프로세스를 수행하는 다중 프로그래밍 환경 하에서 각 프로세스를 구분하기 위한 프로세스 정보 블록이다.

2) PCB 항목
① 프로세스 식별자
② 프로세스 현재 상태
③ 프로그램 카운터(계수기)
④ 프로세스 우선 순위
⑤ 프로세스가 적재된 기억장치 부분을 가리키는 포인터
⑥ 프로세스에 할당된 자원을 가리키는 포인터
⑦ 처리기 레지스터 정보
⑧ CPU의 각종 레지스터 상태를 가리키는 포인터
⑨ 계정 정보
⑩ 기억장치 관리 정보
⑪ 입 출력 정보
⑫ 부모 프로세스를 가리키는 포인터
⑬ 자식 프로세스를 가리키는 포인터

Section 3 프로세스 스케줄링

1) 성능 평가 기준
① CPU 이용률 : CPU를 쉬지 않고 몇 퍼센트(범위 40% ~ 90%)를 이용하는가의 기준
② 처리 능력(Throughput) : 단위 시간당 처리할 수 있는 CPU의 작업량
③ 대기 시간(Waiting time) : 준비 상태에서 대기하는 시간
④ 응답 시간(Response time, 반응 시간) : 입력에 대해 처음 반응하는 시간
⑤ 반환 시간(Turn around time) : 작업을 지시하고 그 결과가 되돌아오는 시간

2) 바람직한 프로세스 스케줄링 정책
① CPU 이용률을 늘린다. ② 처리율을 늘린다. ③ 대기 시간을 줄인다.
④ 응답 시간을 줄인다. ⑤ 반환 시간을 줄인다. ⑥ 오버 헤드를 줄인다.

3) 비 선점(Non Preemptive)형 방식과 선점(Preemptive)형 방식
① 비 선점형 방식
CPU를 점유하고 있을 때에는 다른 프로세스가 현재 실행 중인 프로세스를 중단시킬 수 없으며 실행이 완료될 때까지 CPU를 독점하는 방식으로

이 방식에는 FIFO, SJF, HRN, 우선 순위, 기한부 스케줄링 방식 등이 있다. 비 선점형 방식들은 모든 프로세스를 관리하는데 공정하다. 우선 순위가 높은 작업들이 중간에 입력되어도 영향을 받지 않고 정해진 시간을 모두 사용하므로 응답 시간을 예측하기 어렵다.

② 선점형 방식
하나의 프로세스가 CPU를 점유하고 있을 때에는 다른 프로세스가 현재 사용 중인 프로세스를 중단시키고 CPU를 차지할 수 있는 방식으로 이 방식에는 라운드 로빈, SRT 등이 있다. 선점형 방식들은 높은 우선 순위의 프로세스들이 긴급을 요할 때 유용하다. 또한 대화식 시분할 시스템에서도 선점형은 빠른 응답 시간을 유지하는데 매우 중요하다. 이러한 이유에서 응답 시간을 예측하기가 비선점형 보다 용이하다.

5) 비선점형 - FIFO(FCFS)
① 알고리즘이 가장 간단하고 구현하기 쉽다.
② 작업이 짧은 작업이나 중요한 작업을 오래 동안 기다리게 할 수 있다.
③ 평균 반환 시간이 길다.

6) 비선점형 - SJF
① 작업이 끝나기까지의 실행 시간 추정치가 가장 작은 작업을 먼저 실행시키는 방식
② FIFO보다 평균 대기 시간이 작지만 긴 작업의 경우 FIFO 기법보다 더 크고 예측이 더욱 어렵다.
③ 실행 시간이 많은 작업일 경우에 무한 연기 현상이 발생될 수 있다.
④ 무한 연기 현상을 방지하기 위해 Aging(에이징) 기법을 사용하여 해결한다.
※ 에이징(Aging) 기법
무한 연기 현상이란 어떠한 시스템이든지 자원 할당 스케줄링 및 CPU 스케줄링 결정에 의해서 특정 프로세스가 무한정 기다리는 현상을 말한다. 이를 해결하기 위해서 자원이 할당되기를 오랜 시간 동안 기다린 프로세스에 대하여 기다린 시간에 비례하는 높은 우선 순위를 부여하여 가까운 시간 안에 자원이 할당되도록 하는 기법을 에이징(Aging)기법이라고 한다.

7) 비선점형 - HRN (서비스시간 + 대기시간) / 서비스 시간

8) 비선점형 - 기한부(deadline)
① 작업이 주어진 특별한 시간이나 만료 시간 안에 완료되도록 하는 기법이다.
② 프로세스들이 마감 시간 내에 처리되지 않으면 폐기되거나 처음부터 다시 실행해야 한다.
③ 기한부 스케줄링에 필요한 집약적 자원 관리는 많은 오버헤드를 일으킬 수 있다.
④ 동시에 다수의 기한부 작업이 수행되면 스케줄링은 매우 어려워 진다.
⑤ 사용자는 그 작업에 필요한 자원에 관한 정확한 정보를 시스템에 제시하여야 한다.

9) 선점형 - 라운드 로빈(RR : Round-Robin)
① 시간 할당량이 크면 비 선점의 FIFO와 동일하다.
② 시간 할당량이 작으면 문맥 교환 수가 증가한다.
③ 시간 할당량이 작으면 오버헤드가 커지게 된다.
④ 적절한 응답 시간을 보장해주는 대화식 사용자에게 효과적
⑤ 동일한 시간을 사용하는 시분할 시스템에 효과적

10) 선점형 - SRT
① 작업이 끝나기까지 "남아 있는" 실행 시간의 추정치가 가장 작은 프로세스를 먼저 실행하는 방식
② 서비스 받은 시간을 기록해야 하기 때문에 오버헤드 증가
③ 평균 대기 시간과 대기 시간의 분산(편차의 제곱)도 크다.
④ 임계치(threshold value)를 사용

11) 선점형- 다단계 피드백 큐(MFQ : Multi level Feedback Queue)
① 선점형 방식이다.
② 짧은 작업이나 입출력 위주의 작업에 우선권을 부여하기 위해 개발된 방식이다.
③ 큐(대기 리스트)가 여러 개이며 우선 순위가 있다.
④ 각 큐마다 시간 할당량(quantum)이 존재하며 낮은 큐일수록 시간 할당량은 커진다.
⑤ 각각의 큐들은 종속적으로 연결되어 있다.
⑥ CPU를 시간 할당량만큼 사용한 프로세스는 낮은 큐로 이동된다.
⑦ 맨 마지막 단계의 큐는 RR(라운드 로빈 스케줄러)를 사용한다.

12) 혼합형 - 다단계 큐(MQ : Multilevel Queue)
① 선점형, 비 선점형 방식이다.
② 대기 리스트(큐)를 특성 별로 여러 개를 갖는다.
③ 대기 리스트에 우선 순위가 있다.
④ 대기 리스트마다 독립적인 스케줄링을 갖는다
⑤ 대기 리스트간에 프로세스가 이동이 안 된다.
⑥ 우선 순위가 가장 높은 큐에서는 선점형으로 사용된다.

Section 4 병행 프로세스

1. 병행 프로세스

1) 임계 구역(Critical Section, 위험지구) 정의
다중 프로그래밍 기법에서 두 개 이상의 프로세스가 운영될 때 서로 공유하게 되는 자원(CPU, 메모리, 디스크, I/O장치,...) 등을 말한다.

2) 임계 구역의 원칙
① 두 개 이상의 프로세스가 동시에 사용할 수 없다.

② 순서를 지키면서 신속하게 사용한다.
③ 하나의 프로세스가 독점하게 해서는 안 된다.
④ 사용 중에 중단, 무한반복 되어서도 안 된다.

3) 상호 배제(Mutual Exclusion)
임계 구역(공유 자원)을 어느 시점에서 단지 한 개의 프로세스만이 사용할 수 있도록 하며, 다른 프로세스가 현재 사용중인 임계 구역 (공유 자원)에 대하여 접근하려고 할때 이를 금지하는 행위를 상호배제라고 한다.

4) 세마포어(Semaphore)
E. J. Dijkstra가 제안한 방법으로 반드시 상호 배제의 원리가 지켜져야 하는 공유 영역에 대하여 각각의 프로세스들이 접근하기 위하여 사용되는 두 개의 연산 P와 V라는 연산을 통해서 프로세스 사이의 동기를 유지하고 상호 배제의 원리를 보장하는 알고리즘을 세마포어, 해당 변수를 세마포어 변수 S라고 한다.

5) 세마포어 종류
① 이진 세마포어 : 세마포아 변수가 오직 0과 1값을 가지며 하나의 임계구역(공유자원)만을 상호배제하기 위한 알고리즘으로 인터럽트 봉쇄, 엄격한 교대, TSL 등이 있다.
② 산술(계수형) 세마포어 : 세마포아 변수가 0과 양의 정수를 값으로 가지며 임계 구역을 여러 개 관리하기 위한 상호 배제 알고리즘이다.

6) 세마포어의 특징
① E. J. Dijkstra가 제시한 상호 배제를 위한 알고리즘이다.
② 상호배제의 원리를 보장하는데 사용된다.
③ 여러 개의 프로세스가 동시에 그 값을 수정하지 못한다.
④ 세마포어에 대한 연산(오퍼레이션)은 처리 중에 인터럽트 되어서는 안 된다.
⑤ 세마포어에 대한 연산은 소프트웨어나 하드웨어로 구현 가능하다.
⑥ 이진 세마포어는 오직 0과 1의 두 가지 값을 가지며, 산술 세마포어는 0과 양의 정수를 값으로 가질 수 있다.
⑦ 세마포아 알고리즘은 프로세스 사이의 동기를 유지하게한다.
⑧ V 조작은 블록 큐에 대기 중인 프로세스를 깨우는 신호(wake-up)로서, 흔히 signal 동작이라 한다.
⑨ P 조작은 영계 영역을 사용하려는 프로세서들의 진입여부를 결정하는 조작으로, 흔히 wait 동작이라 한다.

7) 모니터(Monitor)
① 모니터 내의 자원을 원하는 프로세서는 반드시 해당 모니터의 진입루(entry)를 호출해야 한다.
② 모니터 외부의 프로세스는 모니터 내부의 데이터를 직접 액세스 할 수 없다.
③ 자료 추상화와 정보 은폐(Information hiding)의 개념을 기초적으로 사용한다.
④ 스위치 개념을 사용하여 한 순간에 하나의 프로세스만이 모니터에 진입할 수 있다.
⑤ 모니터에서 사용되는 연산은 wait와 signal이 있다.
⑥ 모니터의 경계에서 상호 배제가 시행된다.

2. 교착상태(Dead-Lock)

1) 교착상태(Dead-Lock)의 정의
복수의 프로세스(process)가 가능하지 못한 상태를 무한정 기다리고 있는 상태를 말한다. 두 개 이상의 프로세스가 하나의 자원을 공유하여 사용하고 있을 때 서로가 사용중인 자원을 요구하지만 요구를 영원히 들어줄 수 없는 상태를 말한다.

2) 교착상태 발생 필수 4대 요소
① 상호 배제(Mutual Exclusion)
② 점유와 대기(Hold & Wait)
③ 비 선점(Non Preemption)
④ 순환 대기(Circular Wait, 환형 대기)

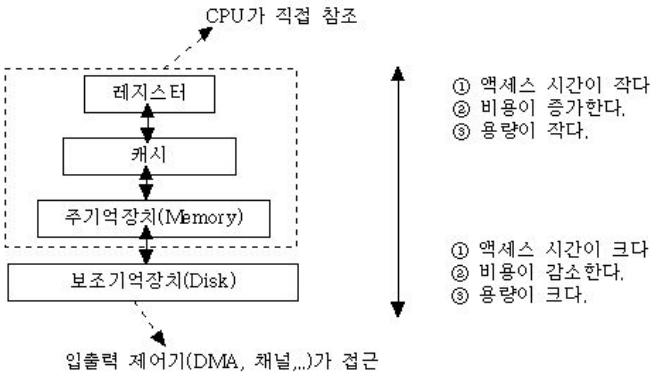
3) 교착상태 예방(Prevention)
① 상호 배제 부정
② 점유와 대기 부정(대기 제거)
③ 비 선점 부정(선점 인정)
④ 순환 대기 부정(순환 상태를 선형 대기 상태로 변환)

4) 교착상태 회피(Avoidance)
프로세스가 자원을 요구할 때 시스템이 안전 상태를 유지할 수 있는 프로세스의 자원 요구만을 할당하여 주는 방안으로 자원 분배를 교착상태가 발생하지 않는 범위 내에서 하는 방안이다. 교착 상태 회피 방안으로 사용하는 알고리즘에는 Dijkstra가 제안한 기법인 은행원(Banker's) 알고리즘이 가장 대표적이다.

