시험에 나오는 것만 공부한다!

2024 시나공

기출문제집

정보처리기사

필기

30쪽 OSI 모델 꼭 외우기

길벗알앤디 지음
(강윤석, 김용갑, 김우경, 김종일)

길벗

2412401

핵심 230 데이터 검증

데이터 검증이란 원천 시스템의 데이터를 목적 시스템의 데이터로 전환하는 과정이 정상적으로 수행되었는지 여부를 확인하는 과정을 말한다.

- 데이터 전환 검증은 검증 방법과 검증 단계에 따라 분류할 수 있다.

2412504

핵심 231 오류 데이터 정제

오류 데이터 정제는 오류 관리 목록의 각 항목을 분석하여 원천 데이터를 정제하거나 전환 프로그램을 수정하는 것이다.

오류 데이터 분석

- 오류 관리 목록의 오류 데이터를 분석하여 오류 상태, 심각도, 해결 방안을 확인 및 기재한다.
- 상태

Open	오류가 보고만 되고 분석되지 않은 상태
Assigned	오류의 영향 분석 및 수정을 위해 개발자에게 오류를 전달한 상태
Fixed	개발자가 오류를 수정한 상태
Closed	수정된 오류에 대해 테스트를 다시 했을 때 오류가 발견되지 않은 상태
Deferred	오류 수정을 연기한 상태
Classified	보고된 오류를 관련자들이 확인했을 때 오류가 아니라고 확인된 상태

4과목 프로그래밍 언어 활용

20.8

2413001

핵심 232 배치 프로그램

배치 프로그램은 사용자와의 상호 작용 없이 여러 작업들을 미리 정해진 일련의 순서에 따라 일괄적으로 처리하는 것을 의미한다.

- 배치 프로그램이 갖추어야 하는 필수 요소는 다음과 같다.

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

23.5, 23.2, 20.8

400102

핵심 233 C/C++의 데이터 타입 크기 및 기억 범위

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

20.9

핵심 234 C언어의 구조체

400131

배열이 자료의 형과 크기가 동일한 변수의 모임이라면 구조체는 자료의 종류가 다른 변수의 모임이라고 할 수 있다. 예를 들어 이름, 직위, 급여 등의 필드가 필요한 사원 자료를 하나의 단위로 관리하려면 이름과 직위는 문자, 급여는 숫자와 같이 문자와 숫자가 혼용되므로 배열로는 처리할 수 없습니다. 이런 경우 구조체를 사용하면 간단하게 처리할 수 있다.

- 구조체를 정의한다는 것은 int나 char 같은 자료형을 하나 만드는 것을 의미한다.
- 구조체는 'structure(구조)'의 약어인 'struct'를 사용하여 정의한다.
- 구조체 정의 예

```
struct sawon {
   char name[10];
   char position[10];
   int pay;
}
```

23.2, 21.3, 20.9

핵심 235 JAVA의 데이터 타입 크기 및 기억 범위

400103

종류	데이터 타입	크기
문자	char	2Byte
정수	byte	1Byte
	short	2Byte
	int	4Byte
	long	8Byte
실수	float	4Byte
	double	8Byte
논리	boolean	1Byte

23.7, 22.4

핵심 236 Python의 시퀀스 자료형

2459955

시퀀스 자료형(Sequence Type)이란 리스트(List), 튜플(Tuple), range, 문자열처럼 값이 연속적으로 이어진 자료형을 말한다.

- 리스트(List) : 다양한 자료형의 값을 연속적으로 저장하며, 필요에 따라 개수를 늘리거나 줄일 수 있음
- 튜플(Tuple) : 리스트처럼 요소를 연속적으로 저장하지만, 요소의 추가, 삭제, 변경은 불가능함
- range : 연속된 숫자를 생성하는 것으로, 리스트, 반복문 등에서 많이 사용됨

23.7, 23.5, 21.8, 21.3, 20.8, 20.6

핵심 237 변수의 개요 / 변수명 작성 규칙

400202

변수의 개요

변수(Variable)는 컴퓨터가 명령을 처리하는 도중 발생하는 값을 저장하기 위한 공간으로, 변할 수 있는 값을 의미한다.

- 변수는 저장하는 값에 따라 정수형, 실수형, 문자형, 포인터형 등으로 구분한다.

변수명 작성 규칙

- 영문자, 숫자, _(under bar)를 사용할 수 있다.
- 첫 글자는 영문자나 _(under bar)로 시작해야 하며, 숫자는 올 수 없다.
- 글자 수에 제한이 없다.
- 공백이나 *, +, −, / 등의 특수문자를 사용할 수 없다.
- 대 · 소문자를 구분한다.
- 예약어를 변수명으로 사용할 수 없다.
- 변수 선언 시 문장 끝에 반드시 세미콜론(;)을 붙여야 한다.
- 변수 선언 시 변수명에 데이터 타입을 명시하는 것을 헝가리안 표기법(Hungarian Notation)이라고 한다.

22.7, 21.8

핵심 238 가비지 콜렉터 (Garbage Collector)

2413333

- 변수를 선언만 하고 사용하지 않으면 이 변수들이 점유한 메모리 공간은 다른 프로그램들이 사용할 수 없게 된다.
- 이렇게 선언만 하고 사용하지 않는 변수들이 점유한 메모리 공간을 강제로 해제하여 다른 프로그램들이 사용할 수 있도록 하는 것을 가비지 콜렉션(Garbage Collection)이라고 하며, 이 기능을 수행하는 모듈을 가비지 콜렉터(Garbage Collector)라고 한다.

23.2, 21.5, 21.3, 20.9

핵심 239 산술 연산자

400401

산술 연산자는 가, 감, 승, 제 등의 산술 계산에 사용되는 연산자를 말한다.

- 산술 연산자에는 일반 산술식과 달리 한 변수의 값을 증가하거나 감소시키는 증감 연산자가 있다.

연산자	의미	비고
+	덧셈	
−	뺄셈	
*	곱셈	
/	나눗셈	
%	나머지	
++	증가 연산자	**전치** : 변수 앞에 증감 연산자가 오는 형태로 먼저 변수의 값을 증감시킨 후 변수를 연산에 사용함(++a, −−a). **후치** : 변수 뒤에 증감 연산자가 오는 형태로 먼저 변수를 연산에 사용한 후 변수의 값을 증감시킴(a++, a−−).
−−	감소 연산자	

핵심 240 관계 연산자

400402

관계 연산자는 두 수의 관계를 비교하여 참(true) 또는 거짓(false)을 결과로 얻는 연산자이다.

- 거짓은 0, 참은 1로 사용되지만 0외의 모든 숫자도 참으로 간주된다.

연산자	의미
==	같다
!=	같지 않다
〉	크다
〉=	크거나 같다
%	
〈	작다
〈=	작거나 같다

23.7, 23.5, 21.5, 20.6

핵심 241 비트 연산자

400403

비트 연산자는 비트별(0, 1)로 연산하여 결과를 얻는 연산자이다.

연산자	의미	비고
&	and	모든 비트가 1일 때만 1
^	xor	모든 비트가 같으면 0, 하나라도 다르면 1
|	or	모든 비트 중 한 비트라도 1이면 1
~	not	각 비트의 부정, 0이면 1, 1이면 0
〈〈	왼쪽 시프트	비트를 왼쪽으로 이동
〉〉	오른쪽 시프트	비트를 오른쪽으로 이동

23.7, 23.5, 22.7, 22.4, 22.3

핵심 242 논리 연산자

400404

논리 연산자는 두 개의 논리 값을 연산하여 참(true) 또는 거짓(false)을 결과로 얻는 연산자이다. 관계 연산자와 마찬가지로 거짓은 0, 참은 1이다.

연산자	의미	비고
!	not	부정
&&	and	모두 참이면 참
||	or	하나라도 참이면 참

단 산 시 관 비 논 대
항 술 포 계 트 리 건 입

400405

핵심 243 대입 연산자

연산 후 결과를 대입하는 연산식을 간략하게 입력할 수 있도록 대입 연산자를 제공한다. 대입 연산자는 산술, 관계, 비트, 논리 연산자에 모두 적용할 수 있다.

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

대분류	중분류	연산자	결합규칙	우선 순위
단항 연산자	단항 연산자	!(논리 not) ~(비트 not) ++(증가) --(감소) sizeof(기타)	←	높음
이항 연산자	산술 연산자	* / %(나머지) + -	→	
	시프트 연산자	⟨⟨ ⟩⟩		
	관계 연산자	⟨ ⟨= ⟩= ⟩		
		==(같다) !=(같지 않다)		
	비트 연산자	&(비트 and) ^(비트 xor) |(비트 or)		
	논리 연산자			
		&&(논리 and) ||(논리 or)		
삼항 연산자	조건 연산자	? :	→	
대입 연산자	대입 연산자	= += -= *= /= %= ⟨⟨= ⟩⟩= 등	←	
순서 연산자	순서 연산자	,	→	낮음

22.4, 20.8

400406

핵심 244 조건 연산자

조건 연산자는 조건에 따라 서로 다른 수식을 수행한다.

- 형식

```
조건 ? 수식1 : 수식2;
```

- '조건'의 수식이 참이면 '수식1'을, 거짓이면 '수식2'를 실행한다.

22.3, 21.8, 21.5

400408

핵심 245 연산자 우선순위

- 한 개의 수식에 여러 개의 연산자가 사용되면 기본적으로 아래 표의 순서대로 처리된다.
- 아래 표의 한 줄에 가로로 나열된 연산자는 우선순위가 같기 때문에 결합규칙에 따라 ←는 오른쪽에 있는 연산자부터, →는 왼쪽에 있는 연산자부터 차례로 계산된다.

23.2

2459932

핵심 246 scanf( ) 함수

scanf( ) 함수는 C언어의 표준 입력 함수로, 키보드로 입력받아 변수에 저장하는 함수이다.

형식

scanf(서식 문자열, 변수의 주소)	• 서식 문자열 : 입력받을 데이터의 자료형을 지정함 • 변수의 주소 : 데이터를 입력받을 변수를 적는다. 변수의 주소로 입력받아야 하기 때문에 변수에 주소연산자 &를 붙임

예 scanf("%3d", &a);

▶ % : 서식 문자임을 지정

▶ 3 : 입력 자릿수를 3자리로 지정

▶ d : 10진수로 입력

▶ &a : 입력받은 데이터를 변수 a의 주소에 저장

특징

- 입력받을 데이터의 자료형, 자릿수 등을 지정할 수 있다.
- 한 번에 여러 개의 데이터를 입력 받을 수 있다.
- 서식 문자열과 변수의 자료형은 일치해야 한다.

예 scanf("%d %f", &i, &j); → '%d'와 i, "%f"와 j는 자료형이 일치해야 한다.

예 printf("%-8.2f", 200.2);
(∨는 빈 칸을 의미함)

200.20∨∨

▶ % : 서식 문자임을 지정
▶ - : 왼쪽부터 출력
▶ 8 : 출력 자릿수를 8자리로 지정
▶ 2 : 소수점 이하를 2자리로 지정
▶ f : 실수로 출력

2459933

핵심 247 서식 문자열

서식 문자열	의미
%d	정수형 10진수를 입 · 출력하기 위해 지정함
%u	부호없는 정수형 10진수를 입 · 출력하기 위해 지정함
%o	정수형 8진수를 입 · 출력하기 위해 지정함
%x	정수형 16진수를 입 · 출력하기 위해 지정함
%c	문자를 입 · 출력하기 위해 지정함
%s	문자열을 입 · 출력하기 위해 지정함
%f	소수점을 포함하는 실수를 입 · 출력하기 위해 지정함
%e	지수형 실수를 입 · 출력하기 위해 지정함
%ld	long형 10진수를 입 · 출력하기 위해 지정함
%lo	long형 8진수를 입 · 출력하기 위해 지정함
%lx	long형 16진수를 입 · 출력하기 위해 지정함
%p	주소를 16진수로 입 · 출력하기 위해 지정함

22.7, 22.4, 22.3, 21.8, 21.5, 20.8

2459935

핵심 249 주요 제어문자

문자	의미	기능
\n	new line	커서를 다음 줄 앞으로 이동함
\b	backspace	커서를 왼쪽으로 한 칸 이동함
\t	tab	커서를 일정 간격 띄움
\r	carriage return	커서를 현재 줄의 처음으로 이동함
\0	null	널 문자를 출력함
\'	single quote	작은따옴표를 출력함
\"	double quote	큰따옴표를 출력함
\a	alert	스피커로 벨 소리를 출력함
\\	backslash	역 슬래시를 출력함
\f	form feed	한 페이지를 넘김

22.7, 22.4, 22.3, 21.8, 21.5, 20.8

2459934

핵심 248 printf( ) 함수

printf( ) 함수는 C언어의 표준 출력 함수로, 인수로 주어진 값을 화면에 출력하는 함수이다.

형식

printf(서식 문자열, 변수)	• 서식 문자열 : 변수의 자료형에 맞는 서식 문자열을 입력함 • 변수 : 서식 문자열의 순서에 맞게 출력할 변수를 적음. scanf( )와 달리 주소 연산자 &를 붙이지 않음

21.3, 20.9

400333

핵심 250 JAVA에서의 표준 출력

- JAVA에서 값을 화면에 출력할 때는 System 클래스의 서브 클래스인 out 클래스의 메소드 print( ), println( ), printf( ) 등을 사용하여 출력한다.
- 형식 1 : 서식 문자열에 맞게 변수의 내용을 출력함

```
System.out.printf(서식 문자열, 변수)
```

– printf( ) 메소드는 C언어의 printf( ) 함수와 사용법이 동일하다.

예 System.out.printf("%-8.2f", 200.2);
(V는 빈 칸을 의미함)

```
200.20VV
```

▶ % : 서식 문자임을 지정
▶ - : 왼쪽부터 출력
▶ 8 : 출력 자릿수를 8자리로 지정
▶ 2 : 소수점 이하를 2자리로 지정
▶ f : 실수로 출력

• 형식 2 : 값이나 변수의 내용을 형식없이 출력함

```
System.out.print( )
```

– 문자열을 출력할 때는 큰따옴표로 묶어줘야 한다.
– 문자열 또는 문자열 변수를 연속으로 출력할 때는 + 를 이용한다.

예 System.out.print("abc123" + "def");

```
abc123def
```

• 형식 3 : 값이나 변수의 내용을 형식없이 출력한 후 커서를 다음 줄의 처음으로 이동함

```
System.out.println( )
```

– println( ) 메소드는 출력 후 다음 줄로 이동한다는 것을 제외하면 print( ) 메소드와 사용법이 동일하다.

예 System.out.print("abc123" + "def");

```
abc123def
|
```

커서의 위치

22.3, 21.5

400502

핵심 251 단순 if문

if문은 조건에 따라서 실행할 문장을 달리하는 제어문이며, 단순 if문은 조건이 한 개 일 때 사용하는 제어문이다.

• 조건이 참일 때만 실행할 문장을 지정할 수도 있고, 참과 거짓에 대해 각각 다른 실행문을 지정할 수도 있다.

• 형식1 : 조건이 참일 때만 실행함

– 조건이 참일 때 실행할 문장이 하나인 경우

```
if(조건)
   실행할 문장;
```

if는 조건 판단문에 사용되는 예약어이므로 그대로 적는다.
조건은 참(1) 또는 거짓(0)이 결과로 나올 수 있는 수식을 ( ) 안에 입력한다.

실행할 문장; 조건이 참일 경우 실행할 문장을 적는다.

– 조건이 참일 때 실행할 문장이 두 문장 이상인 경우

```
if(조건)
{
   실행할 문장1;
   실행할 문장2;
        ⋮
}
```

{ } 사이에 조건이 참일 경우 실행할 문장을 적는다.

예제 1 a가 10보다 크면 a에서 10을 빼기

```
#include <stdio.h>
main( )
{
   int a = 15, b;
   if (a > 10) ❶
      b = a - 10; ❷
   printf("%d\n", b); ❸
}
```

a가 10보다 크면 ❷번 문장을 실행하고, 아니면 ❸번 문장으로 이동해서 실행을 계속한다.

❶번의 조건식이 참일 경우 실행할 문장이다. b는 5가 된다.

여기서는 ❶번의 조건식이 거짓일 경우 실행할 문장이 없다. 조건 판단문을 벗어나면 무조건 ❸번으로 온다.

결과

```
5
```

• 형식2 : 조건이 참일 때와 거짓 때 실행할 문장이 다름

```
if(조건)
   실행할 문장1;
else
   실행할 문장2;
```

조건이 참일 경우 실행할 문장을 적는다. 참일 경우 실행할 문장이 두 문장 이상이면 { }를 입력하고 그 사이에 문장을 적는다.

조건이 거짓일 경우 실행할 문장을 적는다. 두 문장 이상인 경우 {}를 입력하고 그 사이에 문장을 적는다.

예제 2 a가 b보다 크면 'a-b', 아니면 'b-a'를 수행하기

```
#include <stdio.h>
main( )
{
   int a = 10, b = 20, cha;
   if (a > b) ❶
      cha = a - b; ❷
   else ❸
      cha = b - a; ❹
   printf("%d\n", cha);
}
```

a가 b보다 크면 ❷번 문장을 실행하고, 아니면 ❸번의 다음 문장인 ❹번 문장을 실행한다.

❶번의 조건식이 참일 경우 실행할 문장이다. 참이 아니기 때문에 초기화 시키지 않은 cha에는 알 수 없는 값이 그대로 있게 된다.

❶번의 조건식이 거짓일 경우 실행할 문장의 시작점이다.

❶번의 조건식이 거짓일 경우 실행할 실제 처리문이다. cha는 10이 된다.

결과

```
10
```

22.4, 22.3, 21.5, 20.8

400503

핵심 252 다중 if문

다중 if문은 조건이 여러 개 일 때 사용하는 제어문이다.

• 형식1

```
if(조건1)
   실행할 문장1;          조건1이 참일 경우 실행할 문장을 적는다.
else if(조건2)
   실행할 문장2;          조건2가 참일 경우 실행할 문장을 적는다.
else if(조건3)
   실행할 문장3;          조건3이 참일 경우 실행할 문장을 적는다.
        ⋮
else
   실행할 문장4;          앞의 조건이 모두 거짓일 경우 실행할 문장을 적는다.
```

예제 1 점수에 따라 등급 표시하기

```
#include <stdio.h>
main( )
{
  int jum = 85;
  if (jum >= 90) ❶                  jum이 90 이상이면 ❷번을 실행하고, 아니면 ❸번으로 이동한다.
    printf("학점은 A입니다.\n"); ❷   "학점은 A입니다."를 출력하고, ❾번으로 이동하여 프로그램을 종료한다.
  else if (jum >= 80) ❸             jum이 80 이상이면 ❹번을 실행하고, 아니면 ❺번으로 이동한다.
    printf("학점은 B입니다.\n"); ❹   "학점은 B입니다."를 출력하고, ❾번으로 이동하여 프로그램을 종료한다.
  else if (jum >= 70) ❺             jum이 70 이상이면 ❻번을 실행하고, 아니면 ❼번으로 이동한다.
    printf("학점은 C입니다.\n"); ❻   "학점은 C입니다."를 출력하고, ❾번으로 이동하여 프로그램을 종료한다.
  else ❼                           ❺번의 조건식이 거짓일 경우 ❽번을 실행한다.
    printf("학점은 F입니다.\n"); ❽   "학점은 F입니다."를 출력하고, ❾번으로 이동하여 프로그램을 종료한다.
} ❾
```

결과 학점은 B입니다.

• 형식2 : if문 안에 if문이 포함된다.

```
if(조건1)
  {                                 조건1이 참일 경우 실행할 문장의 시작점이다.
    if(조건2)
      실행할 문장1;                 조건2가 참일 경우 실행할 문장을 적는다.
    else
      실행할 문장2;                 조건2가 거짓일 경우 실행할 문장을 적는다.
  }
else
  실행할 문장3;                     조건1이 거짓일 경우 실행할 문장을 적는다.
```

예제 2 홀수, 짝수 판별하기

```
#include <stdio.h>
main( )
{
  int a = 21, b = 10;
  if (a % 2 == 0) ❶                 a를 2로 나눈 나머지가 0이면 ❷번을 실행하고, 아니면 ❻번으로 이동한다.
    if (b % 2 == 0) ❷               b를 2로 나눈 나머지가 0이면 ❸번을 실행하고, 아니면 ❹번으로 이동한다.
      printf("모두 짝수\n"); ❸      "모두 짝수"를 출력하고, ⓫번으로 이동하여 프로그램을 종료한다.
    else ❹                          ❷번의 조건식이 거짓일 경우 ❺번을 실행한다.
      printf("a : 짝수, b : 홀수\n"); ❺
                                    "a : 짝수, b : 홀수"를 출력하고, ⓫번으로 이동하여 프로그램을 종료한다.
  else ❻                           ❶번의 조건식이 거짓일 경우 실행할 문장의 시작점이다.
    if (b % 2 == 0) ❼               b를 2로 나눈 나머지가 0이면 ❽번을 실행하고, 아니면 ❾번으로 이동한다.
      printf("a : 홀수, b : 짝수\n"); ❽
                                    "a : 홀수, b : 짝수"를 출력하고, ⓫번으로 이동하여 프로그램을 종료한다.
    else ❾                          ❼번의 조건식이 거짓일 경우 실행할 문장의 시작점이다.
      printf("모두 홀수\n"); ❿      "모두 홀수"를 출력하고, ⓫번으로 이동하여 프로그램을 종료한다.
} ⓫
```

결과 a : 홀수, b : 짝수

22.7

핵심 253 switch문

400504

switch문은 조건에 따라 분기할 곳이 여러 곳인 경우 간단하게 처리할 수 있는 제어문이다.

• 형식

```
switch(수식) ❶
{ ❷
  case 레이블1: ❸
    실행할 문장1;
    break;
  case 레이블2: ❹
    실행할 문장2;
    break;
    :
  default
    실행할 문장3;
} ❺
```

- switch는 switch문에 사용되는 예약어로 그대로 입력한다.
- 수식 : '레이블1' ~ '레이블n'의 값 중 하나를 도출하는 변수나 수식을 입력한다.

❷~❺번이 switch문의 범위이다. '{'로 시작해서 '}'로 끝난다. 반드시 입력해야 한다.

- case는 switch문에서 레이블을 지정하기 위한 예약어로 그대로 입력해야 한다.
- 레이블1 : ❶번 식의 결과가 될 만한 값 중 하나를 입력한다. 결과가 '레이블1'과 일치하면 이곳으로 찾아온다. 식의 결과가 5종류로 나타나면 case문이 5번 나와야한다.

❶번 식의 결과가 ❸번의 '레이블1'과 일치할 때 실행할 문장이다.

switch문을 탈출하여 ❺번으로 간다.

❶번의 식의 결과가 ❹번의 '레이블2'와 일치하면 찾아오는 곳이다.

❶번의 식의 결과가 ❹번의 '레이블2'와 일치할 때 실행할 문장이다.

switch문을 탈출하여 ❺번으로 간다.

❶번의 식의 결과가 '레이블1' ~ '레이블n'에 해당하지 않는 경우 찾아오는 곳이다.

- case문의 레이블에는 한 개의 상수만 지정할 수 있으며, int, char, enum형의 상수만 가능하다.
- case문의 레이블에는 변수를 지정할 수 없다.
- break문은 생략이 가능하지만 break문이 생략되면 수식과 레이블이 일치할 때 실행할 문장부터 break문 또는 switch문이 종료될 때까지 모든 문장이 실행된다.

예제 점수(jum)에 따라 등급 표시하기

```
#include <stdio.h>
main( )
{
  int jum = 85;
  switch (jum / 10)
  switch(수식)
```

jum을 10으로 나눠 결과에 해당하는 숫자를 찾아간다. 85/10은 8.5지만 C 언어에서 정수 나눗셈은 결과도 정수이므로 결과는 8이다. 8에 해당하는 ❺번으로 이동하여 ❻, ❼번을 실행한다.

```
{ ❶
  case 10:
  case 9: ❷
    printf("학점은 A입니다.\n"); ❸
  break; ❹
  case 8: ❺
    printf("학점은 B입니다.\n"); ❻
  break; ❼
  case 7:
    printf("학점은 C입니다.\n");
    break;
  case 6:
    printf("학점은 D입니다.\n");
    break;
  default:
    printf("학점은 F입니다.\n");
  } ❽
} ❾
```

❶~❽번까지가 switch 조건문의 범위이다.

100점일 경우 'jum/10'의 결과인 10이 찾아오는 곳이지만 할 일은 'case 9:'와 같으므로 아무것도 적지 않는다. 아무것도 적지 않으면 다음 문장인 ❷번으로 이동한다.

'jum/10'이 9일 경우 찾아오는 곳이다. ❸, ❹번을 실행한다.

"학점은 A입니다."를 출력한다.

break를 만나면 switch문을 탈출하여 ❾번으로 이동한다.

'jum/10'이 8일 경우 찾아오는 곳이다. ❻, ❼번을 실행한다.

"학점은 B입니다."를 출력한다.

switch문을 탈출하여 ❾번으로 이동한다.

case 10~6에 해당되지 않는 경우, 즉 jum이 59 이하인 경우 찾아오는 곳이다.

"학점은 F입니다."를 출력한다.

결과 **학점은 B입니다.**

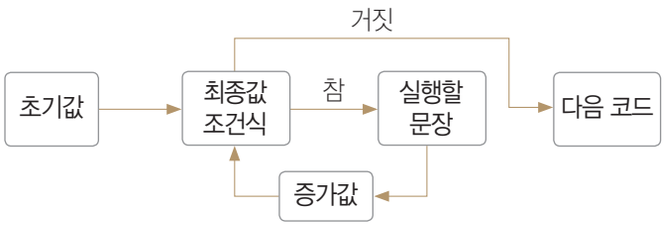
23.2, 22.7, 22.4, 22.3, 21.5, 20.8

핵심 254 for문

400602

for문은 초기값, 최종값, 증가값을 지정하는 수식을 이용해 정해진 횟수를 반복하는 제어문이다.

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



for문의 동작 과정

• 형식

for(식1; 식2; 식3)	• for는 반복문을 의미하는 예약어로 그대로 입력한다. • 식1 : 초기값을 지정할 수식을 입력한다. • 식2 : 최종값을 지정할 수식을 입력한다. • 식3 : 증가값으로 사용할 수식을 입력한다.
실행할 문장;	식2가 참일 동안 실행할 문장을 입력한다. 실행할 문장이 두 문장 이상일 경우 { }를 입력하고 그 사이에 처리할 문장들을 입력한다.

• for문은 처음부터 최종값에 대한 조건식을 만족하지 못하면 한 번도 수행하지 않는다.

23.7, 23.2, 22.3, 21.8, 21.5, 21.3, 20.9

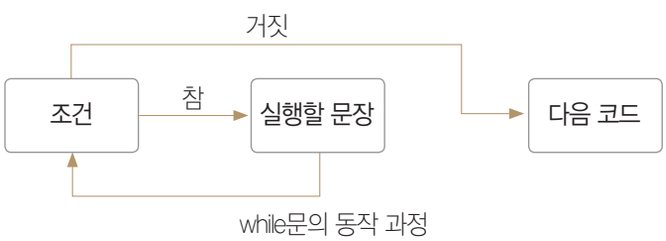
400603

핵심 255 while문

while문은 조건이 참인 동안 실행할 문장을 반복 수행하는 제어문이다.

• while문은 조건이 참인동안 실행할 문장을 반복 수행하다가 조건이 거짓이면 while문을 끝낸 후 다음 코드를 실행한다.