Section 5 기억장치 관리

1. 기억장치의 계층 구조

1) 기억장치의 계층 구조



2, 기억장치의 관리 기법

1) 주기억장치(실기억장치) 사용 방식
① 고정 분할(정적 분할)
주기억장치의 크기를 다르게 분할하되 항상 고정된 크기를 갖는 형태로 분할하는 방식
② 가변 분할(동적 분할)
프로그램의 크기에 따라 주기억장치 분할 크기를 동적으로 분할하는 방식이다.

2) 단편화(fragmentation)
주기억장치 상에서 빈번하게 기억장소가 할당되고 반납됨에 따라 기억장소들이 조각들로 나누어지는 현상

① 내부 단편화
이미 정해진 크기에 프로그램을 할당하고 남은 기억 공간으로 사용되지 못하는 공간이다.
② 외부 단편화
정해진 크기는 아니지만 프로그램의 크기가 커서 기억할 수 없게 된 기억 공간이다.

3) 기억장치 재사용 기술
① 통합(coalescing) : 인접한 공백(기억 공간)들을 더 큰 하나의 공백으로 만든다.
② 집약(압축, compaction, garbage-collection) : 서로 떨어져 있는 여러 개의 낭비 공간을 모은다.

4) 가상 기억장치 사용 - 페이지 기법(Paging)의 특징
① 페이지 기법은 블록(block)이 고정적이다.
② 페이지 크기가 작을 수록 더 많은 페이지 사상 테이블이 존재한다.
③ 페이지 크기가 작을 수록 내부 단편화는 줄어들게 된다.
④ 페이지 크기가 작을 수록 자주 사용하는 페이지의 집합(working set)을 효율적으로 운영할 수 있다.
⑤ 페이지 크기가 작을 수록 특정한 참조 지역성만을 포함하기 때문에 기억장치 효율이 좋을 수 있다.
⑥ 페이지 크기가 클수록 페이지 테이블의 크기가 작아지므로 주기억 장치의 공간이 절약된다.
⑦ 페이지 크기가 클수록 참조되는 정보와는 무관한 많은 양의 정보가 주기억장치에 남게 된다.
⑧ 페이지 크기가 클수록 페이지 테이블이 복잡하지 않으므로 관리가 용이하다.
⑨ 하나의 페이지 크기가 크면 디스크로부터 I/O 전송에 소모되는 시간은 커지게 된다.
⑩ 페이지 크기가 작으면 총 출입력 시간은 늘어난다.
⑪ 내부 단편화만 발생한다.

5) 가상 기억장치 사용 - 세그먼트 기법(Segmentation) 특징
① 세그먼테이션 기법은 블록(block)이 가변적이다.
② 프로그램을 세그먼트화 하는 근본적인 이유는 메모리 절약을 위해 사용한다.
③ 세그먼테이션 기법에서는 기억장치 보호 키(storage protection key)가 필요하다.
④ 외부(External) 단편화만 발생한다.

6) 가상 기억장치 용어
① 오버레이(Overlay)
사용하지 않는 프로그램의 부분을 보조기억장치로 옮겨와서 이제 더 이상 필요하지 않는 프로그램 부분이 사용하고 있던 장소를 다른 프로그램이 사용하게 한다.

② 스와핑(Swapping)
작업의 모든 부분들이 동시에 주기억장치에 상주해 있을 필요가 없다. 이때 작업을 분할하여 필요한 부분만 교체하는 방법을 말한다.

③ 페이지 부재(Page Fault)
가상 기억장치 시스템에서 가상 페이지 주소를 사용하여 데이터를 접근하는 프로그램이 실행될 때, 프로그램에서 접근하려고 하는 페이지가 주기억장치에 있지 않은 경우 발생하는 현상을 페이지 부재(Page Fault)라고 한다.

④ 스래싱(Thrashing)
동시에 여러 개의 작업이 수행되는 다중 프로그래밍 시스템 또는 가상 기억 장치를 사용하는 시스템에서 하나의 프로세스가 작업 수행 과정에서 수행하는 기억 장치 접근에서 지나치게 페이지 폴트가 발생함으로 인하여 전체 시스템의 성능이 저하되는 현상

⑤ 구역성(Locality)
프로그램이 실행할 때 기억장치 내의 모든 정보를 균일하게 참조하는 것이 아니라 어느 한 순간에 특정 부분을 집중적으로 참조하는 프로그램의 순차적인 성질로 한번 호출된 자료나 명령은 곧바로 다시 사용될 가능성을 말한다.
㉠ 시간 구역성 : 반복(looping), 부 프로그램(subroutine, 서브루틴), 스택(stack), 집계(counting, totaling) 변수
㉡ 공간 구역성 : 배열 순회(array traversa), 순차적 코드, 관련된 변수들을 서로 근처에 선언

⑥ 작업 집합(Working Set) : 자주 참조되는 페이지의 집합으로 주기억장치에 고정 배치하여 교체 대상에서 제외함으로 교체 성능을 높이는 방

법이다. 작업 집합은 언제든 변경 될 수 있다.

3. 기억장치의 관리 전략

1) 기억장치 관리 전략
① 반입 전략 : 프로그램/데이터를 주기억장치로 가져오는 시기를 결정하는 전략 → When
② 배치 전략 : 프로그램/데이터의 주기억장치 내의 위치를 정하는 전략 → Where
③ 교체 전략 : 주기억장치 내의 빈 공간 확보를 위해 제거할 프로그램/데이터를 선택하는 전략 → How, What

2) 배치(Placement)전략
① 최초 적합(First Fit) : 입력된 작업을 주기억장치 내에서 그 작업을 수용할 수 있는 첫 번째 공백에 배치
② 최적 적합(Best Fit) : 입력된 작업을 주기억장치 내의 공백 중에서 그 작업에 가장 잘 맞는 공백에 배치
③ 최악 적합(Worst Fit) : 입력된 작업을 주기억장치 내에서 가장 잘 맞지 않는 공백, 즉 가장 큰 공백에 배치

3) 페이지 교체 전략 - 최적화(optimal replacement)
① 페이지 사용 횟수를 정확히 예측하여 교체하는 방법이다.
② 앞으로 가장 오랫동안 사용되지 않을 페이지와 교체한다.
③ Belady의 알고리즘으로 가장 이상적이지만 실현가능성이 희박하다.
④ 페이지 부재(Page Fault) 횟수가 가장 적으므로 Hit율이 가장 크다.

4) 페이지 교체 전략 - FIFO(First Input First Output)
① 주기억장치 내에 가장 오래 있던 페이지와 교체한다.
② 알고리즘이 가장 간단하다.
③ Page 교체가 가장 많다(Page Fault가 가장 많이 발생한다).
④ 프로세스에 할당된 페이지 프레임 수가 증가하면 페이지
부재의 수가 감소하는 것이 당연하지만 페이지 프레임 수가
증가할 때, 현실적으로 페이지 부재가 더 증가하는
모순(anomaly) 현상이 발생한다(Belady 모순).

5) 페이지 교체 전략 - LRU(Least Recently Used)
① 사용(참조된)한 지 가장 오래된 페이지를 대체 대상으로 선정한다.
② 현시점에서 가장 오랫동안 사용하지 않은 페이지와 교체한다.
③ 각 페이지마다 계수기(시간 기억 영역)를 두어 사용하는 기법

6) 페이지 교체 전략 - LFU(Least Frequence Used)
① 사용(참조된)한 횟수가 가장 적은 페이지와 교체한다.
② 사용한 횟수를 기억할 참조 변수를 각 페이지에 두어 사용한다.

7) 페이지 교체 전략 - NUR(Not Used Recently)
① 최근에 사용(참조)하지 않은 페이지를 제거한다.
② 하드웨어 비트 - 참조 비트(referenced bit)와 변형 비트(modified bit)를 사용한다.

8) 페이지 교체 전략 - PFF(Page Fault Frequency)
워킹 셋에 존재하는 페이지들을 관찰하여 최근에 자주 사용되고 있지 않은 페이지와 워킹 셋에 속하지 않은 페이지 중에 최근에 자주 사용하는 페이지와 교체한다.

9) 페이지 교체 전략 - Second Chance (FIFO의 2차 기회 부여)
각 페이지에 프레임을 FIFO 순으로 유지시키면서 LRU 근사 알고리즘처럼 참조 변수 갖게 한다

4. 디스크 관리

1) 디스크 접근 시간
① 탐색 시간(Seek time) : 디스크 상의 원하는 데이터를 액세스하기 위해 트랙 또는 실린더에 헤드를 위치시키는데 걸리는 시간
② 회전 지연 시간(Latency time, rotational delay, RPM 지연 시간) : 지정된 트랙에 위치한 헤드가 원하는 섹터에 위치시키는데 걸리는 시간
③ 전송 시간(Transmission time) : 디스크로부터 주기억장치로 데이터가 이동하는 시간

2) 디스크 스케줄링 전략의 목적
① 처리량을 최대화한다.
② 응답 시간을 최소화한다.
③ 응답 시간의 편차를 최소화한다.

3) 디스크 스케줄링 - FCFS(FIFO)
① 도착 순서에 따라 실행 순서가 고정된다는 점에서 공평하다.
② 요청한 순서대로 진행하기 때문에 순서가 바뀌는 일이 없다.
③ 디스크의 부하가 적을 때 유리하다.
④ 디스크의 부하가 커지면 응답 시간이 길어진다.
⑤ 탐색 시간(Seek time)을 최적화하려는 시도가 없다.

4) 디스크 스케줄링 - SSTF
① 대기 큐에 먼저 들어와 있지 않더라도 탐색 거리가 짧으면 먼저 서비스 받는다.
② 가운데 트랙이 안쪽이나 바깥쪽보다 서비스 받을 확률이 높다.
③ 헤드에서 멀리 떨어진 요청은 기아 상태(Starvation state)가 발생할 수 있다.

④ FCFS보다 처리량이 많고 평균 응답 시간이 짧다.
⑤ 처리량이 많기 때문에 일괄처리에는 유용하다.
⑥ 응답 시간의 편차가 크기 때문에 대화형 시스템에는 부적합하다.

5) 디스크 스케줄링 - SCAN의 특징
① SSTF의 응답 시간의 편차를 개선 기법이다.
② 진행 방향상의 가장 짧은 거리에 있는 요청을 먼저 수행한다.
③ 진행 방향으로 끝까지 진행한다.
④ SSTF에 발생하는 안쪽과 바깥쪽의 차별대우를 최소화하고 응답 시간의 편차를 줄인다.
⑤ 대부분의 디스크 스케줄링 전략의 기본이 된다.
⑥ 부하가 적은 경우에는 SCAN 기법이 가장 좋은 결과를 가진다.

6) 디스크 스케줄링 - C-SCAN 특징
① 안쪽과 바깥쪽의 차별 대우를 모두 없앴다.
② 항상 바깥쪽에서 안쪽으로 움직이면서 서비스한다. (반대는 서비스하지 않는다).
③ 안쪽 방향으로 끝가지 진행한다.
③ 응답 시간의 편차가 매우 작다.
④ 부하가 많은 경우에는 C-SCAN 기법이 가장 좋은 결과를 가진다.

7) 디스크 스케줄링 - LOOK과 C-LOOK
SCAN과 C-SCAN에서 헤드를 디스크의 끝가지 이동하지 않는 방법

8) 디스크 스케줄링 - N-Step SCAN
어떤 방향의 진행이 시작될 당시에 대기 중이던 요청들만 서비스하고,
진행 도중 도착한 요청들은 한데 모아져서 다음의 반대 방향 진행 때
최적으로 서비스 할 수 있도록 배열된다.