• while문은 조건이 처음부터 거짓이면 한 번도 수행하지 않는다.



while문의 동작 과정

• 형식

while(조건)	• while은 반복문에 사용되는 예약어로 그대로 입력한다. • (조건) : 참이나 거짓을 결과로 갖는 수식을 '조건'에 입력한다. 참(1)을 직접 입력할 수도 있다.
실행할 문장;	조건이 참인 동안 실행할 문장을 입력한다. 문장이 두 문장 이상인 경우 { }를 입력하고 그 사이에 처리할 문장들을 입력한다.

예제 다음은 1~5까지의 합을 더하는 프로그램이다. 결과를 확인하시오.

```
#include <stdio.h>
main( )
{
  int a = 0, hap = 0;
  while (a < 5) ❶          a가 5보다 작은 동안 ❷~❻번 문장을 반복하여 수행한다.
  { ❷                      ❷~❻번까지가 반복문의 범위이다. 반복문에서 실행할 문장이 하나인 경우는 { }를 생략해도 된다.
    a++; ❸                 'a = a + 1;'과 동일하다. a의 값을 1씩 증가시킨다.
    hap += a; ❹            'hap = hap + a'와 동일하다. a의 값을 hap에 누적시킨다.
  } ❺                      반복문의 끝이다.
  printf("%d, %d\n", a, hap); ❻
                           결과 5, 15
}                          a가 5가 되었을 때 5를 hap에 누적한 다음 while 문을 벗어나기 때문에 a는 5로 끝난다.
```

23.7, 23.2, 21.5

400604

핵심 256 do~while문

do~while문은 조건이 참인 동안 정해진 문장을 반복 수행하다가 조건이 거짓이면 반복문을 벗어나는 while문과 같은 동작을 하는데, 다른 점은 do~while문은 실행할 문장을 무조건 한 번 실행한 다음 조건을 판단하여 탈출 여부를 결정한다는 것이다.

• do~while문은 실행할 문장을 우선 실행한 후 조건을 판별하여 조건이 참이면 실행할 문장을 계속 반복 수행하고, 거짓이면 do~while문을 끝낸 후 다음 코드를 실행한다.

do~while문의 동작 과정

• 형식

do	do는 do~while문에 사용되는 예약어로, do~while의 시작 부분에 그대로 입력한다.
실행할 문장;	조건이 참인 동안 실행할 문장을 입력한다. 문장이 두 문장 이상인 경우 { }를 입력하고 그 사이에 실행할 문장들을 입력한다.
while(조건);	• while은 do~while문에 사용되는 예약어로, do~while의 끝 부분에 그대로 입력한다. • (조건) : 참이나 거짓을 결과로 갖는 수식을 '조건'에 입력한다. 참(1)을 직접 입력할 수도 있다.

예제 다음은 1부터 10까지 홀수의 합을 더하는 프로그램이다. 결과를 확인하시오.

```
#include <stdio.h>
main( )
{
  int a = 0, hap = 0;
  do ❶                          do~while 반복문의 시작점이다. ❷~❺번 사이의 문장을 반복하여 수행한다.
  { ❷                           ❷~❺번까지가 반복문의 범위이다.
    hap += a; ❸                 'hap = hap + a'와 동일하다. a의 값을 hap에 누적시킨다.
    a += 2; ❹                   'a = a + 2;'와 동일하다. a의 값을 2씩 증가시킨다.
  } while(a < 10); ❺            a가 10보다 작은 동안 ❷~❺번 사이의 문장을 반복하여 수행한다.
  printf("%d, %d\n", a, hap); ❻
                                결과 11, 25
                                a가 9가 되었을 때 9를 hap에 누적한 다음 a에 2를 더해 a가 11이 되었을 때 do~while문을 벗어나기 때문에 a는 11로 끝난다.
}
```

23.5, 22.7, 22.3

핵심 257 break, continue

400605

switch문이나 반복문의 실행을 제어하기 위해 사용되는 예약어이다.

- break : switch문이나 반복문 안에서 break가 나오면 블록을 벗어난다.
- continue
  - continue 이후의 문장을 실행하지 않고 제어를 반복문의 처음으로 옮긴다.
  - 반복문에서만 사용된다.

예제 다음은 1~5까지의 합을 더하되 2의 배수는 배제하는 프로그램이다. 결과를 확인하시오.

```
#include <stdio.h>
 main( )
 {
    int a = 0, hap = 0;
    while(1) ❶                  조건이 참(1)이므로 무한 반복한다. 중간에 반복을 끝내는 문장이 반드시 있어야 한다.
    { ❷                         ❷~❾번까지가 반복문의 범위이다.
       a++; ❸                   'a = a + 1;'과 동일하다. a의 값을 1씩 증가시킨다.
       if(a > 5) ❹              a가 5보다 크면 ❺번 문장을 수행하고, 아니면 ❻번 문장을 수행한다.
          break; ❺              반복문을 탈출하여 ❿번으로 이동한다.
       if (a % 2 == 0) ❻        a를 2로 나눈 나머지가 0이면, 즉 a가 2의 배수이면 ❼번 문장을 수행하고, 아니면 ❽번 문장으로 이동한다.
          continue; ❼           이후의 문장, 즉 ❽번을 생략하고 반복문의 처음인 ❶번으로 이동한다. 2의 배수는 hap에 누적되지 않는다.
       hap += a; ❽              'hap = hap + a'와 동일하다. a의 값을 hap에 누적시킨다.
    } ❾                         반복문의 끝이다.
    printf("%d, %d\n", a, hap); ❿
  결과 6, 9
 }
```

23.7, 22.4, 22.3, 21.8

핵심 258 배열

400702

배열의 개념

배열은 동일한 데이터 유형을 여러 개 사용해야 할 경우 이를 손쉽게 처리하기 위해 여러 개의 변수들을 조합해서 하나의 이름으로 정의해 사용하는 것을 말한다.

- 배열은 하나의 이름으로 여러 기억장소를 가리키기 때문에 배열에서 개별적인 요소들의 위치는 첨자를 이용하여 지정한다.
- 배열은 변수명 뒤에 대괄호 [ ]를 붙이고 그 안에 사용할 개수를 지정한다.
- C언어에서 배열의 위치는 0부터 시작된다.
- 배열은 행 우선으로 데이터가 기억장소에 할당된다.

- C 언어에서 배열 위치를 나타내는 첨자 없이 배열 이름을 사용하면 배열의 첫 번째 요소의 주소를 지정하는 것과 같다.

1차원 배열

- 1차원 배열은 변수들을 일직선상의 개념으로 조합한 배열이다.
- 형식

자료형 변수명 [개수];	• 자료형 : 배열에 저장할 자료의 형을 지정함 • 변수명 : 사용할 배열의 이름으로 사용자가 임의로 지정함 • 개수 : 배열의 크기를 지정하는 것으로 생략할 수 있음

예 int a[5] : 5개의 요소를 갖는 정수형 배열 a

	첫 번째	두 번째	세 번째	네 번째	다섯 번째
배열 a	a[0]	a[1]	a[2]	a[3]	a[4]

※ a[3] : a는 배열의 이름이고, 3은 첨자로서 배열 a에서의 위치를 나타냄. a[3]에 4를 저장시키려면 'a[3] = 4'와 같이 작성함

예제 1 1차원 배열 a의 각 요소에 10, 11, 12, 13, 14를 저장한 후 출력하기

```
#include <stdio.h>
main( )
{
  int a[5];
```
5개의 요소를 갖는 정수형 배열 a를 선언한다. 선언할 때는 사용할 개수를 선언하고, 사용할 때는 첨자를 0부터 사용하므로 주의해야 한다.

	첫 번째	두 번째	세 번째	네 번째	다섯 번째
배열 a	a[0]	a[1]	a[2]	a[3]	a[4]

```
  int i;
```
정수형 변수 i를 선언한다

```
  for (i = 0; i < 5; i++)
```
반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❶번 문장을 반복하여 수행한다. 그러니까 ❶번 문장을 5회 반복하는 것이다.

```
    a[i] = i + 10; ❶
```
배열 a의 i번째에 i+10을 저장시킨다. i는 0~4까지 변하므로 배열 a에 저장된 값은 다음과 같다.

배열 a	10	11	12	13	14
	a[0]	a[1]	a[2]	a[3]	a[4]

```
  for (i = 0; i < 5; i++)
```
반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❷번 문장을 반복하여 수행한다.

```
    printf("%d ", a[i]); ❷
```
배열 a의 i번째를 출력한다. i는 0~4까지 변하므로 출력 결과는 다음과 같다. 서식 문자열에 '\n'이 없기 때문에 한 줄에 붙여서 출력한다.

```
}
```
결과 **10 11 12 13 14**

22.3, 21.5

핵심 259 2차원 배열

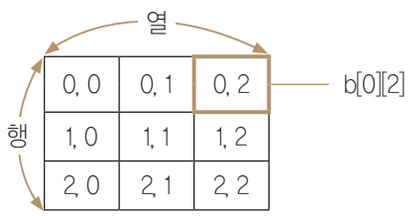
400703

- 2차원 배열은 변수들을 평면, 즉 행과 열로 조합한 배열이다.
- 형식

자료형 변수명 [행개수][열개수]	• 자료형 : 배열에 저장할 자료의 형을 지정함 • 변수명 : 사용할 배열의 이름으로 사용자가 임의로 지정함 • 행개수 : 배열의 행 크기를 지정함 • 열개수 : 배열의 열 크기를 지정함

예 int b[3][3] : 3개의 행과 3개의 열을 갖는 정수형 배열 b

배열 b



b[0][2] : b는 배열의 이름이고, 0은 행 첨자, 2는 열 첨자로서 배열 b에서의 위치를 나타낸다.

예제 3행 4열의 배열에 다음과 같이 숫자 저장하기

배열 a

1	2	3	4
5	6	7	8
9	10	11	12

```
#include <stdio.h>
main( )
{
  int a[3][4];
```
3행 4열의 크기를 갖는 정수형 배열 a를 선언한다.

```
  int i, j, k = 0;
```
정수형 변수 i를 선언한다

```
  for (i = 0; i < 3; i++) ❶
```
반복 변수 i가 0에서 시작하여 1씩 증가하면서 3보다 작은 동안 ❷~❽번을 반복하여 수행한다. 결국 ❸번 문장을 3회 반복한다.

```
  { ❷
```
❷~❽이 ❶번 반복문의 반복 범위이지만 실제 실행할 문장은 ❸번 하나이다.

```
    for (j = 0; j < 4; j++) ❸
```
반복 변수 j가 0에서 시작하여 1씩 증가하면서 4보다 작은 동안 ❹~❼번을 반복하여 수행한다.
- i가 0일 때 j는 0에서 3까지 4회 반복
- i가 1일 때 j는 0에서 3까지 4회 반복
- i가 2일 때 j는 0에서 3까지 4회 반복 수행하므로 ❺~❻번을 총 12회 수행한다.

```
    { ❹                  ❹~❼이 ❸번 반복문의 반복 범위이다.
      k++; ❺             k를 1씩 증가시킨다. k는 총 12회 증가하므로 1~12까지 변한다.
      a[i][j] = k; ❻     배열 a의 i행 j열에 k를 기억시킨다. a[0][0]~a[2][3]까지 1~12가 저장된다.
    } ❼                  ❹번의 짝이다.
  } ❽                    ❹번의 ❷번의 짝이다..
}                        첫 번째 { 의 짝이자 프로그램의 끝이다.
```

20.6

핵심 260 배열의 초기화

400704

- 배열 선언 시 초기값을 지정할 수 있다.
- 배열을 선언할 때 배열의 크기를 생략하는 경우에는 반드시 초기값을 지정해야 초기값을 지정한 개수 만큼의 배열이 선언된다.

예 1차원 배열 초기화

방법1 char a[3] = {'A', 'B', 'C'}

방법2 char a[ ] = {'A', 'B', 'C'}

배열 a

A	B	C
a[0]	a[1]	a[2]

예 2차원 배열 초기화

방법1 int a[2][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} };

방법2 int a[2][4] = {10, 20, 30, 40, 50, 60, 70, 80}

배열 a

	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]

- 배열의 개수보다 적은 수로 배열을 초기화하면 입력된 값만큼 지정한 숫자가 입력되고, 나머지 요소에는 0이 입력된다.

예 int a[5] = { 3, }; 또는 int a[5] = { 3 };

배열 a

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

23.7, 21.8

핵심 261 배열 형태의 문자열 변수

400705

C언어에서는 큰따옴표(“ ”)로 묶인 글자는 글자 수에 관계없이 문자열로 처리된다.

- C언어에는 문자열을 저장하는 자료형이 없기 때문에 배열, 또는 포인터를 이용하여 처리한다.
- 형식

char 배열이름[크기] = “문자열”

- 배열에 문자열을 저장하면 문자열의 끝을 알리기 위한 널 문자(‘₩0’)가 문자열 끝에 자동으로 삽입된다.
- 배열에 문자열을 저장할 때는 배열 선언 시 초기값으로 지정해야 하며, 이미 선언된 배열에는 문자열을 저장할 수 없다.
- 문자열 끝에 자동으로 널 문자(‘\0’)가 삽입되므로, 널 문자까지 고려하여 배열 크기를 지정해야 한다.

예 char a[5] = “love” →

l	o	v	e	\0

예제 다음의 출력 결과를 확인하시오.

```
#include <stdio.h>
main( )
{
  char a = 'A';
```

문자형 변수 a에 문자 ‘A’를 저장한다. 문자형 변수에는 한 글자만 저장되며, 저장될 때는 아스키 코드값으로 변경되어 정수로 저장된다. a가 저장하고 있는 값은 문자로 출력하면 ‘A’가 출력되지만 숫자로 출력하면 ‘A’에 대한 아스키 코드 65가 출력된다.

```
  char b[9] = "SINAGONG";
```

9개의 요소를 갖는 배열 b를 선언하고 다음과 같이 초기화한다. 저장되는 글자는 8자이지만 문자열의 끝에 자동으로 저장되는 널 문자(‘\0’)를 고려하여 크기를 9로 지정한 것이다.

배열 b

S	I	N	A	G	O	N	G	\0
b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]

```
  char *c = "SINAGONG"; ❶
```

포인터 변수 c에 “SINAGONG”이라는 문자열이 저장된 곳의 주소를 저장한다.

```
  printf("%c\n", a);
```

변수 a의 값을 문자로 출력한다.

```
  printf("%s\n", b);
```

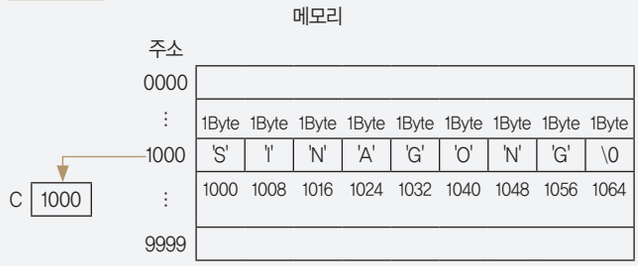
배열 위치를 나타내는 첨자 없이 배열 이름을 사용하면 배열의 첫 번째 요소의 주소를 지정하는 것과 같으므로 배열 b의 첫 번째 요소가 가리키는 곳의 값을 문자열로 출력한다.

```
    printf("%s\n", c);          포인터 변수 c가 가리키는 곳의 값을 문자열로 출력한다.
}
```

결과

```
A
SINAGONG
SINAGONG
```

코드 해설



위 코드 중 ❶번을 실행할 경우 메모리를 그려보면 다음과 같다.

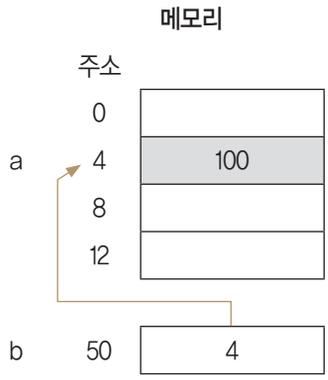
22.2, 22.4, 21.8

400801

핵심 262 포인터와 포인터 변수

포인터는 변수의 주소를 말하며, C언어에서는 주소를 제어할 수 있는 기능을 제공한다.

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



예를 들어, a 변수에 100을 저장시키고, a 변수의 주소를 포인터 변수 b에 기억시켰다면 다음 그림과 같이 표현하고 말할 수 있다.

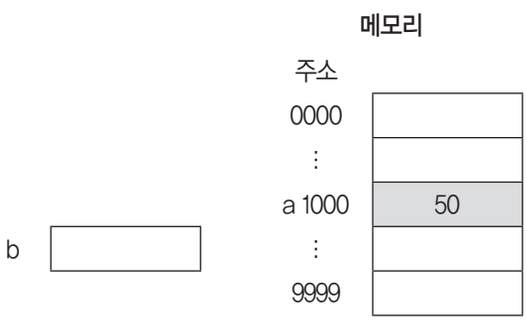
- a는 메모리의 4번지에 대한 이름이다.
- a 변수의 주소는 4다.
- a 변수에는 100이 기억되어 있다.
- 4번지에는 100이 기억되어 있다.
- &a는 a 변수의 주소를 말한다. 즉 &a는 4다.
- 포인터 변수 b는 a 변수의 주소를 기억하고 있다.
- 포인터 변수가 가리키는 곳의 값을 말할 때는 *을 붙인다.
- *b는 b에 저장된 주소가 가리키는 곳에 저장된 값을 말하므로 100이다.

예제 1 다음 C언어로 구현된 프로그램의 출력 결과를 확인하시오.

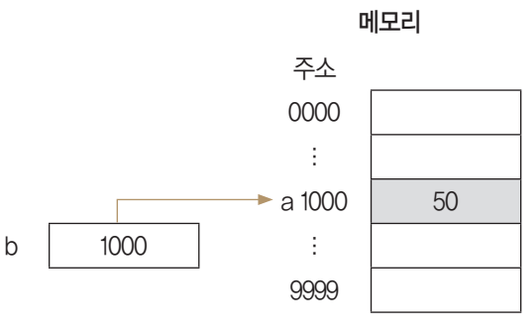
```
main( )
{
   int a = 50; ❶          정수형 변수 a를 선언하고 50으로 초기화한다.
   int *b; ❷              정수형 변수가 저장된 곳의 주소를 기억할 포인터 변수 b를 선언한다.
   b = &a; ❸              정수형 변수 a의 주소를 포인터 변수 b에 기억시킨다. b에는 a의 주소가 저장된다.
   *b = *b+20; ❹          b가 가리키는 곳의 값에 20을 더한다. b가 가리키는 곳이 a이므로 결국 a의 값도 바뀌는 것이다.
   printf("%d, %d", a, *b); ❺
                          결과 70, 70
                          • ❷와 같이 선언할 때 *는 해당 변수가 포인터 변수라는 것을 의미한다.
                          • ❹, ❺와 같이 사용할 때 *를 붙이면 그 포인터 변수가 가리키는 곳의 값을 의미한다.
}
```

위 코드의 실행 과정에 따라 메모리의 변화를 그려보면 다음과 같다.

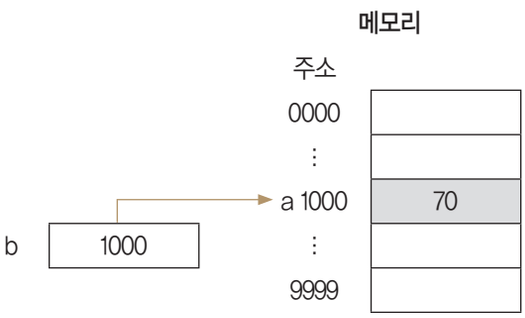
❶, ❷번 수행 : 주기억장치의 빈 공간 어딘가에 a라는 이름을 붙이고 그 곳에 50을 저장함



❸번 수행 : 변수 a의 주소가 b에 기억된다는 것은 b가 변수 a의 주소를 가리키고 있다는 의미



❹번 수행 : b가 가리키는 곳의 값에 20을 더해 다시 b가 가리키는 곳에 저장함. 그곳은 변수 a의 주소이므로 변수 a의 값도 저절로 변경되는 것



2.2.3, 21.5

핵심 263 포인터와 배열

400802

배열을 포인터 변수에 저장한 후 포인터를 이용해 배열의 요소에 접근할 수 있다.

- 배열 위치를 나타내는 첨자를 생략하고 배열의 대표명만 지정하면 배열의 첫 번째 요소의 주소를 지정하는 것과 같다.
- 배열 요소에 대한 주소를 지정할 때는 일반 변수와 동일하게 & 연산자를 사용한다.

예 int a[5], *b;

b = a; → 배열의 대표명을 적었으므로 a 배열의 시작 주소인 a[0]의 주소를 b에 저장한다.

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

	a[0]	a[1]	a[2]	a[3]	a[4]	← 배열 표기 방법
배열 a	첫 번째	두 번째	세 번째	네 번째	다섯 번째	
	*(a+0)	*(a+1)	*(a+2)	*(a+3)	*(a+4)	← 포인터 표기 방법

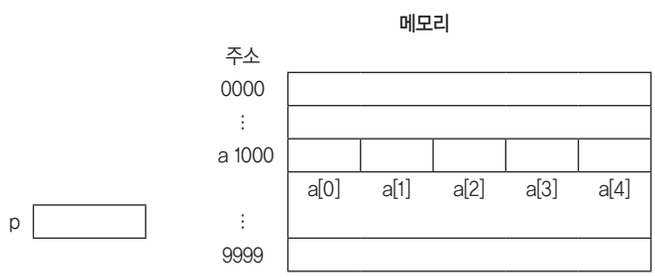
- 배열의 요소가 포인터인 포인터형 배열을 선언할 수 있다.

예제 다음의 출력 결과를 확인하시오.

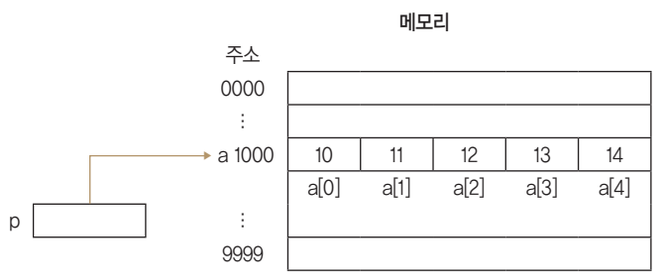
```
main( )
{
  int a[5];                    5개의 요소를 갖는 정수형 배열 a를 선언한다. 선언할 때 사용할 개수를 선언하고, 사용할 때는 첨자를 0부터 사용한다.
  int i;                       정수형 변수 i를 선언한다.
  int *p; ❶
  for (i = 0; i < 5; i++)      반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❷번을 반복 수행한다.
     a[i] = i + 10; ❷
  p = a; ❸
  for (i = 0; i < 5; i++)      반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❹번을 반복하여 수행한다.
     printf("%d ", *(p+i)); ❹   결과 10 11 12 13 14
}
```

코드의 실행 과정에 따라 메모리의 변화를 그려보면 다음과 같다.

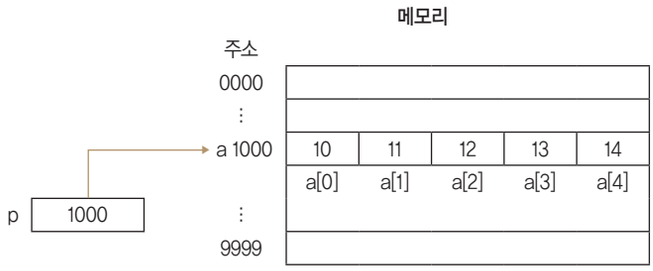
❶ 정수형 변수가 저장된 곳의 주소를 기억할 정수형 포인터 변수 p를 선언한다.



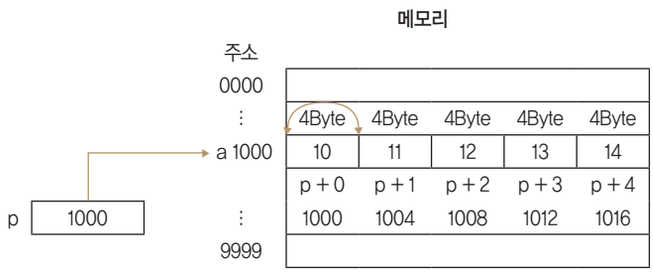
❷ 배열 a의 i번째에 i+10을 저장한다. i는 0~4까지 변하므로 배열 a에 저장된 값은 다음과 같다.



❸ 배열명 a는 배열의 주소이므로 포인터 변수 p에는 배열 a의 시작 위치가 기억된다. 배열의 이름은 주소이므로 'p = &a'처럼 입력하지 않도록 주의해야 한다.



❹ p에 저장된 값은 정수형 배열의 시작 주소이다. p의 값을 1 증가 시킨다는 것은 현재 p가 가리키고 있는 정수형 자료의 주소에서 다음 정수형 자료의 주소로 가리키는 주소를 증가시킨다는 것이다. 정수형 자료의 크기는 4바이트이므로 다음 물리적 메모리의 주소는 4Byte 증가한 곳을 가리키는 것이다. p에 저장된 배열의 시작 주소에서 1번지씩, 즉 4Byte씩 증가시키는 것을 그림으로 표현하면 다음과 같다.



- p+0 : 배열의 시작 주소에 0을 더했으므로, 배열의 시작 주소인 '1000' 번지 그대로이다.
- *(p+0) : '1000' 번지의 값은 10이다. 10을 출력한다.
- p+1 : '1000'에서 한 번지 증가한 주소는 '1004' 번지이다.
- *(p+1) : '1004' 번지의 값은 11이다. 11을 출력한다.
- p+2 : '1000'에서 두 번지 증가한 주소는 '1008' 번지이다.
- *(p+2) : '1008' 번지의 값은 12이다. 12를 출력한다.

⋮

401101

핵심 264 Python의 기본 문법

- 변수의 자료형에 대한 선언이 없다.
- 문장의 끝을 의미하는 세미콜론(;)을 사용할 필요가 없다.
- 변수에 연속하여 값을 저장하는 것이 가능하다.

  예 x, y, z = 10, 20, 30
- if나 for와 같이 코드 블록을 포함하는 명령문을 작성할 때 코드 블록은 콜론(:)과 여백으로 구분한다.
- 여백은 일반적으로 4칸 또는 한 개의 탭만큼 띄워야하며, 같은 수준의 코드들은 반드시 동일한 여백을 가져야 한다.

22.7, 22.4, 22.3, 21.8, 21.5, 21.3, 20.8

401102

핵심 265 Python의 데이터 입 · 출력 함수

input( ) 함수

- input( ) 함수는 Python의 표준 입력 함수로, 키보드로 입력받아 변수에 저장하는 함수이다.
- 형식

변수 = input(출력문자)
- '출력문자'는 생략이 가능하며, '변수'는 사용자가 임의로 지정할 수 있다.
- 값을 입력하고 Enter를 누르면, 입력한 값이 '변수'에 저장된다.

예 a = input('입력하세요.') → **화면에 입력하세요.**가 출력되고 그 뒤에서 커서가 깜빡거리며 입력을 기다린다. 키보드로 값을 입력하면 변수 a에 저장된다.

print( ) 함수

- 형식1

print(출력값1, 출력값2, …, sep = 분리문자, end = 종료문자)
- '출력값'에는 숫자, 문자, 문자열, 변수 등 다양한 값이나 식이 올 수 있다.
- 'sep'는 여러 값을 출력할 때 값과 값 사이를 구분하기 위해 출력하는 문자로, 생략할 경우 기본값은 공백 한 칸(' ')이다.
- 'end'는 맨 마지막에 표시할 문자로, 생략할 경우 기본값은 줄 나눔이다.