10) 디스크 스케줄링 - 에센바흐
탐색 시간, 회전 지연 시간, 전송 시간을 모두 고려해서 개발된 스케줄링

5. 파일 제어 블록 (FCB : File Control Block, File descriptor)
① 파일명 ② 보조 기억장치의 파일 위치
③ 파일의 구조 - 순차 파일, 직접 파일, 색인 파일,..
④ 보조 기억 장치 유형 - 디스크, 테이프,..
⑤ 접근(Access) 제어 정보
⑥ 파일 유형 ⑦ 제거 시기 ⑧ 생성 날짜, 시간
⑨ 제거 날짜 ⑩ 최종 수정 날짜 ⑪ Access 횟수

Section 6 운영체제의 실제

1. Unix 운영체제

1) Unix의 특징
① 상당 부분 C 언어를 사용하여 작성되었으며, 이식성이 우수하다. (어셈블리어 10%, C언어 90%).
② 사용자는 하나 이상의 작업을 백그라운드에서 수행할 수 있어 여러 개의 작업을 병행 처리할 수 있다.
③ 두 사람 이상의 사용자가 동시에 시스템을 사용할 수 있어 정보와 유틸리티들을 공유하는 편리한 작업 환경을 제공한다.
④ 소스를 누구나 볼 수 있도록 설계된 개방형 시스템(Open System)이다. (MS-DOS, MS-Windows는 Close System).
⑤ 표준이 정해져 있고 제품의 공급 업자가 많다.
⑥ 라이센스 비용이 저렴하다.
⑦ 커널의 크기가 비교적 작아서 이식성이 뛰어나다.
⑧ Multi-Tasking : 여러 개의 프로그램을 동시에 수행할 수 있다.
⑨ Multi-User : 여러 사용자가 동시에 사용할 수 있다.
⑩ 풍부한 네트워킹 기능이 존재한다.
⑪ 계층적(트리 구조)의 파일 시스템이다.
⑫ 사용자 위주의 시스템 명령어가 제공된다.
⑬ 쉘(Shell) 명령어 프로그램이 제공된다.

2) Unix의 기본 구성

① 커널(Kernel)
핵심 루틴으로, 하드웨어 보호기능, 사용자 서비스 제공, 프로세스 관리, 메모리 관리, 네트워크(통신)관리, 입출력 관리, 파일 관리 기능 등을 제공한다.

② 쉘(Shell)
사용자 명령의 입력을 받아 시스템 기능을 수행하는 명령 해석기로서 사용자와 시스템간의 인터페이스를 담당한다. 즉, 사용자와 커널 사이에서 중계자 역할을 한다. 또한 여러 가지의 내장 명령어를 가지고 있다.

③ 유틸리티(Utility)
문서 편집기, 데이터베이스 관리, 컴파일러, 네트워크(통신) 응용,..

3) Unix의 파일 시스템(I-node(Index node)의 항목)
① UID : 사용자 ID (파일 소유자의 식별 번호)
② GID : 그룹 ID (파일 소유 그룹의 식별 번호)
③ Protection : 파일 보호 모드(rwx - rwx - rwx), 보호 비트
④ 블록 주소 : 디스크의 실제 주소, (직접블록 지정, 간접주소 지정, 이중, 삼중 간접 블록 지정 방식)
⑤ 파일의 크기
⑥ 처음 생성 시기(파일이 만들어진 시간)
⑦ 마지막 사용 시기(파일을 최후로 접근(access)한 시간)
⑧ 최종 수정 시기(파일의 최종 수정 시간)
⑨ 파일 링크 수
⑩ 파일 속성(타입) : 정규(-), 디렉토리(d), 소켓(s), 장치(l, c, b)

4) Unix의 주요 명령어
① chmod : 파일의 속성을 및 Protection을 변경
② creat : 파일을 생성하거나 다시 기록한다.
③ dup : open FCB 복사한다.
④ fsck : 파일 시스템을 일관성 있게 검사하고 대화식으로 복구하는 명령
⑤ mkfs : 파일 시스템을 구성한다.
⑥ open : FCB(File Control Block)을 연다.
⑦ mount : 파일 시스템에 새로운 파일 시스템을 서브디렉토리에 연결할 때 사용
⑧ cp : 파일을 복사한다.
⑨ mknod : 특수 파일을 만드는 명령
⑩ exit : 프로세스를 끝마친다.
⑪ fork : 시스템 콜(Call) 중에서 새로운 프로세스를 생성, 복제하는 기능
⑫ getpid : 프로세스명, 그룹명, 부프로세스(getppid)의 정보를 얻는다.
⑬ exec : 새로운 프로그램을 수행시키기 위한 시스템 호출
⑭ wait : 자식 프로세스를 멈추거나 끝내기를 기다린다.
⑮ signal : 신호를 받았을 때 프로세스가 할 일을 지정한다.
⑯ preemption : 프로세스의 자원 사용 권한을 선점한다.
⑰ & : 백그라운드 작업 지시
⑱ pipe : 프로세스간에 통신 경로를 만들어 프로세스간 정보 교환이 가능하도록 한다.
⑲ finger : 로그인하고 있는 사용자의 정보를 표시하는 명령
⑳ cat : 파일에 내용을 화면에 출력한다.
(21). ls : 파일의 목록을 보여준다.
(22). make : 프로그램을 컴파일 한다.

2. PC 운영체제
1) MS-DOS의 특징
① Microsoft사에서 개발한 소규모 운영체제이다.
② Stand alone : 단일 작업용 시스템 방식이다.
③ Single-Tasking : 한 번에 한 개의 프로그램만을 실행할 수 있다.
④ Single-User : 한 명의 사용자만 사용할 수 있다.
⑤ Text 기반 인터페이스 방식이다.
⑥ 디렉토리 구조 : 계층형 트리 구조
⑦ 디스크 공간 할당 : 블록 단위 파일 사상 기법

2) MS-DOS의 파일 시스템
① MSDOS.SYS
파일의 입출력 시스템의 호출을 담당하며 디스크 상의 파일 처리, 메모리 관리, 프로세스 관리 등을 담당하며 Unix에서 커널(Kernel)과 유사하다.
② IO.SYS
MSDOS.SYS의 입출력 요구에 따라서 실제의 입출력 처리를 담당한다.
③ COMMAND.COM
사용자가 입력한 명령어를 해석하고 실행한다. Unix에서 쉘(Shell)과 같은 역할을 한다.
④ CONFIG.SYS
부팅시에 실행되며 시스템의 환경 설정, 메모리 관리자 설치.
⑤ AUTOEXEC.BAT
부팅시에 우선적으로 실행할 프로그램을 모아놓은 파일.

3) MS-DOS의 명령어
① 내부 명령어
메모리 상주, 파일로 존재하지 않는다. 용량이 작다. 빠르다.
㉠ DIR : 파일 목록을 출력한다. (Unix의 ls)
㉡ COPY : 파일을 복사한다.(Unix의 cp)
㉢ TYPE : 파일의 내용을 출력한다. (Unix의 cat)
㉣ CD : 디렉토리를 변경한다. (Unix의 cd)
② 외부 명령어
디스크에 존재. 실행 파일로 존재, 용량이 크다, 느리다.
㉠ ATTRIB : 파일의 속성을 변경한다. (Unix의 chmod)

㉡ FORMAT : 디스크를 초기화한다.
㉢ FDISK : 하드디스크를 분할(파티션)하거나 삭제, 변경한다.
㉣ SYS : 메모리에 있는 시스템 파일을 디스크로 복사한다.
㉤ FIND : 파일을 찾는 명령
㉥ RAMDRIVE : 메모리 일부를 디스크처럼 사용하게 해준다.
(가상 디스크).

4) MS-Windows 특징
① GUI : 그래픽 지원 인터페이스
② 32Bit 명령어 형식
③ 가상 기억장치 사용
④ OLE(객체 삽입 기능) : 응용 프로그램간의 자료 공유
⑤ 선점형 Multi-Tasking
⑥ Plug & Play (PNP 기능) : 설치 자동 셋팅
⑦ VFAT : 가상 FAT 예) 파일명 255자
⑧ 네트워킹(Networking) 기능
⑨ 멀티미디어 지원
⑩ 폴더, 휴지통
⑪ Windows 95 : Single-User / Multi-Tasking
⑫ Windows NT : Multi-User / Multi-Tasking

Section 7 C 언어

1. C언어의 기초

(1) C 언어 기본 프로그래밍
① 국어, 영어, 수학 점수 표준입력 받아 총점, 평균 표준 출력

```
#include <stdio.h>
main()
{
int kor, eng, mat, tot;
float  ave;
scanf("%d%d%d",&kor,&eng,&mat);
tot = (kor + eng + mat) / 3.0 ;
ave = (kor + eng + mat) / 3.0 ;   /* 혹은 ave = tot / 3.0;  */
printf("총점 = %5d  평균 = %5.1f\n", tot, ave);
}
```

② 두수의 정수를 표준 입력 받아 큰 값을 표준 출력하기

```
#include <stdio.h>
main()
{
int  value1, value2,  max;
scanf("%d%d",&value1,&value2);
if(value1 >= value2)
max =value1;
else
max =value2;
printf("큰값 = %5d\n", max);
}
```

(2) C 언어의 특징
- 이식성, 효율성이 높은 언어 -> 시스템 프로그래밍 언어
- 구조적 프로그래밍이 가능
- 고급언어이면서 저급언어 프로그래밍도 가능
- 자료의 주소를 조작할 수 있는 포인터를 제공
- 컴파일러 방식의 언어
- 항상 main( )이라는 함수로부터 실행이 시작
- 주석문은 /* ~ */ 출제 : 주석은 소스 프로그램에 간단한 설명
- 문장을 끝마칠 때 " ; " 이 필요하다.

(3) 변수명 규칙
- 영문자(_ 포함) 첫 번 째, 숫자가 첫 번째 올 수 없다.
- 영문 대소 문자 구분, 예약어를 사용 할 수 없다.
- 공백이나 특수 문자 사용할 수 없다.

(4) 예약어 (Key word) 출제 : C언어는 예약어가 없다. 틀림
- 프로그램 판독성 증가 출제 : 판독성은 감소, 틀린 답
- 변수이름으로 사용할 수 없음
- 신뢰성 향상, 번역 과정 속도 향상

2. C언어의 자료형
출제 : 문자 자료형은? 정수형?

(1) C 언어의 기본 자료형 출제 : 자료 형이 아닌것 ?

의미	예약어	대형기종	소형기종
문자형	char	1	1
	short	2	2
	int	4	2
	long	4	4
	unsigned	4	4
실수형	float	4	4
	double	8	8

- C언어에서는 Integer, character, real 사용하지 않음
- 상수 : 65, 0x41, 0101, 'A', 34.56, 3e+02, "ABC"

(2) 기억 클래스
- 자동 변수 : (auto) int kor;
  스택(Stack) 영역에 할당, 블록이나 함수 내부 선언,
  실행시 확보(동적할당), 블록이나 함수 종료시 소멸
- 정적 변수 : static int kor;
  힙(heap) 영역에 할당, 선언 이후 영역에 영향,
  번역시 할당(정적할당), 프로그램 종료시 소멸
- 레지스터 변수 : register int kor;
  CPU 내의 고속메모리에 할당, 고속 처리 용도
  번지 지정 연산자 &를 사용할 수 없다.
- 외부 변수 : extern int kor ;
  힙(heap) 영역에 할당, 파일과 상관없이 프로그램 전 영역에
  영향, 번역시 할당, 프로그램 종료시 소멸
  출제 : 종류가 아닌것? 선언시 생략하면 어떤 변수로 간주?

3. 입출력 함수
(1) 입출력 함수
한 문자 표준 입출력 : getchar(), putchar()
문자열 표준 입출력 : gets(), puts()
포멧 문자열 표준 입출력 : scanf(), printf()

한 문자 파일 입출력 : getc(), putc()
문자열 파일 입출력 : fgets(), fputs()
포멧 문자열 파일 입출력 : fscanf(), fprintf()
출제 : 문자열 입력 함수는?