예 print(82, 24, sep = '–', end = ',') → 82와 24 사이에 분리문자 '–'가 출력되고, 마지막에 종료문자 ','가 출력된다.

결과 82–24,

22.7

핵심 266 입력 값의 형변환(Casting)

2414031

input( ) 함수는 입력되는 값을 무조건 문자열로 저장하므로, 숫자로 사용하기 위해서는 형을 변환해야 한다.

- 변환할 데이터가 1개일 때

```
변수 = int(input( ))      정수로 변환 시
변수 = float(input( ))    실수로 변환 시
```

예 a = int(input( )) → input( )으로 입력받은 값을 정수로 변환하여 변수 a에 저장한다.

- 변환할 데이터가 2개 이상일 때

```
변수1, 변수2, … = map*(int, input( ).split( ))
                                  정수로 변환 시
변수1, 변수2, … = map(float, input( ).split( ))
                                  실수로 변환 시
```

예 a, b = map(int, input( ).split( ))
→ input( ).split( )으로 입력받은 2개의 값을 정수로 변환하여 변수 a, b에 저장한다.

23.5, 22.3

핵심 267 리스트(List)

2414003

- C와 Java에서는 여러 요소들을 하나의 이름으로 처리할 때 배열을 사용했는데 Python에서는 리스트를 사용한다.
- 리스트는 필요에 따라 개수를 늘이거나 줄일 수 있기 때문에 리스트를 선언할 때 크기를 적지 않는다.
- 배열과 달리 하나의 리스트에 정수, 실수, 문자열 등 다양한 자료형을 섞어서 저장할 수 있다.
- Python에서 리스트의 위치는 0부터 시작한다.
- 형식

```
리스트명 = [ 값1, 값2, … ]
```

리스트명은 사용자가 임의로 지정하며, 리스트를 의미하는 대괄호 사이에 저장할 값들을 쉼표(,)로 구분하여 입력한다.

```
리스트명 = list([ 값1, 값2, … ])
```

예 1 방법1 : a = [10, 'mike', 23.45]
방법2 : a = list([10, 'mike', 23.45])

		a[0]	a[1]	a[2]
결과	리스트 a	10	mike	23.45

※ 두 방법에 대한 결과는 같습니다.

예 2 a[0] = 1 → a[0]에 1을 저장한다.

		a[0]	a[1]	a[2]
결과	리스트 a	1	mike	23.45

23.5, 22.3

핵심 268 딕셔너리(Dictionary)

2414004

- 딕셔너리는 연관된 값을 묶어서 저장하는 용도로 사용한다.
- 리스트는 저장된 요소에 접근하기 위한 키로 위치에 해당하는 0, 1, 2 등의 숫자를 사용하지만 딕셔너리는 사용자가 원하는 값을 키로 지정해 사용한다.
- 딕셔너리에 접근할 때는 딕셔너리 뒤에 대괄호([ ])를 사용하며, 대괄호([ ]) 안에 키를 지정한다.
- 형식

```
딕셔너리명 = { 키1:값1, 키2:값2, … }
```

딕셔너리명은 사용자가 임의로 지정하며, 딕셔너리를 의미하는 중괄호 사이에 저장할 값들을 쉼표로 구분하여 입력한다.

```
딕셔너리명 = dict({ 키1:값1, 키2:값2, … })
```

예 1 방법1 : a = {'이름':'홍길동', '나이':25, '주소':'서울'}
방법2 : a = dict({'이름':'홍길동', '나이':25, '주소':'서울'})

		a['이름']	a['나이']	a['주소']
결과	리스트 a	'홍길동'	25	'서울'

예 2 a['이름'] = '이순신' → 딕셔너리 a의 '이름' 위치에 '이순신'을 저장한다.

		a['이름']	a['나이']	a['주소']
결과	리스트 a	'이순신'	25	'서울'

20.9, 20.8

401106

핵심 269 슬라이스(Slice)

슬라이스는 문자열이나 리스트와 같은 순차형 객체에서 일부를 잘라(slicing) 반환하는 기능이다.

• 형식1

객체명[초기위치:최종위치]	'초기위치'에서 '최종위치'-1까지의 요소들을 가져온다.
객체명[초기위치:최종위치:증가값]	• '초기위치'에서 '최종위치'-1까지 '증가값'만큼 증가하면서 해당 위치의 요소들을 가져온다. • '증가값'이 음수인 경우 '초기위치'에서 '최종위치'+1까지 '증가값' 만큼 감소하면서 해당 위치의 요소들을 가져온다.

• 슬라이스는 일부 인수를 생략하여 사용할 수 있다.

객체명[:] 또는 객체명[::]	객체의 모든 요소를 반환한다.
객체명[초기위치:]	객체의 '초기위치'에서 마지막 위치까지의 요소들을 반환한다.
객체명[:최종위치]	객체의 0번째 위치에서 '최종위치'-1까지의 요소들을 반환한다.
객체명[::증가값]	객체의 0번째 위치에서 마지막 위치까지 '증가값'만큼 증가하면서 해당 위치의 요소들을 반환한다.

예 a = ['a', 'b', 'c', 'd', 'e']일 때

a[1:3] → ['b', 'c']

a[0:5:2] → ['a', 'c', 'e']

a[3:] → ['d', 'e']

a[:3] → ['a', 'b', 'c']

a[::3] → ['a', 'd']

23.7, 22.4

401201

핵심 270 Python – if문

• 형식

if 조건:	예약어 if와 참 또는 거짓이 결과로 나올 수 있는 조건을 입력한 후 끝에 콜론(:)을 붙여준다.
실행할 문장	조건이 참일 경우 실행할 문장을 적는다.

예제 a가 10보다 크면 a에서 10을 빼기

```
a = 15
if a > 10:  ❶
    a = a - 10  ❷
print(a)  ❸
```

a가 10보다 크면 ❷번 문장을 실행하고, 아니면 ❸번 문장으로 이동해서 실행을 계속한다.

❶번의 조건식이 참일 경우 실행할 문장이다. a는 5가 된다.

여기서는 ❶번의 조건식이 거짓일 경우 실행할 문장이 없다. if문을 벗어나면 무조건 ❸번으로 온다.

결과 5

22.6, 21.8

401202

핵심 271 Python – for문

• 형식1 : range를 이용하는 방식이다.

for 변수 in range(최종값):	0에서 '최종값'-1까지 연속된 숫자를 순서대로 변수에 저장하며 '실행할 문장'을 반복 수행한다.
실행할 문장	반복 수행할 문장을 적는다.

예 1 for i in range(10):
sum += i → • i에 0에서 9까지 순서대로 저장하며 실행할 문장을 반복 수행한다.
• i의 값을 sum에 누적한다. sum에는 0부터 9까지의 합 45가 저장된다.

예 2 for i in range(11, 20):
sum += i → • i에 11에서 19까지 순서대로 저장하며 실행할 문장을 반복 수행한다.
• i의 값을 sum에 누적한다. sum에는 11부터 19까지의 합 135가 저장된다.

예 3 for i in range(-10, 20, 2):
sum += i → • i에 -10에서 19까지 2씩 증가하는 숫자를 순서대로 저장하며 실행할 문장을 반복 수행한다.
• i의 값을 sum에 누적한다. sum에는 -10, -8, -6, ···, 16, 18의 합 60이 저장된다.

• 형식2 : 리스트(List)를 이용하는 방식이다.

for 변수 in 리스트	리스트의 0번째 요소에서 마지막 요소까지 순서대로 변수에 저장하며 실행할 문장을 반복 수행한다.
실행할 문장	반복 수행할 문장을 적는다.

예제 다음은 리스트 a에 저장된 요소들의 합과 평균을 구하는 프로그램을 Python으로 구현한 것이다.

```
❶ a = [ 35, 55, 65, 84, 45 ]
❷ hap = 0
❸ for i in a:
❹     hap += i
❺ avg = hap / len(a)
❻ print(hap, avg)
```

코드 해설

❶ 리스트 a를 선언하면서 초기값을 지정한다.

	a[0]	a[1]	a[2]	a[3]	a[4]
리스트 a	35	55	65	84	45

❷ 총점을 저장할 변수 hap을 0으로 초기화한다.

❸ for문의 시작이다. 리스트 a의 요소 수만큼 ❹번을 반복 수행한다.

❹ i의 값을 hap에 누적한다. i는 리스트 a의 각 요소의 값을 차례대로 받는다. 변수의 변화는 다음과 같다.

첫 번째 수행 : 리스트 a의 첫 번째 값이 i를 거쳐 hap에 누적된다.

hap	i	리스트 a				
35	35	35	55	65	84	45

두 번째 수행 : 리스트 a의 두 번째 값이 i를 거쳐 hap에 누적된다.

hap	i	리스트 a				
90	55	35	55	65	84	45

⋮

이런 방식으로 리스트 a의 요소 수만큼 반복한다.

❺ hap을 리스트 a의 요소 수로 나눈 후 결과를 avg에 저장한다.
- len(리스트) : 리스트의 요소 수를 구한다. len(a)는 5다.

❻ 결과 `284 56.8`

21.3

핵심 272 Python – While문

401203

• 형식

```
while 조건:
    실행할 문장
```

- while은 예약어로, 그대로 입력한다.
- 참이나 거짓을 결과로 갖는 수식을 조건에 입력한다. 참(1 또는 True)을 직접 입력할 수도 있다.

실행할 문장 : 조건이 참인 동안 반복 수행할 문장을 적는다.

예제 다음은 1~5까지의 합을 구하는 프로그램을 Python으로 구현한 것이다.

```
i, hap = 0, 0 ❶      i와 hap을 0으로 초기화한다.
while i < 5: ❷       i가 5보다 작은 동안 ❸, ❹번 문장을 반복하여 수행한다.
    i += 1 ❸         i의 값을 1씩 증가시킨다.
    hap += i ❹       i의 값을 hap에 누적시킨다.
print(hap) ❺         결과 15
```

21.5

핵심 273 Python – 클래스

401204

• 정의 형식

```
class 클래스명:
    실행할 문장
    def 메소드명(self, 인수):
        실행할 문장
        return 값
```

- def는 메소드를 정의하는 예약어로, 그대로 입력하고, 메소드명은 사용자가 임의로 지정한다.
- self는 메소드에서 자기 클래스에 속한 변수에 접근할 때 사용하는 명칭으로, 일반적으로 self를 사용하지만 사용자가 임의로 지정해도 된다.
- '인수'는 메소드를 호출하는 곳에서 보낸 값을 저장할 변수로, 사용자가 임의로 지정한다.
- return은 메소드를 호출한 위치로 값을 돌려주기 위해 사용하는 예약어로, 그대로 입력한다. return 값이 없는 경우에는 생략할 수 있다.
- '값'에는 변수, 객체, 계산식 등이 올 수 있다.

• 객체의 선언 형식

```
변수명 = 클래스명( )
```

변수명은 사용자가 임의로 지정하고, 사전에 정의한 클래스명과 괄호( )를 적는다.

예제 다음은 두 수를 교환하는 프로그램을 Python으로 구현한 것이다.

```
class Cls:                  Cls 클래스 정의부의 시작점이다. 여기서부터 ❼번까지가 클래스 정의부에 해당한다.
    x, y = 10, 20           Cls 클래스의 변수(속성) x와 y를 선언하고, 각각 10과 20으로 초기화한다.
❹   def chg(self):
❺       temp = self.x
❻       self.x = self.y
❼       self.y = temp
❶ a = Cls( )
❷ print(a.x, a.y)
❸ a.chg( )
❽ print(a.x, a.y)
```

코드 해설

❶ Cls 클래스의 객체 a를 생성한다. 객체 a는 Cls의 속성 x, y와 메소드 chg( )를 갖는다.
- a : 사용자 정의 변수다. 사용자가 임의로 지정함
- Cls( ) : 클래스의 이름이다. 괄호( )를 붙여 그대로 적음

	a.x	a.y
a	10	20

❷ a 객체의 속성 x와 y를 출력한다.
- 객체와 속성은 .(마침표)로 연결한다.

결과
```
10 20
```

❸ a 객체의 메소드 chg를 호출한다. ❹번으로 이동한다.
- 객체와 메소드는 .(마침표)로 연결한 후 괄호( )를 붙여 적는다.

❹ a 객체의 메소드 chg의 시작점이다. 별도로 사용되는 인수가 없으므로 괄호( )에는 self만 적는다.

❺ a 객체의 속성 x의 값을 temp에 저장한다.
- self : 메소드 안에서 사용되는 self는 자신이 속한 클래스를 의미함
- self.x : a.x와 동일함

			a.x	a.y
temp	10	a	10	20

❻ a 객체의 속성 y의 값을 a 객체의 속성 x에 저장한다.

			a.x	a.y
temp	10	a	20	20

❼ temp의 값을 a 객체의 속성 y에 저장한다. 메소드 chg가 종료되었으므로 메소드를 호출한 다음 문장인 ❽번으로 제어를 옮긴다.

			a.x	a.y
temp	10	a	20	10

❽ a 객체의 속성 x와 y를 출력한다.

결과
```
10 20
20 10
```

23.7, 23.2, 21.8

401232

핵심 274 클래스 없는 메소드의 사용

C언어의 사용자 정의 함수와 같이 클래스 없이 메소드만 단독으로 사용할 수 있다.

예제 다음 프로그램의 실행 결과를 확인하시오.

```
def calc(x, y):          ❸ 메소드 calc의 시작점이다. ❷번에서 calc(a, b)라고 했으므로 x는 a의 값 3을 받고, y는 b의 값 12를 받는다.
    x *= 3               ❹ x = x * 3이므로 x는 9가 된다.
    y /= 3               ❺ y = y / 3이므로 y는 4가 된다.
    print(x, y)          ❻ 결과 9 4.0
    return x             ❼ x의 값을 반환한다. x의 값 9를 ❷번의 a에 저장한 후 제어를 ❽번으로 옮긴다.

a, b = 3, 12             ❶ 변수 a와 b에 3과 12를 저장한다.
a = calc(a, b)           ❷ a, b 즉 3과 12를 인수로 하여 calc 메소드를 호출한 결과를 a에 저장한다. ❸번으로 이동한다.
print(a, b)              ❽ 결과
                           9 4.0
                           9 12
```

401403

핵심 275 절차적 프로그래밍 언어의 종류

언어	특징
C	• 1972년 미국 벨 연구소의 데니스 리치에 의해 개발됨 • 시스템 소프트웨어를 개발하기 편리하여 시스템 프로그래밍 언어로 널리 사용됨 • 자료의 주소를 조작할 수 있는 포인터를 제공함 • 고급 프로그래밍 언어이면서 저급 프로그램 언어의 특징을 모두 갖춤 • UNIX의 일부가 C 언어로 구현됨 • 컴파일러 방식의 언어 • 이식성이 좋아 컴퓨터 기종에 관계없이 프로그램을 작성할 수 있음
ALGOL	• 수치 계산이나 논리 연산을 위한 과학 기술 계산용 언어 • PASCAL과 C 언어의 모체가 됨
COBOL	• 사무 처리용 언어 • 영어 문장 형식으로 구성되어 있어 이해와 사용이 쉬움 • 4개의 DIVISION으로 구성되어 있음
FORTRAN	• 과학 기술 계산용 언어임 • 수학과 공학 분야의 공식이나 수식과 같은 형태로 프로그래밍 할 수 있음

핵심 276 객체지향 프로그래밍 언어의 종류

401503

언어	특징
JAVA	• 분산 네트워크 환경에 적용이 가능하며, 멀티스레드 기능을 제공하므로 여러 작업을 동시에 처리할 수 있음 • 운영체제 및 하드웨어에 독립적이며, 이식성이 강함 • 캡슐화가 가능하고 재사용성 높음
C++	• C 언어에 객체지향 개념을 적용한 언어 • 모든 문제를 객체로 모델링하여 표현함
Smalltalk	• 1세대 객체지향 프로그래밍 언어 중 하나로 순수한 객체지향 프로그래밍 언어 • 최초로 GUI를 제공한 언어

21.8, 21.5, 20.8, 20.6

401604

핵심 277 스크립트 언어의 종류

자바 스크립트 (JAVA Script)	• 웹 페이지의 동작을 제어하는 데 사용되는 클라이언트용 스크립트 언어로, 클래스가 존재하지 않으며 변수 선언도 필요 없음 객체지향 언어이다. • 서버에서 데이터를 전송할 때 아이디, 비밀번호, 수량 등의 입력 사항을 확인하기 위한 용도로 많이 사용됨 Prototype Link와 Prototype Object를 활용한다 Haskell
VB 스크립트 (Visual Basic Script)	마이크로소프트 사에서 자바 스크립트에 대응하기 위해 제작한 언어로, Active X를 사용하여 마이크로소프트 사의 애플리케이션들을 컨트롤할 수 있음
ASP(Active Server Page)	• 서버 측에서 동적으로 수행되는 페이지를 만들기 위한 언어로 마이크로 소프트 사에서 제작함 • Windows 계열에서만 수행 가능한 프로그래밍 언어
JSP(Java Server Page)	JAVA로 만들어진 서버용 스크립트로, 다양한 운영체제에서 사용이 가능함
PHP (Professional Hypertext Preprocessor)	• 서버용 스크립트 언어로, Linux, Unix, Windows 운영체제에서 사용 가능함 • C, Java 등과 문법이 유사하므로 배우기 쉬워 웹 페이지 제작에 많이 사용됨
파이썬 (Python)	객체지향 기능을 지원하는 대화형 인터프리터 언어로, 플랫폼에 독립적이고 문법이 간단하여 배우기 쉬움
쉘 스크립트	• 유닉스/리눅스 계열의 쉘(Shell)에서 사용되는 명령어들의 조합으로 구성된 스크립트 언어 • 컴파일 단계가 없어 실행 속도가 빠름 • 저장 시 확장자로 '.sh'가 붙음 • 쉘의 종류 : Bash Shell, Bourne Shell, C Shell, Korn Shell 등 • 쉘 스크립트에서 사용되는 제어문 – 선택형 : if, case – 반복형 : for, while, until
Basic	절차지향 기능을 지원하는 대화형 인터프리터 언어로, 초보자도 쉽게 사용할 수 있는 문법 구조를 가짐

401703

핵심 278 선언형 프로그래밍 언어 종류

HTML	인터넷의 표준 문서인 하이퍼텍스트 문서를 만들기 위해 사용하는 언어로, 특별한 데이터 타입이 없는 단순한 텍스트이므로 호환성이 좋고 사용이 편리함
LISP	• 인공지능 분야에 사용되는 언어 • 기본 자료 구조가 연결 리스트 구조이며, 재귀(Recursion) 호출을 많이 사용함
PROLOG	논리학을 기초로 한 고급 언어로, 인공 지능 분야에서의 논리적인 추론이나 리스트 처리 등에 주로 사용됨
XML	• 기존 HTML의 단점을 보완하여 웹에서 구조화된 폭넓고 다양한 문서들을 상호 교환할 수 있도록 설계된 언어 • HTML에 사용자가 새로운 태그(Tag)를 정의할 수 있으며, 문서의 내용과 이를 표현하는 방식이 독립적임
Haskell	함수형 프로그래밍 언어로 부작용(Side Effect)이 없음 코드가 간결하고 에러 발생 가능성이 낮음

21.3

401301

핵심 279 라이브러리

라이브러리는 프로그램을 효율적으로 개발할 수 있도록 자주 사용하는 함수나 데이터들을 미리 만들어 모아 놓은 집합체이다.

- 자주 사용하는 함수들의 반복적인 코드 작성을 피하기 위해 미리 만들어 놓은 것으로, 필요할 때는 언제든지 호출하여 사용할 수 있다.
- 라이브러리에는 표준 라이브러리와 외부 라이브러리가 있다.
- **표준 라이브러리** : 프로그래밍 언어에 기본적으로 포함되어 있는 라이브러리로, 여러 종류의 모듈이나 패키지 형태
- **외부 라이브러리** : 개발자들이 필요한 기능들을 만들어 인터넷 등에 공유해 놓은 것으로, 외부 라이브러리를 다운받아 설치한 후 사용함

23.5, 23.2, 22.7, 21.5, 21.3

핵심 280 C언어의 대표적인 표준 라이브러리

401302

C언어는 라이브러리를 헤더 파일로 제공하는데, 각 헤더 파일에는 응용 프로그램 개발에 필요한 함수들이 정리되어 있다.

- C언어에서 헤더 파일을 사용하려면 '#include 〈stdio.h〉'와 같이 include문을 이용해 선언한 후 사용해야 한다.

헤더 파일	기능
stdio.h in/out	• 데이터의 입 · 출력에 사용되는 기능들을 제공함 • 주요 함수 : printf, scanf, fprintf, fscanf, fclose, fopen 등
math.h	• 수학 함수들을 제공함 • 주요 함수 : sqrt, pow, abs 등
string.h	• 문자열 처리에 사용되는 기능들을 제공함 • 주요 함수 : strlen, strcpy, strcmp 등
stdlib.h	• 자료형 변환, 난수 발생, 메모리 할당에 사용되는 기능들을 제공함 • 주요 함수 : atoi, atof, srand, rand, malloc, free 등
time.h	• 시간 처리에 사용되는 기능들을 제공함 • 주요 함수 : time, clock 등

핵심 281 예외 처리

401801

프로그램의 정상적인 실행을 방해하는 조건이나 상태를 예외(Exception)라고 하며, 이러한 예외가 발생했을 때 프로그래머가 해당 문제에 대비해 작성해 놓은 처리 루틴을 수행하도록 하는 것을 예외 처리(Exception Handling)라고 한다.

- 예외가 발생했을 때 일반적인 처리 루틴은 프로그램을 종료시키거나 로그를 남기도록 하는 것이다.
- C++, Ada, JAVA, 자바스크립트와 같은 언어에는 예외 처리 기능이 내장되어 있으며, 그 외의 언어에서는 필요한 경우 조건문을 이용해 예외 처리 루틴을 작성한다.
- 예외의 원인에는 컴퓨터 하드웨어 문제, 운영체제의 설정 실수, 라이브러리 손상, 사용자의 입력 실수, 받아들일 수 없는 연산, 할당하지 못하는 기억장치 접근 등 다양하다.

핵심 282 운영체제의 정의 및 목적

2459936

운영체제(OS; Operating System)는 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효과적으로 사용할 수 있도록 환경을 제공하는 여러 프로그램의 모임이다.

- 컴퓨터 사용자와 컴퓨터 하드웨어 간의 인터페이스로서 동작하는 시스템 소프트웨어의 일종으로, 다른 응용 프로그램이 유용한 작업을 할 수 있도록 환경을 제공해 준다.
- 운영체제의 목적에는 처리 능력 향상, 사용 가능도 향상, 신뢰도 향상, 반환 시간 단축 등이 있다.
- 처리 능력, 반환 시간, 사용 가능도, 신뢰도는 운영체제의 성능을 평가하는 기준이 된다.

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

21.3

핵심 283 운영체제의 구성

2414903

제어 프로그램

제어 프로그램(Control Program)은 컴퓨터 전체의 작동 상태 감시, 작업의 순서 지정, 작업에 사용되는 데이터 관리 등의 역할을 수행하는 것으로 다음과 같이 구분할 수 있다.

감시 프로그램 (Supervisor Program)	제어 프로그램 중 가장 핵심적인 역할을 하는 것으로, 자원의 할당 및 시스템 전체의 작동 상태를 감시하는 프로그램
작업 관리 프로그램 (Job Management Program)	작업이 정상적으로 처리될 수 있도록 작업의 순서와 방법을 관리하는 프로그램
데이터 관리 프로그램 (Data Management Program)	작업에 사용되는 데이터와 파일의 표준적인 처리 및 전송을 관리하는 프로그램

처리 프로그램

처리 프로그램(Processing Program)은 제어 프로그램의 지시를 받아 사용자가 요구한 문제를 해결하기 위한 프로그램으로, 다음과 같이 구분할 수 있다.

언어 번역 프로그램	사용자가 고급언어로 작성한 원시 프로그램을 기계어 형태의 목적 프로그램으로 변환시키는 것으로, 컴파일러, 어셈블러, 인터프리터 등이 있음
서비스 프로그램	• 사용자가 컴퓨터를 더욱 효율적으로 사용할 수 있도록 제작된 프로그램 • 분류/병합(Sort/Merge), 유틸리티 프로그램 등이 여기에 해당됨

20.8

핵심 284 운영체제의 기능

2414904

- 프로세서(처리기, Processor), 기억장치(주기억장치, 보조기억장치), 입 · 출력장치, 파일 및 정보 등의 자원을 관리한다.
- 자원을 효율적으로 관리하기 위해 자원의 스케줄링 기능을 제공한다.
- 사용자와 시스템 간의 편리한 인터페이스를 제공한다.
- 시스템의 각종 하드웨어와 네트워크를 관리 · 제어한다.
- 데이터를 관리하고, 데이터 및 자원의 공유 기능을 제공한다.
- 시스템의 오류를 검사하고 복구한다.
- 자원 보호 기능을 제공한다.
- 입 · 출력에 대한 보조 기능을 제공한다.
- 가상 계산기 기능을 제공한다.

핵심 285 Windows

2459937

- Windows는 1990년대 마이크로소프트(Microsoft) 사가 개발한 운영체제이다.
- Windows의 주요 특징

그래픽 사용자 인터페이스 (GUI; Graphic User Interface)	키보드로 명령어를 직접 입력하지 않고, 마우스로 아이콘이나 메뉴를 선택하여 모든 작업을 수행하는 방식
선점형 멀티태스킹 (Preemptive Multi-Tasking)	동시에 여러 개의 프로그램을 실행하는 멀티태스킹을 하면서 운영체제가 각 작업의 CPU 이용 시간을 제어하여 응용 프로그램 실행중 문제가 발생하면 해당 프로그램을 강제 종료시키고 모든 시스템 자원을 반환하는 방식
PnP(Plug and Play, 자동 감지 기능)	컴퓨터 시스템에 프린터나 사운드 카드 등의 하드웨어를 설치했을 때, 해당 하드웨어를 사용하는 데 필요한 시스템 환경을 운영체제가 자동으로 구성해 주는 기능
OLE(Object Linking and Embedding)	다른 여러 응용 프로그램에서 작성된 문자나 그림 등의 개체(Object)를 현재 작성 중인 문서에 자유롭게 연결(Linking)하거나 삽입(Embedding)하여 편집할 수 있게 하는 기능
255자의 긴 파일명	• Windows에서는 파일 이름을 지정할 때 VFAT(Virtual File Allocation Table)를 이용하여 최대 255자까지 지정할 수 있음 • 파일 이름으로는 ₩ / : * ? " 〈 〉 |를 제외한 모든 문자 및 공백을 사용할 수 있으며, 한글의 경우 127자까지 지정할 수 있음
Single-User 시스템	컴퓨터 한 대를 한 사람만이 독점해서 사용함

22.4

핵심 286 UNIX의 개요 및 특징

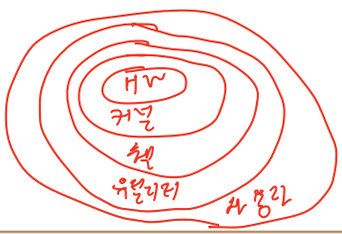
2415101

UNIX는 1960년대 AT&T 벨(Bell) 연구소, MIT, General Electric이 공동 개발한 운영체제이다.