(2) 포멧 문자열의 데이터 변환 문자

%d : 10진 정수, %o : 8진 정수, %x : 16진 정수
%c : 문자, %s : 문자열
%f : 실수형, 부동 소수점, %e : 배정도 실수형

(3) 이스케이프 시퀀스 출제 : 설명이 틀린것? 영어로
\n : New line , \r : Carriage Return
\t : Tab , \b : Back space
\0 : NULL, \': '자체
\": "자체 \\ : Back Slash
\f : Form Feed

4. 연산자
(1) 연산자 우선 순위 출제 : 단항 연산가가 아닌것? and

연산자	종류	결합	혼용
	(), [], ->	->	높음
단항	++, --, -, !, ~, &, *, sizeof	<-	
산술	*, /, %		
	+, -		
시프트	<<, >>		
관계	<, >, <=, =>	->	
	==, !=		
비트논리	&, |, ^		
논리	&&, ||		
조건	? :		
할당	\, +=, -=, *=, /=, %=, <<=,..	<-	
콤마	,	->	낮음

(2) 혼합 연산시 결정 자료형
char < short < int < unsigned < long < float < double

선언	수식	결과
char c; **short s;** **int i** **unsigned u;** **long l;** **float f;** **double d;**	c-s/i	int 형
	u*3-i	unsigned 형
	d*3.0+i	double 형
	f*3-i	float 형
	c*2	int 형
	d+2.0	double 형
	3*s*l	long 형

(3) 혼합 연산시 확장과 축소
① 확장(widening) 출제 : widening이란 ?
정수형에서 실수형 혹은 배정도로 변환될 때 손실이 없다.
② 축소(narrowring)
실수형을 정수형으로 변환할 때 손실이 있다.

```
char c;   int i;
i=c;  /* c의 비트폭으로  확장한 후 대입한다.*/
short int s;   long int l;
s=l;  /* 상위 비트를 잘라버린다.*/
int i;   float f=12.3;
i=f;  /* 소수점 이하를 잘라버린다.  */
float f;   double d;
f=d;  /* 유효 숫자를 맞추어 반올림을 한다. */
```

(4) 주요 연산자 예제
① ++ 증감 연산자

```
#include <stdio.h>
main() {
        int a, b, c;
        a=b=c=0;
        a = ++b + ++c;
        printf("\n %d %d %d", a, b, c);    /* 2 1 1 출력 */
        a = b++ + c++;
        printf("\n %d %d %d", a, b, c);    /* 2 2 2 출력 */
        a = b++ + c++;
        printf("\n %d %d %d", a, b, c);    /* 4 3 3 출력 */
        a = b-- + --c;
        printf("\n %d %d %d", a, b, c);    /* 5 2 2 출력 */
}
```

② % : 나머지 연산자

```
#include <stdio.h>
main() {
 printf("9 mod 4 = %d \n", 9%4);  /* 1이 출력 */
 printf("10 mod 4 = %d \n", 10%4);  /* 2이 출력 */
 printf("11 mod 4 = %d \n", 11%4);  /* 3이 출력 */
 printf("12 mod 4 = %d \n", 12%4);  /* 0이 출력 */
}
```

③ 논리 연산자, 관계 연산자
출제 : "A와 B와 같지 않다"는 ? A!=B

선언	식	동등한 식	값
char c= 'w' ; int i=1; int j=2; int k=-7; double x=7e+33 double y=0.001;	'a'+1 < c	('a'+1) < c	1(참)
	-i-5*j>=k+1	((-i)-(5*j)) >= (k+1)	0(거짓)
	3<j<5	(3<j)<5	1(참)
	'V'==c-1	'V'==(c-1)	1(참)
	i+j+k==-2*j	((i+j)+k)==((-2)*j)	1(참)
	k==j-9==i	(k==(j-9))==i	1(참)
	x+x!=x*y	(x+x)!=(x*y)	1(참)

④ 시프트 연산자

```
 int a, b;
 a=2;
 b=a<<2;
```

a의 비트 표현 : 00000000 00000010 → 2
1번 시프트 : 00000000 00000100 → 4 = 2*2의 1승
2번 시프트 : 00000000 00001000 → 8 = 2*2의 2승

b=a<<2; → b = a * 2의 2승
b=a<<5; → b = a * 2의 5승
b=a>>3; → b = a * 2의 -3승

⑤ 비트 논리 연산자

연산	16진수	2진수
c	0x45	01000101
~c	0xBA	10111010

연산	16진수	2진수
c1	0x45	01000101
c2	0x71	01110001
c1 & c2	0x41	01000101

연산	16진수	2진수
c1	0x47	01000111
c2	0x53	01010011
c1 | c2	0x57	01010111

연산	16진수	2진수
c1	0x47	01000111
c2	0x53	01010011
c1 ^ c2	0x14	00010100

⑥ 논리 연산자의 절단

```
int x, y, z;
x=y=1;
z = ++x || ++y;              /* x=2, y=1, z=1 */

int x, y, z;
x=y=-1;
z = ++x && ++y;              /* x=0, y=1, z=0 */
```

⑦ ! 연산자

선언	식	동등한 식	값
char c= 'w' ; int i=7; int j=7; double x=0.0; double y=2.3;	!c	!c	0
	!i-j	(!i)-j	-7
	!-i-j	(!(-i))-j	-7
	x*!y	x*(!y)	0.0
	!x*!y	(!x)*(!y)	0
	x/!y	x/(!y)	error
	x/!!y	x/(!(!y))	0.0

5. 제어문
출제 : 문장 구조 성격이 다른것? while, for, if, do~while

(1)선택문

① 단일 if

```
if(a >= b)
    max = a;

if(a >= b)
{   max = a;
    flag = 1;   }
```

② 이중 if

```
if(a >= b)
    max = a;
else
    max = b;

if(a>=b)
 {   max = a;
     flag = 1;   }
else
 {   max = b;
     flag = 0;   }
```

③ 다중 if

```
if(jum >= 90)
    hak = 'A';
else if (jum >= 80)
        hak = 'B';
        else if (jum >= 80)
                hak = 'C';
                else if (jum >= 80)
                        hak = 'D';
                else
                        hak = 'F';
```

(2) switch ~ case ~ default

```
switch(jum / 10)
{
  case 10 :
  case  9 :  hak = 'A';   break;
  case  8 :  hak = 'B';   break;
  case  7 :  hak = 'C';   break;
  case  6 :  hak = 'D';   break;
  default  :  hak = 'F';
}
```

(3) 반복문 for

```
main()  {
int i;
int sum = 0;

for(i=1 ; i<=10 ; i++)
 {
    sum += i;
  }

printf("sum = %d \n", sum);
}
```

(4) 반복문 while, 반복문 do ~ while

```
:
int i=0, sum=0;
while(i < 10)
{
  i++;
  sum += i;
}
printf("%d",sum);


:
int i=0, sum=0;
do {
  i++;
  sum += i;
} while(i < 10);

printf("%d",sum);
```

Section 8 프로그램 언어론의 개념

1. 프로그램 언어론의 개념

(1) 프로그램 언어의 정의

기계가 읽을 수 있고, 인간이 읽을 수 있는 형태로 계산을 서술하기 위한 표기 체계

① 계산
산술계산 + 자료 조작 + 문서처리 + 정보 저장과 인출 등

② 기계가 읽을 수 있는(machine readable)
효율적인 번역, 모호함이 없고, 유한한 단계적 과정 복잡도는 최소화

③ 인간이 읽을 수 있는(human readable)
기계의 지식이 없는 사람도 이해하기 쉽도록, 추상화 제공 자연 언어화

(2) 프로그램 언어를 공부 해야 하는 이유

① 효과적인 알고리즘을 개발할 수 있는 능력 향상
② 현재 사용하고 있는 프로그래밍 언어의 이용 증진
③ 유용한 프로그래밍 요소들에 대한 어휘를 증진
④ 적합한 프로그래밍 언어를 선택
⑤ 새로운 언어를 쉽게 배울 수 있다.
⑥ 새로운 언어를 쉽게 설계 할 수 있다.

(3) 좋은 프로그래밍 언어의 요건
① 프로그래밍 언어의 개념이 단순, 명료하고 통일성이 있어야함
② 구문(syntax)가 명료해야 한다.
③ 응용 문제에 자연스럽게 적용할 수 있어야 한다.
④ 학습, 작성 용이성이 있어야 한다. 출제 : 보안 용이성은 아님
⑤ 프로그램 검증이 용이해야 한다.
⑥ 적절한 프로그래밍 작성 환경이 갖춰 있어야 한다.
⑦ 프로그램이 호환성이 있어야 한다.
⑧ 효율적이어야 한다.

(4) 프로그램 문서화의 목적
- 프로그램 개발 팀에서 운용 팀으로 인계 인수를 쉽게 할 수 있다.
- 프로그램 개발 중 추가 변경에 따른 혼란을 감소시킬 수 있다.
- 프로그램 개발 후 유지 보수가 용이하다.
- 오류 수정이나 확장 성이 좋다.
- 다른 프로그램에 적용 시 이식 성이 좋다.
출제 : 문서화의 목적이 아닌것은?

(4) 프로그래밍 패러다임
① 명령형 언어 - FORTRAN, COBOL, Pascal, C
② 객체 지향형 언어 - Smalltalk, C++, Java, Modula-3
③ 함수형 언어 - LISP, Scheme, ML, Haskell
④ 논리 언어 - Prolog

(5) 프로그래밍 영역
① 과학 응용 분야 - FORTRAN
② 사무 응용 분야 - COBOL
③ 인공 지능 분야 - LISP, Prolog
④ 시스템 프로그래밍 - Assembly, BLISS, C
⑤ 스크립트 언어 - AWK, Tcl/Tk, Perl, JavaScript
⑥ 특정 목적언어 - RPG : 사무 보고서를 생성하는데 사용
APT : 기계도구에 명령어를 입력하는데 사용
GPSS : 시스템 시뮬레이션을 위해 사용

Prolog 언어
```
like(kim, candy);
like(kim, juice);
like(kim, cake);
like(lee, candy);
like(lee, cake);
like(lee, kimchi);
? like(kim, T); <- 실행
T = candy;
T = juice;
T = cake ;
Yes.
```
인공지능, 데이터베이스 쿼리.. 등

(6) 세대별 언어
1세대 - 저급 언어 - 기계어, 어셈블리어
2세대 - 비구조적 고급 언어 - FORTRAN
3세대 - 구조적 고급 언어 - Pascal, C
4세대 - 사용자 중심 언어
비 절차적언어
자연어와 같은 대화 형식으로 표현 - SQL, Lotus, Delphi
초 고급 언어 - Prolog

(7) 세대별 언어의 특징
① 1950 년대 - 최초의 프로그래밍 언어
FORTRAN - 과학 계산용, 복소수 타입
COBOL - 사무 처리용, 레코드, 자료 구조를 실행 부분과 분리
Algol60 - 자유형식, 구조적 문장, begin ~ end, BNF 최초
LISP - 인공지능, 쓰레기 수집 처음 사용
APL - 함수적 형태, 반복 자동 수행, 많은 연산자

② 1960년대 - 프로그래밍 언어의 범람
PL/1 - FORTRAN + COBOL + Algol60의 장점
Algol68 - 일관성 있는 있는 구조 생성, Algol60개선, 직교적언어
SNOBOL - 최초의 스트링 처리 언어
Simula67 - 최초의 객체지향 언어, 클래스 개념 도입
BASIC - 교육용

직교적(직교성) : 독립적 성질, 중복 성 배제
/성별, 생년월일, 직업/ /HTML, Javascript, PHP/

③ 1970년대 - 단순성, 추상화 연구
Pascal - Algol 개선, 작고, 단순하고, 효율적, 구조적(대표), 교육
C - 단순성 추구, 복잡성 감소, 기계에 많은 접근(시스템 프로그램)
CLU - 추상화 메커니즘, 자료, 제어, 예외처리
Euclid - Pascal 확장, 추상 자료형 포함, 정형적인 검증
출제 : 시스템 프로그래밍에 적합한 언어 : C언어

④ 1980년대 - 새로운 방향과 객체 지향의 부상
Ada - 미 국방성 주도, 추상화, 정보은폐, I/O 기능 탁월, 대량 자료
Modula-2 - Pascal의 후속 언어, 시스템 프로그램, 모의 실험
Smalltalk - 일관성, 가장 순수한 객체 지향
C++ - C +방대한 라이브러리, 모든 플랫폼에 이식, 객체지향
Prolog - 논리형 언어