- 시분할 시스템(Time Sharing System)을 위해 설계된 대화식 운영체제로, 소스가 공개된 개방형 시스템(Open System)이다.
- 대부분 C 언어로 작성되어 있어 이식성이 높으며 장치, 프로세스 간의 호환성이 높다.
- 크기가 작고 이해하기가 쉽다.
- 다중 사용자(Multi-User), 다중 작업(Multi-Tasking)을 지원한다.
- 많은 네트워킹 기능을 제공하므로 통신망(Network) 관리용 운영체제로 적합하다.
- 트리 구조의 파일 시스템을 갖는다.
- 전문적인 프로그램 개발에 용이하다.
- 다양한 유틸리티 프로그램들이 존재한다.

※ 다중 사용자(Multi-User), 다중 작업(Multi-Tasking)

- 다중 사용자(Multi-User)는 여러 사용자가 동시에 시스템을 사용하는 것이고, 다중 작업(Multi-Tasking)은 여러 개의 작업이나 프로그램을 동시에 수행하는 것을 의미한다.
- 하나 이상의 작업을 백그라운드에서 수행하므로 여러 작업을 동시에 처리할 수 있다.



22.3, 20.9, 20.6

핵심 287 UNIX 시스템의 구성

2415102

커널(Kernel)

- UNIX의 가장 핵심적인 부분이다.
- 컴퓨터가 부팅될 때 주기억장치에 적재된 후 상주하면서 실행된다.
- 하드웨어를 보호하고, 프로그램과 하드웨어 간의 인터페이스 역할을 담당한다.
- 프로세스(CPU 스케줄링) 관리, 기억장치 관리, 파일 관리, 입·출력 관리, 프로세스간 통신, 데이터 전송 및 변환 등 여러 가지 기능을 수행한다.

쉘(Shell)

- 사용자의 명령어를 인식하여 프로그램을 호출하고 명령을 수행하는 명령어 해석기이다.
- 시스템과 사용자 간의 인터페이스를 담당한다.
- DOS의 COMMAND.COM과 같은 기능을 수행한다.
- 주기억장치에 상주하지 않고, 명령어가 포함된 파일 형태로 존재하며 보조 기억장치에서 교체 처리가 가능하다.
- 파이프라인 기능을 지원하고 입·출력 재지정을 통해 출력과 입력의 방향을 변경할 수 있다.
- 공용 Shell(Bourne Shell, C Shell, Korn Shell)이나 사용자 자신이 만든 Shell을 사용할 수 있다.

Utility Program

- 일반 사용자가 작성한 응용 프로그램을 처리하는 데 사용한다.
- DOS에서의 외부 명령어에 해당된다.
- 유틸리티 프로그램에는 에디터, 컴파일러, 인터프리터, 디버거 등이 있다.

21.8

핵심 288 파일 디스크립터 (File Descriptor)

2415133

파일을 관리하기 위한 시스템(운영체제)이 필요로 하는 파일에 대한 정보를 가진 제어 블록을 의미하며, 파일 제어 블록(FCB; File Control Block)이라고도 한다.

- 파일 디스크립터는 파일마다 독립적으로 존재하며, 시스템에 따라 다른 구조를 가질 수 있다.
- 보통 파일 디스크립터는 보조기억장치 내에 저장되어 있다가 해당 파일이 Open될 때 주기억장치로 옮겨진다.
- 파일 디스크립터는 파일 시스템이 관리하므로 사용자가 직접 참조할 수 없다.

23.7, 22.3, 21.3, 20.8

핵심 289 기억장치 관리 – 배치(Placement) 전략

2415204

배치 전략은 새로 반입되는 프로그램이나 데이터를 주기억장치의 어디에 위치시킬 것인지를 결정하는 전략이다.

최초 적합 (First Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 첫 번째 분할 영역에 배치시키는 방법
최적 적합 (Best Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 작게 남기는 분할 영역에 배치시키는 방법
최악 적합 (Worst Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치시키는 방법

단편화 많게

23.2, 21.3

핵심 290 페이징(Paging) 기법

2415402

- 페이징 기법은 가상기억장치에 보관되어 있는 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 후 나눠진 프로그램(페이지)을 동일하게 나눠진 주기억장치의 영역(페이지 프레임)에 적재시켜 실행하는 기법이다.
- 프로그램을 일정한 크기로 나눈 단위를 페이지(Page)라고 하고, 페이지 크기로 일정하게 나누어진 주기억장치의 단위를 페이지 프레임(Page Frame)이라고 한다.
- 외부 단편화는 발생하지 않으나 내부 단편화는 발생할 수 있다.
- 주소 변환을 위해서 페이지의 위치 정보를 가지고 있는 페이지 맵 테이블(Page Map Table)이 필요하다.

↳ 사용하면 비용 ↑, 처리 속도 ↓

23.2, 21.3, 20.9

핵심 291 세그먼테이션 (Segmentation) 기법

2415403

- 세그먼테이션 기법은 가상기억장치에 보관되어 있는 프로그램을 다양한 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 실행시키는 기법이다.
- 프로그램을 배열이나 함수 등과 같은 논리적인 크기로 나눈 단위를 세그먼트(Segment)라고 하며, 각 세그먼트는 고유한 이름과 크기를 갖는다.
- 주소 변환을 위해서 세그먼트가 존재하는 위치 정보를 가지고 있는 세그먼트 맵 테이블(Segment Map Table)이 필요하다.
- 내부 단편화는 발생하지 않으나 외부 단편화는 발생할 수 있다.

23.7, 23.2, 22.7, 22.4, 22.3, 21.8, 20.9, 20.6

핵심 292 페이지 교체 알고리즘

2415404

페이지 교체 알고리즘은 페이지 부재(Page Fault)가 발생했을 때 가상기억장치의 필요한 페이지를 주기억장치에 적재해야 하는데, 이때 주기억장치의 모든 페이지 프레임이 사용중이면 어떤 페이지 프레임을 선택하여 교체할 것인지를 결정하는 기법이다.

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
FIFO(First In First Out)	• 각 페이지가 주기억장치에 적재될 때마다 그때의 시간을 기억시켜 가장 먼저 들어와서 가장 오래 있었던 페이지를 교체하는 기법 • 이해하기 쉽고, 프로그래밍 및 설계가 간단함
LRU(Least Recently Used)	• 최근에 가장 오랫동안 사용하지 않은 페이지를 교체하는 기법 • 각 페이지마다 계수기(Counter)나 스택(Stack)을 두어 현 시점에서 가장 오랫동안 사용하지 않은, 즉 가장 오래 전에 사용된 페이지를 교체함
LFU (Least Frequently Used)	• 사용 빈도가 가장 적은 페이지를 교체하는 기법 • 활발하게 사용되는 페이지는 사용 횟수가 많아 교체되지 않고 사용됨
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
NUR (Not Used Recently)	• LRU와 비슷한 알고리즘으로, 최근에 사용하지 않은 페이지를 교체하는 기법 • 최근에 사용되지 않은 페이지는 향후에도 사용되지 않을 가능성이 높다는 것을 전제로, LRU에서 나타나는 시간적인 오버헤드를 줄일 수 있음 • 최근의 사용 여부를 확인하기 위해서 각 페이지마다 두 개의 비트, 즉 참조 비트(Reference Bit)와 변형 비트(Modified Bit, Dirty Bit)가 사용됨

21.5

핵심 293 페이지 크기

2415501

페이지 크기가 작을 경우 페이지의 수는 늘어난다.

- 페이지 단편화가 감소되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 줄어든다.
- 불필요한 내용이 주기억장치에 적재될 확률이 적으므로 효율적인 워킹 셋을 유지할 수 있다.
- Locality에 더 일치할 수 있기 때문에 기억장치 효율이 높아진다.
- 페이지 정보를 갖는 페이지 맵 테이블의 크기가 커지고, 매핑 속도가 늦어진다.
- 디스크 접근 횟수가 많아져서 전체적인 입 · 출력 시간은 늘어난다.

페이지 크기가 클 경우 페이지의 수는 줄어든다.

- 페이지 정보를 갖는 페이지 맵 테이블의 크기가 작아지고, 매핑 속도가 빨라진다.
- 디스크 접근 횟수가 줄어들어 전체적인 입 · 출력의 효율성이 증가된다.
- 페이지 단편화가 증가되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 늘어난다.
- 프로세스(프로그램) 수행에 불필요한 내용까지도 주기억장치에 적재될 수 있다.

23.5, 21.5

핵심 294 Locality

2415502

Locality(국부성, 지역성, 구역성, 국소성)는 프로세스가 실행되는 동안 주기억장치를 참조할 때 일부 페이지만 집중적으로 참조하는 성질이 있다는 이론이다.

- 스래싱을 방지하기 위한 워킹 셋 이론의 기반이 되었다.
- Locality의 종류에는 시간 구역성(Temporal Locality)과 공간 구역성(Spatial Locality)이 있다.

시간 구역성(Temporal Locality)

- 시간 구역성은 프로세스가 실행되면서 하나의 페이지를 일정 시간 동안 집중적으로 액세스하는 현상이다.
- 한 번 참조한 페이지는 가까운 시간 내에 계속 참조할 가능성이 높음을 의미한다.
- 시간 구역성이 이루어지는 기억 장소 : Loop(반복, 순환), 스택(Stack), 부 프로그램(Sub Routine), Counting(1씩 증감), 집계(Totaling)에 사용되는 변수(기억장소)

공간 구역성(Spatial Locality)

- 공간 구역성은 프로세스 실행 시 일정 위치의 페이지를 집중적으로 액세스하는 현상이다.
- 어느 하나의 페이지를 참조하면 그 근처의 페이지를 계속 참조할 가능성이 높음을 의미한다.
- 공간 구역성이 이루어지는 기억장소 : 배열 순회(Array Traversal, 배열 순례), 순차적 코드의 실행, 프로그래머들이 관련된 변수(데이터를 저장할 기억장소)들을 서로 근처에 선언하여 할당되는 기억장소, 같은 영역에 있는 변수를 참조할 때 사용

21.3

핵심 295 워킹 셋(Working Set)

2415503

워킹 셋은 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합이다.

- 데닝(Denning)이 제안한 프로그램의 움직임에 대한 모델로, 프로그램의 Locality 특징을 이용한다.
- 자주 참조되는 워킹 셋을 주기억장치에 상주시킴으로써 페이지 부재 및 페이지 교체 현상이 줄어들어 프로세스의 기억장치 사용이 안정된다.
- 시간이 지남에 따라 자주 참조하는 페이지들의 집합이 변화하기 때문에 워킹 셋은 시간에 따라 변경된다.

23.2, 22.7, 21.5

핵심 296 스래싱(Thrashing)

2415505

프로세스의 처리 시간보다 페이지 교체에 소요되는 시간이 더 많아지는 현상이다.

- 다중 프로그래밍 시스템이나 가상기억장치를 사용하는 시스템에서 하나의 프로세스 수행 과정중 자주 페이지 부재가 발생함으로써 나타나는 현상으로, 전체 시스템의 성능이 저하된다.
- 다중 프로그래밍의 정도가 높아짐에 따라 CPU의 이용률은 어느 특정 시점까지는 높아지지만, 다중 프로그래밍의 정도가 더욱 커지면 스래싱이 나타나고, CPU의 이용률은 급격히 감소하게 된다.
- 스래싱 현상 방지 방법
  - 다중 프로그래밍의 정도를 적정 수준으로 유지한다.
  - 페이지 부재 빈도(Page Fault Frequency)를 조절하여 사용한다.
  - 워킹 셋을 유지한다.
  - 부족한 자원을 증설하고, 일부 프로세스를 중단시킨다.
  - CPU 성능에 대한 자료의 지속적 관리 및 분석으로 임계치를 예상하여 운영한다.

22.7

핵심 297 프로세스(Process)의 정의

2415601

프로세스는 일반적으로 프로세서(처리기, CPU)에 의해 처리되는 사용자 프로그램, 시스템 프로그램, 즉 실행중인 프로그램을 의미하며, 작업(Job), 태스크(Task)라고도 한다.

- 프로세스는 다음과 같이 여러 형태로 정의할 수 있다.
  - PCB를 가진 프로그램
  - 실기억장치에 저장된 프로그램
  - 프로세서가 할당되는 실체로서, 디스패치가 가능한 단위
  - 프로시저가 활동중인 것
  - 비동기적 행위를 일으키는 주체
  - 지정된 결과를 얻기 위한 일련의 계통적 동작
  - 목적 또는 결과에 따라 발생되는 사건들의 과정
  - 운영체제가 관리하는 실행 단위

21.8

핵심 298 PCB

2415602

PCB(Process Control Block, 프로세스 제어 블록)는 운영체제가 프로세스에 대한 중요한 정보를 저장해 놓는 곳으로, Task Control Block 또는 Job Control Block이라고도 한다.

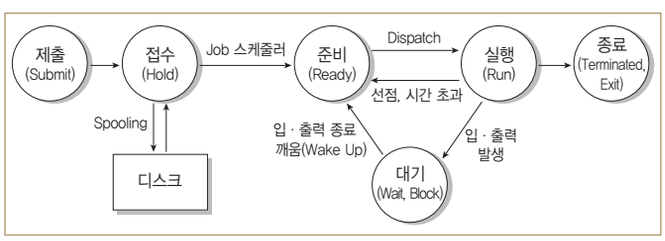
- 각 프로세스가 생성될 때마다 고유의 PCB가 생성되고, 프로세스가 완료되면 PCB는 제거된다.
- PCB에 저장되어 있는 정보
  - 프로세스의 현재 상태
  - 포인터
    - ▶ 부모 프로세스에 대한 포인터
    - ▶ 자식 프로세스에 대한 포인터
    - ▶ 프로세스가 위치한 메모리에 대한 포인터
    - ▶ 할당된 자원에 대한 포인터
  - 프로세스 고유 식별자
  - 스케줄링 및 프로세스의 우선순위
  - CPU 레지스터 정보
  - 주기억장치 관리 정보
  - 입 · 출력 상태 정보
  - 계정 정보

22.7, 20.6

핵심 299 프로세스 상태 전이

2415603

프로세스 상태 전이는 프로세스가 시스템 내에 존재하는 동안 프로세스의 상태가 변하는 것을 의미하며, 프로세스의 상태를 다음과 같이 상태 전이도로 표시할 수 있다.