출제 : 미 국방성 주도, 데이터 추출, 추상화, 정보 은폐, 입출력 기능이 뛰어나서 대량 자료 처리에 적합한 언어는? Ada

⑤ 1990년대 - 통합, 인터넷, 라이브러리, 스크립트
Java - C++기반, 객체지향, WWW 페이지
Ada95 - 객체지향 + 병행성
Haskell - 순수 함수형 언어
라이브러리(library) - Java API
스크립트 언어(scripting language) - 특수 목적 언어
Visual Basic - 프로그래밍 언어 + 스크립트 언어 + 시각 지향

(8) 프로그래밍 언어의 설계 원칙
① 작성력-쉽게 사용할 수 있는가에 대한 척도
② 판독성-프로그래밍 언어의 질에 대한 중요한 척도
③ 효율성-목적코드, 번역, 구현성, 프로그래밍의 효율성
④ 신뢰성-예상치 못한 또는 위험한 실행을 방지
⑤ 일반성-광범위한 응용분야에 대한 적용성 예)
⑥ 직교성-구문이 문맥에 따라 다른 의미를 가지면 안된다. 예)
⑦ 획일성-구문의 외형과 조화에 초점을 둔다.
⑧ 단순성-간결성, 지나친 단순성은 표현력 부족
⑨ 표현력-복잡한 과정을 쉽게 표현할 수 있는가의 정도 예)
⑩ 확장성-새로운 기능의 첨가가 용이한 정도
⑪ 제약성-언어의 부분집합(최소한 구문 사용)을 정의하는 기능
⑫ 기존 표기법 관례와 일관성-표준화된 기능과 개념 많이 포함
⑬ 정확성-실행 예측이 명확한 정의가 존재하는가 의미
⑭ 기계 독립성-기계 구조 등의 상세한 부분이 포함되지 않은
⑮ 보안성-오류 발견 시 보고되기 쉽도록 하는 것

일반성
C에서 배열과 레코드는 등가 연산자 '=='를 이용하여 직접 비교할 수 없고 원소를 하나씩 비교해야 한다. 따라서 '=='연산자는 일반성이 부족하다.
직교성
C는 매개 변수 전달 방법에 직교성이 없다. C에서 배열을 제외한 모든 매개 변수는 값-전달이지만 배열만은 참조-전달이다.

```
class Example { … };      //클래스 선언, 세미콜론
int example {…}             // 메소드 선언, 세미콜론 없다,
                            function f : boolean;
                            begin
                                …
                             f := false;     // 리턴값이 배정문 처럼 사용
                             end;
```

표현력 : C언어 while(*s++ == *t++);

(8) 프로그래밍 언어의 평가 기준
판독성 : 단순성, 직교성, 제어문, 자료형과 구조, 구문 설계
작성력 : 단순성과 직교성, 추상화 지원, 표현력
신뢰성 : 형검사, 예외처리, 별칭, 판독성과 작성력
비용 : 교육비용, 작성 비용, 컴파일 비용, 실행 비용, 시스템 비용, 신뢰성 부족 비용, 유지보수 비용
이식성, 일반성, 명확성

Section 9 프로그램 언어의 구현 기법

2. 프로그래밍 언어의 구현 기법
(1) 저급 언어와 고급 언어

① 저급 언어
- 기계 중심의 언어, 전문 지식 필요
- 기계어 : 컴퓨터 직접 이해, 2진수(0,1), 호환성 부족
  전문적인 지식이 필요 , 속도 빠름 출제 : 호환성 없다.
- 어셈블리어 : 기계어와 1:1로 대응하는 기호, 번역과정 필요
  연상 기호 코드 식(Mnemonic Code) 언어

출제 : C언어는 Mnemonic Code 언어이다. 틀림

② 고급 언어
- 사람 중심의 언어, 호환성, 전문 지식이 없어도, 번역과정 필요
- 컴파일러 언어 출제 : 포트란, 코볼은 컴파일러 언어이다.
  FORTRAN, COBOL, ALGOL, PL/1, PASCAL, Ada, C
- 인터프리터 언어
  LISP, SNOBOL4, APL, Prolog, BASIC 등

(2) 번역 프로그램의 종류

원시프로그램	→	번역기	→	목적 프로그램
어셈블리어 프로그램	→	어셈블러	→	기계어 프로그램
고급언어 프로그램	→	컴파일러	→	어셈블리, 기계어
고급언어 프로그램	→	인터프리터	→	실행 결과
고급언어 프로그램	→	전처리기	→	고급언어 프로그램

출제 : 디버거 : 실행 오류를 찾기 위한 프로그램
출제 : 번역 기능이 없는 것은 ? 전처리기 (프리프로세서)

원시프로그램 → 전처리기 → 확장된 원시 프로그램 → 언어 번역기 → 목적 프로그램

원시프로그램 → A기종에서 수행되는 크로스 컴파일러 → B기종에 대한 목적 프로그램

출제 : 다른 기종에 맞는 기계어로 번역하는 것은?
크로스 컴파일러(Cross Compiler)

컴파일러 출제 : 순서

입력자료
↓
고급언어-컴파일러-목적모듈
고급언어-컴파일러-목적모듈 -> 링커 -> 로드모듈 -> 로더 -> 기계어
↓
실행결과

출제 : 기계어로 된 프로그램을 묶어서 로드 모듈을 만드는 것은?

인터프리터

입력자료
↓
고급언어 -> 인터프리터
↓
실행결과

하이브리드(hybrid) 구현 기법

컴파일러(번역기법) 과 인터프리터 (시뮬레이션) 기법 혼용
Visual Basic, Java 형태

입력자료
↓
고급언어 -> 적당한 번역과정 -> 중간 코드형태 -> 인터프리터
↓
실행결과

(3) 번역기법(컴파일러)과 인터프리터 기법 비교

	인터프리터	번역기법
실행시간	느리다. 출제	빠르다.
기억장소	적게 사용한다.	많이 사용한다.
대화식	용이하다.	복잡하다.
목적파일	생성하지 않는다.	생성한다.

(4) 컴파일러의 논리적 구조

원시프로그램
↓
어휘 분석 I am a boy. -> I, am, a, boy, .
↓
구문 분석 주어 + 동사 + 보어 -> 2형식.
↓
의미 분석 I am a sky
↓
중간코드 생성 나는 이다 하나의 소년
↓
코드 최적화 나는 이다 소년
↓
목적코드 생성 나는 소년 이다. 출제 : 순서
↓
목적프로그램

① 어휘 분석
원시 프로그램을 하나의 긴 스트링으로 보고 원시 프로그램을 문자단위로 스캐닝하여 문법적으로 의미있는 일련의 문자들로 분할한다.

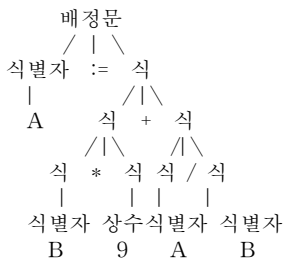
A := B * 9 + A / B;
[A][:=][B][*][9][+][A][/][B][;]
10개의 토큰

	구분	예
일반형태(사용자)	식별자	sum, tot
	상수	2011, 8.5E+01
특수형태(설계자)	예약어	begin, if, while
	연산자	*, +, = 등
	구분자	괄호나 따옴표

출제 : 의미있는 일련의 문자들로 분해하는 역할? 어휘 분석기

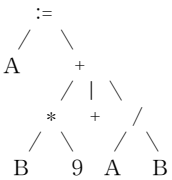
② 구문 분석 - 파싱, 파스(parse) 트리

A := B * 9 + A / B; 의 파스 트리



BNF를 이용하여 고급 언어로 작성된 프로그램을 구문 분석하여 문장을 문법 구조에 따라 트리 형태로 작성하는 것이다.
출제 : 위 내용의 설명은? 파스 트리

A := B * 9 + A / B; 의 파스 트리



Postfix(left-right-root) : AB9*AB/+:=

- 어떤 표현이 BNF에 의해 바르게 작성 되었는지 확인
- 파스 트리가 존재하면 BNF에 의해 작성 된다. 출제 : 작성될 수 없다.
- 트리의 모든 가지 터미널로 유도되어 가지치기가 끝난 상태
- 문법의 시작 기호로부터 적합한 생성 규칙을 적용할 때 가지치기 된다.

③ 의미분석
- 구문 트리를 보고 어떠한 의미와 기능을 하는지 분석
- 기능이 올바르게 수행 될 수 있도록 환경 조성
- 형검사

예)
실수가 배열의 첨자로 사용되었는지 검사, 정수의 실수 혼합 연산을
허용 시 정수를 실수로 바꾸어 주는 형 변환 작업을 수행한다.

④ 중간 코드 생성
구문 트리를 이용하여 적당한 명령어 생성

예)
A := B * 9 + A / B; 의 Quadruple
(*, B, 9, T0), (/, A, B, T1), (+, T0 , T1 , T2), (:= , T2 , Φ , A)

```
MUL    B,  9
DIV    A,  B
ADD    T0 , T1
MOV    T2 ,  A
```

⑤ 코드 최적화
지역 최소화

연산의 세기 경감	
x = y**2	x = y*y
a = b*3	a = b+b+b
q = q/5	q = q * 0.2

연산의 세기 경감		
x = 3	x = 3	x = 3
y = 2 * x	y = 2 * 3	y = 6

연산의 세기 경감	
a = b + 0	a = b
x = y * 1	x = y

⑥ 목적 코드 생성
- 연산을 수행할 레지스터를 선택
- 자료에 대한 기억장치들의 위치를 지정
- 실제 목적 기계어에 대한 코드 생성

(5) 컴파일러의 물리적 구조
① 1-패스 컴파일러
한 번에 번역, 개발이 어렵다, 실행 속도가 빠르다.
② 2-패스 컴파일러
두 번에 번역, 개발이 쉽다, 실행 속도가 느리다.

	내용
장점	이식성이 좋다. 기계의 독립적인 최적화가 가능하다. 기억공간 재사용으로 기억장소 절약
단점	기계 코드의 제한을 받는다. 실행 속도가 느리다.

(6) 형식 언어
어떤 알파벳에서 얻은 기호들로 구성되는 문자열의 집합

① 유한 집합 알파벳 T = {ㄱ, ㄴ, …, ㅏ, ㅑ,… ㅣ}
② 문자열이란 T = { 0, 1} 일 때, 문자열은 0, 1, 00, 01, 010,…
③ 문자열= ω, 문자열 길이= |ω | ω = korea 이면 |ω | = 5
④ 공문자열=ε, uε= εu=u, uεv=uv, a0 =ε, ω=abc이면 ω^R =cba
⑤ 문자열 접속, u=abc, v=010, u • v=uv=abc010(교X), |uv|=|u|+|v|
⑥ T={0} T* = {ε, 0, 00,…}, T^+ = {0, 00,…}
⑦ 알파벳 T에 대한 언어 L은 T^*의 부분 집합이다.
T={a, b} 일 때,
L1= {ε, a, b, aa, ab, aaa,…},
L2= {a, b, bb, ba, bbb,…},

L3= {a p | p는 소수},
L4= {a^n a^n | n≥1} 는 언어이다.

(7) 형식 문법
형식언어를 생성하기 위한 문법

문법 : G=(V_N , V_T , P, S)
V_N : Nonterminal 기호의 유한 집합 (V : Vocabulary)
V_T : terminal 기호의 유한 집합(VN ∩VT =Ø, VN ∪VT =V)
P : 생성 규칙의 집합, 각 생성 규칙의 형태는 α→β(α∈ T^+ β ∈ T^*)
S : VN에 속하는 기호로, 다른 Nonterminal들과 구별됨, 시작 기호

① α→β 에서 →는 단순 대치
② α⇒β 에서 ⇒는 생성 규칙 적용, *는 0이상, + 1번이상

G=({S, A}, {0, 1}, P, S)
P : S→0AS, S→0, A→S1A, A→10, A→SS
혹은
P : S→0AS | 0, A→ S1A | 10 | SS

S ⇒0AS (S→0AS 적용)
⇒010S (A→10 적용)
⇒0100 (S→0 적용)

(8) 촘스키 계층

문법		인식기	
Type0	Unrestricted G	Turing Machine	
Type1	Context-Sensitive G	Linear Bounded Automata	
Type2	Context-Free G	Push-down Automata	구문 분석
Type3	Regular G	Finite Automata	어휘 분석

Type 0 Unrestricted G (무제한 문법)
Type 1 Context-Sensitive G (문맥 - 감지 문법)
Type 2 Context-Free G (문맥 - 자유 문법) - 일반적인 기법
Type 3 Regular G (정규 문법)

Type 0 ⊃ Type 1 ⊃ Type 2 ⊃ Type 3

② Type 1 Context-Sensitive G (문맥 - 감지 문법)
생성 규칙 α→β에서 |α|≤ |β|

G1 ={S, A, C}, {a, b}, P, S)
P : S→A, A→aABC, A→abC, CB →BC, bB→bb, bC →bb

S ⇒ A 생성 규칙 S→A 적용
⇒ aABC 생성 규칙 A→aABC 적용
⇒ aabCBC 생성 규칙 A→abC 적용
⇒ aabBCC 생성 규칙 CB→BC 적용
⇒ aabbCC 생성 규칙 bB→bb 적용
⇒ aabbbC 생성 규칙 bC→bb 적용
⇒ aabbbb 생성 규칙 bC→bb 적용

∴ L(G1) = {a^n b^m |n, m ≥1} 는 문맥-감지 언어이다.

③ Type 2 Context-Free G (문맥 - 자유 문법) - 일반적인 기법
생성 규칙 A → α에서 α∈V *

G2 ={S}, {a, b}, P, S)
P : S→ab, S→aSb

S ⇒ ab
S ⇒ aSb ⇒ aabb
S ⇒ aSb ⇒ aaSbb ⇒ aaabbb

∴ L(G2) = {a n b n |n ≥1} 는 문맥-자유 언어이다.

④ Type 3 Regular G (정규 문법)
생성 규칙
우선형(right-linear) 문법 : A → tB, A → t
좌선형(left-linear) 문법 : A → Bt, A → t
단, A, B ∈ VN , t ∈ VT*
G3 ={S}, {a}, P, S)
P : S→a, S→aS

S ⇒ a
S ⇒ aS ⇒ aa
S ⇒ aS ⇒ aaS ⇒ aaa

∴ L(G3) = {a n |n ≥1} 는 정규 언어이다.

(9) 유한 오토마타(FA)

어떤 알파벳 T로부터 만들어지는 특정한 문자열들을 받아드리는 시스템의 수학적인 모델로서 변화할 수 있는 상태가 유한개인 것을 말한다. 즉, 특정 문자열을 입력 받아 문자열이 그 언어의 문장이면 “yes”를 , 그렇지 않으면 “no”를 답하는 프로그램이다.

① 결정적 유한 오토마타(DFA)

DFA M = (Q, Σ, δ, q_0 , F)

Q : 상태들의 유한 집합
Σ : 입력 기호의 유한 집합
δ : 상태의 전이 함수로 Q ×Σ → Q 즉 δ(q, a) = p이다.
q0 : 시작 상태 (q0 ∈Q)
F : 종결 상태의 집합 (F ⊆Q)

δ(q, a) = p 는 q 상태에서 입력 기호 a를 본 다음 상태는 p가 된다.

문자열 1001과 0110의 인식 여부

DFA M = ({p, q, r}, {0, 1}, δ, p, {r})
δ : δ(p, 0) =q, δ(p, 1) =p, δ(q, 0) =r, δ(q, 1) =p,
δ(r, 0) =r, δ(r, 1) =r

- 문자열 1001

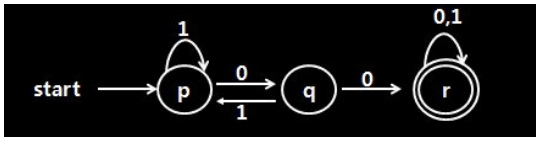
δ(p, 1001) = δ(p, 001) = δ(q, 01) = δ(r, 1) = r ∈ F 인식

- 문자열 0110

δ(p, 0110) = δ(q, 110) = δ(p, 10) = δ(p, 0) = q Not ∈ F 인식 안됨

DFA M = ({p, q, r}, {0, 1}, δ, p, {r})
δ : δ(p, 0) =q, δ(p, 1) =p, δ(q, 0) =r, δ(q, 1) =p,
δ(r, 0) =r, δ(r, 1) =r

δ	0	1
p	q	p
q	r	p
r	r	r

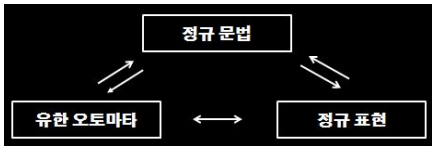


② 비 결정적 유한 오토마타(NFA)

결정적 유한 오토마타(DFA)
δ : 전이 함수로 Q ×Σ → Q 즉 δ(q, a) = p이다.
비 결정적 유한 오토마타(NFA)
δ : 전이 함수로 Q ×Σ → 2Q 즉 δ(q, a) = {p1 , p2 ,…} 이다.

③ 동치 관계

정규 문법, 정규 표현, 유한 오토마타는 서로 동치 관계를 이룬다.
서로 변환이 가능하다.



3. 구문 표현

(1) BNF : 구문에 대한 생성 규칙들의 집합

기호	의미
::=	정의 된다.
|	선택 한다.
각 괄호 <>	Non Terminal 기호
각 괄호로 묶이지 않은 기호	Terminal 기호

첫 문자가 영문으로 시작하고, 두 번째 문자 부터
영문자나 숫자가 오는 식별자

<식별자> ::= <영문자>|<식별자><영문자>|<식별자><숫자>
<영문자> ::= a|b|c|d…|z
<숫자> ::=0|1|2|3|…|9
출제 : 택일을 의미하는 기호는? |, 정의를 의미 하는 것은?

(2) EBNF(Extended BNF)

BNF의 반복 표시 어려움, 메타 기호 사용, 읽기 쉽고 간결

기호	의미	설명
{ }	반복	{a} : a가 0번이상 반복, 정규표현 a^*
[ ]	선택	[x] :x가 없거나 한번, $\{x\}_0^1$

영문자로 시작하고, 최대 8개의 문자를 가질 수 있는 식별자

<식별자> ::= <영문자>{<영숫자> }07
<영숫자> ::= <영문자> |<숫자>
<영문자> ::= a|b|c|… |z
<숫자> ::= 0|1|2|… |9

if 문장에서 else 부분이 선택적

BNF

<if_statement> ::= if <condition> then <statement>
|
if <condition> then
else
<statement>

EBNF

<if_statement> ::= if <condition> then <statement>
[else <statement>]

괄호와 |를 이용하여, 여러 개의 생성 규칙을 묶어서 간단히 표현

BNF

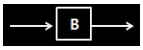
<exp> ::= <exp>+<exp> |
<exp>-<exp> |
<exp>*<exp> |
<exp>/<exp>

EBNF

<exp> ::= <exp>(+|-|* |/)<exp>

(3) 구문 도표

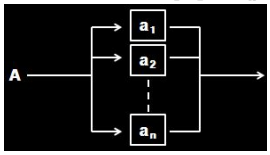
terminnal 

nonterminnal 

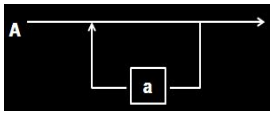
생성규칙 A::=X1+X2+X3+X4+…Xn
X가 terminal인 경우

X가 nonterminal인 경우

생성규칙 A::=$a_1|a_2|\cdots|a_n$

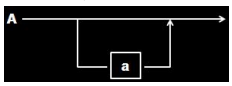


생성규칙 A::={a}

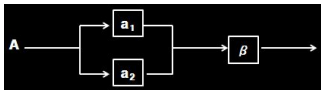


출제 : EBNF에서 반복을 의미하는 것?

생성규칙 A::=[a]



생성규칙 A::=(a1 | a2)β



(4) 유도 트리

① 유도
좌측 유도 : 왼쪽에 있는 nonterminal 기호를 계속 대치
우측 유도 : 오른쪽에 있는 nonterminal 기호를 계속 대치
좌파스(left parse) : 좌측 유도에서 적용된 생성 규칙
우파스 (right parse): 우측 유도에서 적용된 생성 규칙

문장 -(id+id)의 유도 과정

좌측 유도 : 왼쪽에 있는 nonterminal 기호를 계속 대치
좌파스(left parse) : 좌측 유도에서 적용된 생성 규칙

P: 1.E→E+E 2. E→E*E 3. E→(E) 4.E→-E 5. E→id

E⇒ -E 4
⇒ -(E) 43
⇒ -(E+E) 431
⇒ -(id+E) 4315
⇒ -(id+id) 43155 좌패스 = 43155
→

문장 -(id+id)의 유도 과정

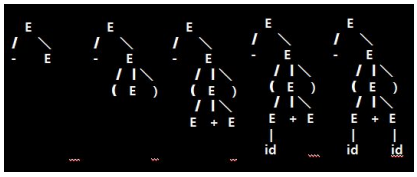
우측 유도 : 오른쪽에 있는 nonterminal 기호를 계속 대치
우파스 (right parse): 우측 유도에서 적용된 생성 규칙

P: 1.E→E+E 2. E→E*E 3. E→(E) 4.E→-E 5. E→id

E⇒ -E 4
⇒ -(E) 43
⇒ -(E+E) 431
⇒ -(E+id) 4315
⇒ -(id+id) 43155 우패스 = 55134
←

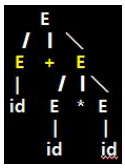
문장이 유도 되는 과정을 트리 형태로 표현

-(id+id)에 대한 좌측 유도 트리



(5) 모호성
id+id*id의 서로 다른 좌측 유도 트리
P : E→E+E | E-E | E*E | E/E | id

E⇒ E+E
⇒ id+E
⇒ id+E*E
⇒ id+id*E
⇒ id+id*id



E⇒ E*E
⇒ E+E*E
⇒ id+E*E
⇒ id+id*E
⇒ id+id*id

모호하지 않은 문법으로 변환, 연산 순위, 결합 법칙 사용 이유
id+id*id
P : E→E+T | E-T | T 같은 연산자 좌 → 우
T→T*F | T/F | F 다른 연산자 위 → 아래
F →id

(6) 구문 분석 방법
① 하향식과 상향식
하향식 : 루트로 시작해서 terminal로 만들어 가는 과정(좌패스)
상향식 : terminal로 시작해서 루트로 만들어 가는 과정(우패스)

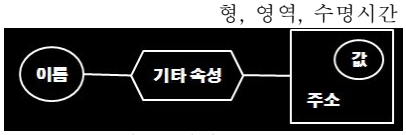
② 하향식의 backtracking
문자열 생성을 실패하면, 다른 생성 규칙으로 재시도를 한다.
시간이 많이 걸리는 단점이 있다.

③ 재시도를 하지 않고 결정하는 구문 방식
LL 구문 방식 : recursive-descent, predictive

④ 하향식 보다 상향식이 효율적이므로 컴파일러 파싱으로 사용
일반적인 상향식 : Shift-reduce
LR 구문 방식 : SLR, LALR, CLR

Section 10 자료 객체

3. 자료 객체
(1) 자료 객체의 구분
프로그래머 정의 자료 : 변수, 상수, 배열, 입출력 파일
시스템 정의 자료 : 스택, 입출력 버퍼, 자료 명세표, 참조 환경
출제 : 스택 (LIFO), 큐(FIFO)의 삽입 삭제 형태 ?
(2) 변수와 상수 출제: 값이 수시로 변하는 것은?
① 변수 : 저장 값을 변경, 이름, 주소나 위치(포인터), 값, 형, 영역, 수명시간



[이름 |기타속성 | 값 | 주소]
[bun | int | 12 | 3400]
② 상수 : 저장 값을 변경하지 못함, 위치 없음
[없음|변경불가|12| 3400]

(3) 바인딩
바인딩 : 대상물과 그것의 실제 속성을 연결되어 결정 되는 것
대상물 : 변수, 상수, 객체, 파일, 예약어 등
① 바인딩 시간
- 언어 정의 시간 : 여러 예약어의 의미 (예약어 인식)
- 언어 구현 시간 : 정수형이 실제 가지는 값의 집합 (값 할당)

- 번역 시간 :
컴파일 시간 - 변수의 형 (변수의 구조)
링크시간 - 외부 함수 (호출 정보인식)
적재시간 - 전역 변수의 위치 (메모리 할당)
- 실행시간 : 변수의 값 (자동 변수, 연산)

② 식별자의 바인딩
- 번역시간
컴파일 시간 : 컴파일러 언어의 변수형, 상수 값의 내부 표현
링크시간 : COMMON문, 부프로그램 이름의 관한 주소
적재시간 : 기억장소 할당 (정적 변수)
- 실행시간
모듈의 시작 시간 : 매개변수 연결, 기억장소 할당 (지역변수)
사용 시점 : 인터프리터 언어의 변수 형 배정, 기억장소 할당
모든 언어 : 배정문 등에서 변수 값을 할당

③ 정적 바인딩과 동적 바인딩
- 정적 바인딩 : 컴파일링시에, 컴파일러 언어, 정적 변수
- 동적 바인딩 : 실행시에, 인터프리터 언어, 지역 변수

(4) 선언과 동적 형 바인딩
① 선언의 정의
실행 시 사용될 자료의 속성을 번역기에 알려주는 문장
② 선언의 목적
- 주기억장치 사용과 접근 방법의 효율성
자료구조의 속성 한정, 기억장소 절감
- 주기억장치의 경영의 효율성
생성과 소멸 시간 인식 static, auto 변수구분
- 정적 형 검사
혼합형 연산 시 고정된 연산 변환, 오류 최소화

③ 변수의 선언
- 명시적 선언 : 대부분 언어에서 사용 (FORTRAN 예)
INTEGER SUM
REAL TOT
- 묵시적 선언 : FORTRAN, Perl, BASIC, APL, SNOBOL
㉠ 영문 규정 (FORTRAN 예, 첫글자 I, J, K, L, M, N 시작)
ICNT - 정수형
TJUMSU - 실수형

㉡ 특수 문자 (Perl 예, 첫글자 $, @, %)
$sum - 스칼라변수 $sum = $sum+1
@sum - 배열 @a = (“Seoul”, “Busan”)
%sum - 연관 변수 %b =(“red” 0xf00, “blue”, 0x00f)

④ 동적 형 바인딩
선언 시에 변수형이 결정되지 않고 배정문에 의해서 결정
즉, 상수 값에 의해 결정

APL
문장 A = 1,0, 2.0, 3.0 1차원 배열 3개 원소
다음 문장 A = 40 이전에 형에 관계없이 정수형

Perl
문장 $A = 10 정수형 변수 $A
다음 문장 $A =“string” 문자열형 변수 $A

장점 : 융통성
단점 : 오류 검사 능력저하, 실행 시 형 검사 실행시간 증가
실행 시 관련 정보를 저장하고 있어야 한다.
변수의 크기는 가변적이어야 한다.

(5) 블록과 영역
① 블록
프로그램 내에서 시작과 끝을 begin-end , {}로 사용
블록 구조 언어 : 블록의 내포를 허용한 언어

② 블록에 선언된 변수의 종류
- 지역 변수 : 자기 블록에서 선언된 변수
- 비 지역 변수 : 상위 블록 선언, 자기 블록에서 사용
- 전역 변수 : 모든 블록에서 사용

```
int x;                          /* 전역적 */
void fun(void) {      /*  fun 블록  */
    double a, b;      /* 지역적, 비 지역적*/
        {                       /* 내포된 블록 */
            int p, q;     /* 내포된 블록에 대하여 지역적 */
        }
    }
```

③ 블록 구조의 장점
- 변수 이름을 자유롭게 사용
- 프로그램을 분할 하여 사용
- 기억장소를 효율적으로 관리

④ Nesting
정적 내포 : 블록들의 내포 관계, 가장 안쪽 블록 선택
블록의 위치에 따라 같은 변수 선언시
가장 안쪽이 우선
동적 내포 : 블록들의 실행 순서, 가장 최근 호출 블록 선택
재귀적 호출

⑤ 영역 : 변수 및 프로시저가 유효하게 사용할 수 있는 범위
- 정적 영역 규칙 : 선언된 위치에서 끝으로 바인딩
- 동적 영역 규칙 : 호출된 역순 추적하여 찾아 바인딩

정적 영역 규칙의 예 C언어

```
int x;
void f(void) {
    char a;
      :
}
void g(void) {
    int b;
      :
}
main() {
    float c;
    :
}
```

정적 영역 규칙의 예

```
int x;
void f(void) {
    char x;
    x = 'A';
      :
}
main() {
    x = 20;
     :
```

```
}
```

정적 영역 규칙의 예

C++의 영역 전환 연산자

```
int  x;
     void  f(void)  {
           char  x;
           x = 'A';          // char  x에 배정
           ::x = 10          // 전역 int  x에 배정
             :
      }

     main() {
        x = 20;        // 전역  int x에 배정
            :
     }
```

(6) 기억장소 바인딩과 수명시간
변수의 수명시간은 바인딩 되어 있는 시간 간격,
생성되었다가 소멸되는 시간

① 정적 변수
실행 전에 바인딩 되어 실행이 종료 될 때까지,
수명시간은 프로그램 실행 전체

장점 : 위치가 변하지 않으므로 바로 접근
할당 회수가 없어 효율적

단점 : 순환 프로시저를 작성할 수 없다.
기억장소를 공유 할 수 없어 효율적 관리 어려움

② 스택-동적 변수
변수가 선언된 곳 또는 블록이 실행될 때 생성되어
그 블록을 벗어날 때 까지를 수명시간으로 갖는다.
장점 : 순환 프로시저 작성 용이, 기억장소 효율적 사용
단점 : 실행 시 기억장소 관리를 위한 시간과 공간 추가

③ 명시적 히프-동적 변수
프로그래머가 필요에 따라 생성시켰다가 소멸 시키는 변수
히프 공간에 할당되며 포인터 사용에 의하여 접근
할당/회수 : C는 malloc / free , C++는 new / delete,
Java는 new / 쓰레기 수집
장점 : 리스트, 트리와 같은 동적 구조에 유리
단점 : 포인터로 접근하기 때문에 복잡하고, 오류 발생

④ 묵시적 히프-동적 변수
변수의 값이 부여될 때 히프의 기억장소에 할당, APL, Perl
장점 : 융통성
단점 : 동적 속성 노력, 오류 처리 능력 낮다.

(7) 변수 초기화
바인딩되는 시간에 값을 바인딩하는 것
정적 변수 : 한 번에 이루어짐
동적 변수 : 할당과 함께 이루어짐

FORTRAN의 DATA문

```
INTEGER     SUM
REAL    PI
DATA   SUM   /0/,     PI /3.14159/
```

C언어

```
int   sum = 0;
```

PASCAL, Modula-2
실행시간에 수행되는 배정문에서만 사용,
선언 시 초기화가 안 된다.

(8) 형 검사
연산 시에 자료 형의 적당한 인수를 받았는지 검사
호환 가능한 형(적법/강제변환)인지 확인

- 정적 형 검사
  정적 바인딩에 검사, 즉 번역(컴파일) 시간에 검사
  자료 형 오류를 일찍 발견, 비용이 적다, 융통성 낮다.
- 동적 형 검사

실행시간 에 검사, 인터프리터에 많이 사용
실행되지 않는 연산 시 검사가 안됨
수행 중에 형 정보 유지 해야 하므로 기억장소 추가되며,
실행 속도가 느려진다.

(9) 강형 언어와 약형 언어
① 강형언어 : 모든 형의 오류를 검사 할 수 있는 언어
② 약형언어 : 정적인 형 체계를 갖추지 않은 언어
LISP, APL, SNOBOL, BLISS

- Java, ML 등은 강형 언어
- Pascal, Ada는 강형에 가까운 언어이다.
- 3C와 C++는 공용체등이 포함, 강형언어가 아니다.

(10) 형 추론
식의 형을 그 부분 식의 형으로 유추하는 것으로
ML, Miranda, Haskell 등에서 사용한다.

fun a(x) = 5 * x
매개 변수와 반환 값에 대한 형이 식에 포함된 상수의 형으로 추론

(11) 형 호환성
① 이름 동등성 : 형 이름이 동일한 객체
struct t1 { int x; int y; };
struct t2 { int x; int y; };
struct t1 a, b; 호환가능 a와 b, c와 d
struct t2 c, d; 호환불가 a와c, b와 d

② 구조적 동등성
형의 이름은 갖더라도 구조가 같다면 호환가능
위의 예에서 a, b, c, d는 모든 구조적 동등성인 자료형이다.

③ 선언적 동등성 : 재 선언
type
type1 = array[1..10] of integer;
type2 = array[1..10] of integer;
type3 = type2; 호환 가능 type2와 type3
호환 불가 type1와 type2

(12) 형 변환
① 확장 변환과 축소 변환
확장 C : int ⇒ float 축소 double ⇒ float

② 묵시적 형 변환과 명시적 형 변환

- 묵시적 형 변환
컴파일러가 자동으로 형을 변환시키는 강제 변환

- 명시적 형 변환
C : x = (int)(3.4 + (double)/(x/2));
C++ : x = int(3.4 + double(x/2));

(13) 기본 자료형과 사용자-정의 자료형
- 기본 자료형
정수, 부동 소수점, 십진수형, 문자형, 불리안형
수치형은 종종 하드웨어에서 직접 지원

- 사용자-정의 순서형
① 열거형
C : enum Color (Red, Green, Blue)
0 1 2
② 부분 범위형
Ada :
type Digit_type is range 0..9;

(14) 문자열 형
- 정적 길이 스트링
선언된 길이가 고정
부족한 문자는 공백으로 채워짐
FORTRAN77, COBOL, Pascal, Ada

- 제한된 동적 길이 스트링
선언된 범위 내에서 가변 길이
C, C++

- 동적 길이 스트링
  무제한 길이
  SNOBOL, Perl

(15) 배열 형
첫원소의 상대적 위치로 각 원소를 식별하는 동질적인 자료의 집합
배열이름, 차원, 원소의 형, 인덱스 집합과 범위 등으로 지정
- 인덱스(첨자)
일반적으로 정수를 사용하지만 Pascal, Modula-2, Ada 등은
불리안형, 문자형, 열거형도 첨자로 사용한다.
- 인덱스(첨자) 의 시작
0부터 시작 : C, C++, Java
1부터 시작 : FORTRAN

동적 배열 : 실행 중에 상한, 하한의 값으로 배열 생성
생성된 배열의 크기가 변하지 않는다.
Algiol68, Ada
유연 배열 : 실행하면서 배열의 크기가 변한다.
APL, SNOBOL4
(16) 레코드 형
각 원소가 이름에 의해서 식별되는 이질적인 자료의 집합

COBOL : 계층 번호를 이용한 레코드 구조

```
01   EMPLOYEE-RECORD.
       03  EMPLOYEE-NAME.
             05  SUNG          PIC     XXXX.
             05  MYOUNG        PIC     X(10).
       03  HOURLY-RATE         PIC     999.
```

(17) 공용체 형
실행 중에 다른 시기에 다른 형의 값을 저장할 수 있는 형

FORTRAN 예

```
EQUIVALENCE  (A, X), (B, Y)

DIMENSION  K(5),  M(5)
EQUIVALENCE  (K(4),  M(2))
```

(18) 포인터 형
① 허상 참조
히프-동적 변수의 주소를 포함하고 있는 포인터

```
int  *x, *y;
x = (int *) malloc(sizeof(int))
   :
*x = 10;
   :
y = x;
free(x);                  → 포인터가 지적한  할당 위치가 사라진다.
   :
/*  *y는 허상 참조  */
printf("%d", *p);
```

② 허상 참조
히프-동적 변수의 주소를 포함하고 있는 포인터

```
{
      int   *x;
       {
            int  y;
            y  =  10;
            x  =  &y;              → 블록이 종료되면서 y가 사라진다.
        }
        :
/* x는 허상 참조 */
}
```

(19) 쓰레기

```
main()  {
     char   *ptr1 = "garbage";
     char   *ptr2 = "multiple  reference";
```

```
        :
    ptr1 = ptr2;
        :   /*  "garbage"는 더이 상 접근 할 수 없다.  */
}
```

(20) 별칭
FORTRAN : EQUIVALENCE 문
Pascal, Ada : 가변 레코드
C, C++ : union

Section 11 순서 제어

4.순서제어
묵시적 순서제어 : 미리 정해진 나열 순서대로(우선순위) 제어
명시적 순서제어 : 괄호를 이용하여 제어
출제 : 묵시적 순서제어란? 우선 순위?

(1) 산술식
단항 : 연산자와 항이 한 개, 예) C언어 : ++, --, - 등
이항 : 연산자와 항이 두 개, 예) C언어 : +, -, *, / 등
삼항 : 연산자와 항이 세 개, 예) C언어 : : ?

```
     +
    / \
   3   *
      / \
     4   5
```

중위(infix) 좌근우 : 3+4*5
전위(perfix) 근좌우 : +3*45
후위(postfix) 좌우근 : 345*+
출제 : A*(B-C) 를 후위로 변환 ,혹은 반대로 A BC-*
A+(B*C) 를 후위로 변환, 혹은 반대로 ABC*+
대부분 언어 중위, Perl 은 전위로 표현되는 연산자 제공
출제 : 단항은 중위에 적합하지 않다. 이항은 중위

① 연산자 평가 순서
우선 순위 : 다른 연산자끼리의 계산 순서
결합 법칙 : 동일한 연산자끼리의 계산 순서
괄호 : 우선 순위와 결합 순서와 상관 없이 우선 순위 결정

② 피연산자 평가 순서 - 부작용 발생(side effect)

```
int  x=5;                  /* x는 전역변수  */
int  f1()  {               /* 함수 f1()은 x값을 17로  */
      x = 17;              /* 변경시키는 부작용을 가지고 있다. */
      return  3;     }
void  f2()  {              /* f2에서 계산된 x의 값은 */
      x = x + f1();  }     /* 피 연산자들의 평가 수준에 따라 다르다. */
void  main()  {            /* x값은 8이거나 20이다. */
      f2();     }
```

중복 연산자 : 수치+수치, 문자열+문자열
관계식
두 개의 피연산자의 값을 비교하는 연산자, 산술보다 우선순위 낮음

(3) 논리식
왼쪽으로 오른쪽으로 식의 값을 계산하는 도중 중단 a|b|c

(4) 배정문
Check point
자료의 l-value(주소), r-value (변수, 상수, 수식)
모든 변수들은 l-value와 r-value를 갖는다.
상수는 r-value만을 갖는다.
배열 A에서 A[i]의 l-value는
배열 A에서 i번째 원소의 주소이고, r-value는 그 주소에 저장되어 있는 값이다.
C의 int *p; l-value는 p 자신의 주소, r-value는 다른 변수의 l-value이다.

(5) 문장에서의 순서 제어
① 무조건 분기문, goto 문
- 장점
  ㉠ 간결한 형태의 goto문은 곧바로 하드웨어로 제공된다.
  ㉡ 범용성(이 구조로 모든 알고리즘 표현 가능)
- 단점
  ㉠ 프로그램이 빈약하게 설계 된다.
  ㉡ 프로그램을 디버그하고, 이해하고, 유지보수하기 어렵다.

Check point
구조적 프로그래밍 기법
GOTO문 사용을 억제
순차, 선택, 반복 제어만 사용 출제 : 기본구조가 아닌것?
계층적(하향식) 설계

기능별로 모듈화
단일 입구, 단일 출구
프로그램의 순서 = 실행 순서

② 선택문
- 양방향 선택문
Dangling else - 어떤 if문와 결합되는지 모호성

```
if (tot == 0)
    if(sum == 0)
        result = 0;
else
    result = 1;
```

Dangling else 문제점 해결

```
if TOT = 0 Then
  if SUM = 0 then
    RESULT := 0;
  end if;
else
  RESULT := 1;
end if;
```

```
if TOT = 0 Then
  if SUM = 0 then
    RESULT := 0;
  else
    RESULT := 1;
  end if;
end if;
```

```
if(tot == 0) {
   if(sum == 0)
      result = 0;
}
else
   result = 1;
```

③ 다중 선택 구조
C언어 : swich ~ case
Pascal, Modula-2, Ada : case 문
VB : Select Case
C언어

```
switch(exp) {
  case  label_1 : 문장1; break;
  case  label_2 : 문장1; break;
     :
  case  label_n : 문장n; break;
  default          : 문장n+1;
}
```

④ 반복문
- 계속 루프

FORTRAN DO 문	C언어 for문

DO 10 I=1, 10, 1
ISUM = ISIM + I
10 CONTINUE
	for(i=1 ; i<=10 ; i++) {
	sum = sum + i;
	}

- 조건 루프
㉠ while - C, C++, Java

사전 검사 루프	사후 검사 루프
while(식) 루프 몸체	Do 루프 몸체 while(식)

㉡ repeat - COBOL

```
PERFORM  SUM-RTN  VARYING I FROM 1 BY 1
                  UNTIL  I > 10.
```

- 조건 루프 break, continue

break 문	continue 문

while(sum<100) {
scanf("%d", &num);
if(num==0) break;
sum += num;
}
	for(i=1; i<=5 ; i++) {
	if(i==3) continue;
	printf("%d",i);
	}

- 자료 구조에 따른 루프 - Perl

```
@names = {"Park",  "Kim", "Lee", "Hong"};
   :
foreach  $name (@names) {
      print  $name;
```

```
}
```

Section 12 부 프로그램

5. 부프로그램

(1) 일반적인 부 프로그램의 특성
코루틴 제외 (종속이 아닌 대등한 함수는 제외)
① 각 부프로그램은 단일 진입점을 갖는다.
② 호출 프로그램이 중단 되면 피호출 프로그램도 중단 된다.
③ 임의의 주어진 시간에 수행중인 부프로그램은 꼭 하나이다.
④ 부프로그램의 실행이 끝났을 때, 제어는 항상 호출자에게 돌아간다.
⑤ 부 프로그램에 적합한 자료 구조는 스택이다.
출제 : 서브루틴 호출처리 시 복귀주소, 조회하는데 적합한 자료구조는? 스택

(2) 기본 정의
① 호출(Call) - 수행 명령, 활성(Active) - 실행중
② 정의 - 표제부와 본체
표제부 - 키워드, 부프로그램 이름, 매개변수
③ 매개변수 프로파일 - 매개변수 개수, 순서, 형
부프로그램의 프로토콜 - 프로파일+반환치의 형+부프로그램형

(3) 매개변수
실 매개변수(실인수) : 호출 시 공급되는 매개변수
가 매개변수(가인수) : 호출 시 공급받는 매개변수

(4) 매개변수 전달 방식
① 값 전달(call by value) - 전달 값만 존재
장점 : 매개변수 값의 접근이 빠르다.
단점 : 매개변수가 긴 배열처럼 크다면 비용이 많이 든다.
② 결과 전달(call by result) - 결과값이 존재
③ 값-결과 전달(call by value-result) - 전달 값, 결과 값
④ 참조 전달(call by reference , address) - 주소 전달
⑤ 이름 전달(call by name) - 주소는 같고, 변수 명은 다르다.
장점 : 매개변수 값을 복사할 필요 없다.
공간과 시간 면에서 효율적이다.
단점 : 별칭이 발생할 수 있다.
부작용, 주소를 사용하므로 접근 시간이 길다.

(5) 주요 언어의 매개변수 전달 방식

	value	result	value-result	reference	name
FORTRAN				○	
FORTRAN77			○	○	○
ALGOL 60	○				
Pascal	○			○	
C	○			○	
Ada	○	○	○		
C++	○			○	

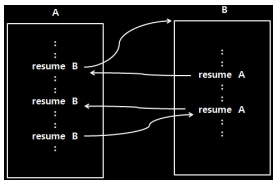
Java 는 매개변수는 값 전달, 객체 참조 변수만 접근되므로
객체 매개 변수는 본질적으로 참조 전달이다.

C : 참조 전달의 효과
```
/* swap(&c, &d) */
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```
C++ : p1은 참조 전달이지만 fun에서 변경될 수 없고,
p2는 값 전달, p3는 참조 전달
```
void fun(const int &p1, int p2, int &p3) { … }
```

(6) 중복 부 프로그램
```
Ada
Declare
function SQUARE(C: integer) return is
begin
    return C * C;
End SQURE;
function SQUARE(X: float) return is
begin
    return X * X;
end SQURE;
begin
  put(SQUARE(3));
```

```
    put(SQUARE(3.4));
end;
```

(7) 코루틴



- 부 프로그램의 수행이 완전히 종료되기 전에 호출할 수 있는 프로그램
- 호출하는 모듈과 호출된 모듈이 서로 실행 제어권을 주고 받으면서 대등한 관계를 가지며 작업을 수행
- 호출되면 일부분만 수행하고 제어를 반환할 수 있고 다시 호출될 때 제어가 반환된 곳에서 다시 시작한다.

출제 : 위의 설명은 ? 코루틴

(8) 순환 호출이 가능한 구조

```
    int   factorial(int   n)  {
                                        /* 이 지점에서 스택 내용 */
        if   (n <= 1)   return 1;
        else   return (n * factorial(n-1));
  }

void   main()  {
      int   value;
      value   =   factorial(3);
   }
```

Section 13 객체 프로그래밍

6. 객체지향(Object oriented) 프로그래밍

(1) 객체지향 프로그래밍의 개요
1960대 Simula 67 → 1980년의 Smalltalk 80에서 완전한 모습
객체 지향 언어의 분류
- 객체 기반 : 객체를 지원, Ada
- 클래스 기반 : 객체와 클래스를 지원. CLU
- 객체 지향 : Smalltalk, Ada 95. C++, Java 등

(2) 기본 구성 요소의 개념
객체지향 언어에서의 프로그램은 객체들의 집합,
객체들을 생성하고 객체들 간에 메시지를 통해 정보 교환
① 추상 자료형은 정의 클래스, 클래스의 인스턴스가 객체
② 클래스의 정의에 따라 객체가 소유하는 변수를 인스턴스
③ 클래스에 정의된 부프로그램을 메소드
④ 메소드 호출을 메시지, 메시지의 집합을 메시지 프로토콜 혹은 메시지 인터페이스

(3) 클래스
클래스는 하나 이상의 유사한 객체들을 묶어서 공통된 특성으로 표현한 것으로, 하나의 클래스는 여러 개의 객체를 생성하기 위한 형판(template)이라 할 수 있다. 공통된 속성과 행위의 집합
출제 : 하나 이상의 유사 객체 집합은?
공통된 속성과 행위를 갖는 객체들의 집합은?

(4) 객체
객체는 실세계에 존재하는 하나의 단위에 대한 소프트웨어적 표현으로, 자신의 속성을 다타내는 자료와, 그 자료를 조작 처리하는 절차 또는 행위가 결합된 것이다.
객체 = 변수 + 메소드

(5) 메소드 출제 : 메시지를 받아 실행하는 구체적인 연산을 정의?
클래스의 객체 연산을 정의하는 함수 또는 프로시저를 의미

(6) 메시지
객체에게 일을 시키는 행위
구성 요소: 객체의 이름, 메소드이름, 인수

(7) 캡슐화
여러 부프로그램과 자료를 하나로 묶는것, 캡슐화된 모듈의

구체적인 표현 및 구현 내용을 사용자가 집적 접근할 수 없도록
감춰 놓은 것을 정보 은폐(information hiding),
출제 : 캡슐화된 객체는 언제든 재사용이 가능하다

(8) 추상화
세부적인 사항이 아닌 다른 것과 구별 되는 중요한 특징만을
나타내는 것. 실제 대상물의 복잡한 속성 중에서 응용 분야에
필요한 요소들로만 구성된 모델로 표현, 추상화 결과는
하나의 모듈로 나타낸다.

(9) 상속
① 기존 클래스로 부터 모든 속성과 메소드를 상속받아
재 사용하는 기술

② 상속받는 클래스 : 하위 또는 유도 클래스
상속하는 클래스 : 상위, 부모, 기저 클래스
상위 계층으로 갈수록 일반화 되고 간단해지며,
하위 계층으로 갈수록 특수화 되고 개별화된다.

③ 단일 상속 : Smalltalk, Ada95, C++, Java
다중 상속 : C++

④ 오버라이드(override) : 클래스간 메소드명 동일
오버로딩(overroad) : 클래스 내부 안에 메소드,명 동일