- 프로세스의 상태는 제출, 접수, 준비, 실행, 대기 상태로 나눌 수 있으며, 이 중 주요 세 가지 상태는 준비, 실행, 대기 상태이다.
- 제출(Submit) : 작업을 처리하기 위해 사용자가 작업을 시스템에 제출한 상태
- 접수(Hold) : 제출된 작업이 스풀 공간인 디스크의 할당 위치에 저장된 상태
- 준비(Ready) : 프로세스가 프로세서를 할당받기 위해 기다리고 있는 상태
- 실행(Run) : 준비상태 큐에 있는 프로세스가 프로세서를 할당받아 실행되는 상태
- 대기(Wait), 보류, 블록(Block) : 프로세스에 입 · 출력 처리가 필요하면 현재 실행 중인 프로세스가 중단되고, 입 · 출력 처리가 완료될 때까지 대기하고 있는 상태
- 종료(Terminated, Exit) : 프로세스의 실행이 끝나고 프로세스 할당이 해제된 상태

21.8

핵심 300 프로세스 상태 전이 관련 용어

2415604

Dispatch	준비 상태에서 대기하고 있는 프로세스 중 하나가 프로세서를 할당받아 실행 상태로 전이되는 과정
Wake Up	입 · 출력 작업이 완료되어 프로세스가 대기 상태에서 준비 상태로 전이 되는 과정
Spooling	입 · 출력장치의 공유 및 상대적으로 느린 입 · 출력장치의 처리 속도를 보완하고 다중 프로그래밍 시스템의 성능을 향상시키기 위해 입 · 출력할 데이터를 직접 입 · 출력장치에 보내지 않고 나중에 한꺼번에 입 · 출력하기 위해 디스크에 저장하는 과정
교통량 제어기 (Traffic Controller)	프로세스의 상태에 대한 조사와 통보를 담당함

22.4, 21.8, 20.6

핵심 301 스레드(Thread)

2415605

스레드는 프로세스 내에서의 작업 단위로서 시스템의 여러 자원을 할당받아 실행하는 프로그램의 단위이다.

- 하나의 프로세스에 하나의 스레드가 존재하는 경우에는 단일 스레드, 하나 이상의 스레드가 존재하는 경우에는 다중 스레드라고 한다.

- 프로세스의 일부 특성을 갖고 있기 때문에 경량(Light Weight) 프로세스라고도 한다.
- 스레드 기반 시스템에서 스레드는 독립적인 스케줄링의 최소 단위로서 프로세스의 역할을 담당한다.
- 동일 프로세스 환경에서 서로 독립적인 다중 수행이 가능하다.
- 스레드의 분류

사용자 수준의 스레드	• 사용자가 만든 라이브러리를 사용하여 스레드를 운용함 • 속도는 빠르지만 구현이 어려움
커널 수준의 스레드	• 운영체제의 커널에 의해 스레드를 운용함 • 구현이 쉽지만 속도가 느림

- 스레드 사용의 장점
  - 하나의 프로세스를 여러 개의 스레드로 생성하여 병행성을 증진시킬 수 있다.
  - 하드웨어, 운영체제의 성능과 응용 프로그램의 처리율을 향상시킬 수 있다.
  - 응용 프로그램의 응답 시간(Response Time)을 단축시킬 수 있다.
  - 실행 환경을 공유시켜 기억장소의 낭비가 줄어든다.
  - 프로세스들 간의 통신이 향상된다.
  - 스레드는 공통적으로 접근 가능한 기억장치를 통해 효율적으로 통신한다.

22.7, 22.4, 20.9, 20.8, 20.6

핵심 302 주요 스케줄링 알고리즘

2459938

FCFS(First Come First Service, 선입 선출) = FIFO(First In First Out)

FCFS는 준비상태 큐(대기 큐, 준비 완료 리스트, 작업준비 큐, 스케줄링 큐)에 도착한 순서에 따라 차례로 CPU를 할당하는 기법으로, 가장 간단한 알고리즘이다.

- 먼저 도착한 것이 먼저 처리되어 공평성은 유지되지만 짧은 작업이 긴 작업을, 중요한 작업이 중요하지 않은 작업을 기다리게 된다.

SJF(Shortest Job First, 단기 작업 우선)

SJF는 준비상태 큐에서 기다리고 있는 프로세스들 중에서 실행 시간이 가장 짧은 프로세스에게 먼저 CPU를 할당하는 기법이다.

- 가장 적은 평균 대기 시간을 제공하는 최적 알고리즘이다.

HRN(Hightest Response-ratio Next)

실행 시간이 긴 프로세스에 불리한 SJF 기법을 보완하기 위한 것으로, 대기 시간과 서비스(실행) 시간을 이용하는 기법이다.

- 우선순위 계산 공식을 이용하여 서비스(실행) 시간이 짧은 프로세스나 대기 시간이 긴 프로세스에게 우선순위를 주어 CPU를 할당한다.
- 서비스 실행 시간이 짧거나 대기 시간이 긴 프로세스일 경우 우선순위가 높아진다.
- 우선순위를 계산하여 그 숫자가 가장 높은 것부터 낮은 순으로 우선순위가 부여된다.
- 우선순위 계산식

$$\text{우선순위 계산식} = \frac{\text{대기 시간+서비스 시간}}{\text{서비스 시간}}$$

20.9

핵심 303 UNIX / LINUX의 주요 환경 변수

2415903

UNIX나 LINUX에서 환경 변수를 명령어나 스크립트에서 사용하려면 변수명 앞에 '$'를 입력해야 한다.

- UINIX나 LINUX에서는 set, env, printenv, setenv 중 하나를 입력하면 모든 환경 변수와 값을 표시한다.

환경 변수	용도
$DISPLAY	현재 X 윈도 디스플레이 위치
$PS1	쉘 프롬프트 정보
$HOME	사용자의 홈 디렉터리
$PWD	현재 작업하는 디렉터리
$LANG	프로그램 사용 시 기본적으로 지원되는 언어
$TERM	로긴 터미널 타입
$MAIL	메일을 보관하는 경로
$USER	사용자의 이름
$PATH	실행 파일을 찾는 경로

23.7, 23.5, 22.7, 21.3, 20.8

핵심 304 UNIX / LINUX 기본 명령어

2416003

명령어	기능
cat	파일 내용을 화면에 표시함
chdir	현재 사용할 디렉터리의 위치를 변경함
chmod	파일의 보호 모드를 설정하여 파일의 사용 허가를 지정함
chown	소유자를 변경함
cp	파일을 복사함
exec	새로운 프로세스를 수행함
find	파일을 찾음
fork	새로운 프로세스를 생성함(하위 프로세스 호출, 프로세스 복제 명령)
fsck	파일 시스템을 검사하고 보수함
getpid	자신의 프로세스 아이디를 얻음
getppid	부모 프로세스 아이디를 얻음
ls	현재 디렉터리 내의 파일 목록을 확인함
mount/unmount	파일 시스템을 마운팅한다/마운팅 해제함
rm	파일을 삭제함
wait	fork 후 exec에 의해 실행되는 프로세스의 상위 프로세스가 하위 프로세스 종료 등의 event를 기다림
uname	시스템의 이름과 버전, 네트워크 호스트명 등의 시스템 정보를 표시한다.

2개

23.5, 22.3, 21.8

핵심 305 IP 주소 (Internet Protocol Address)

2416102

IP 주소는 인터넷에 연결된 모든 컴퓨터 자원을 구분하기 위한 고유한 주소이다.

- 숫자로 8비트씩 4부분, 총 32비트로 구성되어 있다.
- IP 주소는 네트워크 부분의 길이에 따라 다음과 같이 A 클래스에서 E 클래스까지 총 5단계로 구성되어 있다.

A Class	국가나 대형 통신망에 사용 (0~127로 시작) 2^{24} = 16,777,216개의 호스트 사용 가능	8bit 1 8 9 16 17 24 25 32bit
B Class	중대형 통신망에 사용 (128~191로 시작) 2^{16} = 65,536개의 호스트 사용 가능	
C Class	소규모 통신망에 사용 (192~223으로 시작) 2^{8} = 256개의 호스트 사용 가능	
D Class	멀티캐스트용으로 사용 (224~239로 시작)	네트워크 부분 호스트 부분
E Class	실험적 주소이며 공용되지 않음	

21.8, 21.5, 20.8 에기컬고 P158

핵심 304 서브네팅(Subnetting)

2459939

서브네팅은 할당된 네트워크 주소를 다시 여러 개의 작은 네트워크로 나누어 사용하는 것을 말한다.

- 4바이트의 IP 주소 중 네트워크 주소와 호스트 주소를 구분하기 위한 비트를 서브넷 마스크(Subnet Mask)라고 하며, 이를 변경하여 네트워크 주소를 여러 개로 분할하여 사용한다.
- 서브넷 마스크는 각 클래스마다 다르게 사용된다.

23.7, 23.5, 23.2, 22.7, 22.3, 21.3, 20.8, 20.6

핵심 307 IPv6(Internet Protocol version 6)

2416104

IPv6은 현재 사용하고 있는 IP 주소 체계인 IPv4의 주소 부족 문제를 해결하기 위해 개발되었다.

- 128비트의 긴 주소를 사용하여 주소 부족 문제를 해결할 수 있으며, IPv4에 비해 자료 전송 속도가 빠르다.
- 인증성, 기밀성, 데이터 무결성의 지원으로 보안 문제를 해결할 수 있다.
- IPv4와 호환성이 뛰어나다.
- 주소의 확장성, 융통성, 연동성이 뛰어나며, 실시간 흐름 제어로 향상된 멀티미디어 기능을 지원한다.
- 패킷 크기를 확장할 수 있으므로 패킷 크기에 제한이 없다.

21.3, 20.6

핵심 308 IPv6의 구성

2416105

- 16비트씩 8부분, 총 128비트로 구성되어 있다.
- 각 부분을 16진수로 표현하고, 콜론(:)으로 구분한다.
- IPv6은 다음과 같이 세 가지 주소 체계로 나누어진다.

유니캐스트 (Unicast)	단일 송신자와 단일 수신자 간의 통신(1 대 1 통신에 사용)
멀티캐스트 (Multicast)	단일 송신자와 다중 수신자 간의 통신(1 대 다 통신에 사용)
애니캐스트 (Anycast)	단일 송신자와 가장 가까이 있는 단일 수신자 간의 통신(1 대 1 통신에 사용)

23.1, 23.5, 23.2, 22.7, 22.3, 21.5, 21.3, 20.9, 20.8, 20.6

핵심 309 OSI 참조 모델

2459940

- 다른 시스템 간의 원활한 통신을 위해 ISO(국제표준화기구)에서 제안한 통신 규약(Protocol)이다.
- OSI 7계층은 1~3 계층을 하위 계층, 4~7 계층을 상위 계층이라고 한다.
  - 하위 계층 : 물리 계층 → 데이터 링크 계층 → 네트워크 계층
  - 상위 계층 : 전송 계층 → 세션 계층 → 표현 계층 → 응용 계층

물리 계층 (Physical Layer)	전송에 필요한 두 장치 간의 실제 접속과 절단 등 기계적, 전기적, 기능적, 절차적 특성에 대한 규칙을 정의함
데이터 링크 계층 (Data Link Layer)	• 두 개의 인접한 개방 시스템들 간에 신뢰성 있고 효율적인 정보 전송을 할 수 있도록 시스템 간 연결 설정과 유지 및 종료를 담당함 • 송신 측과 수신 측의 속도 차이를 해결하기 위한 흐름 제어 기능을 함 • 프레임의 시작과 끝을 구분하기 위한 프레임의 동기화 기능을 함 • 오류의 검출과 회복을 위한 오류 제어 기능을 함
네트워크 계층 (Network Layer, 망 계층)	• 개방 시스템들 간의 네트워크 연결을 관리하는 기능과 데이터의 교환 및 중계 기능을 함 • 네트워크 연결을 설정, 유지, 해제하는 기능을 함 • 경로 설정(Routing), 데이터 교환 및 중계, 트래픽 제어, 패킷 정보 전송을 수행함
전송 계층 (Transport Layer)	• 논리적 안정과 균일한 데이터 전송 서비스를 제공함으로써 종단 시스템(End-to-End) 간에 투명한 데이터 전송을 가능하게 함 • 종단 시스템(End-to-End) 간의 전송 연결 설정, 데이터 전송, 연결 해제 기능을 함 • 주소 설정, 다중화(분할 및 재조립), 오류 제어, 흐름 제어를 수행함
세션 계층 (Session Layer)	• 송 · 수신 측 간의 관련성을 유지하고 대화 제어를 담당함 • 대화(회화) 구성 및 동기 제어, 데이터 교환 관리 기능을 함
표현 계층 (Presentation Layer)	• 응용 계층으로부터 받은 데이터를 세션 계층에 보내기 전에 통신에 적당한 형태로 변환하고, 세션 계층에서 받은 데이터는 응용 계층에 맞게 변환하는 기능을 함 • 서로 다른 데이터 표현 형태를 갖는 시스템 간의 상호 접속을 위해 필요한 계층 • 코드 변환, 데이터 암호화, 데이터 압축, 구문 검색, 정보 형식(포맷) 변환, 문맥 관리 기능을 함
응용 계층 (Application Layer)	사용자(응용 프로그램)가 OSI 환경에 접근할 수 있도록 서비스를 제공함

23.2, 22.3, 21.5

핵심 310 네트워크 관련 장비

2459941

네트워크 인터페이스 카드 (NIC; Network Interface Card)	컴퓨터와 컴퓨터 또는 컴퓨터와 네트워크를 연결하는 장치로, 정보 전송 시 정보가 케이블을 통해 전송될 수 있도록 정보 형태를 변경함
허브(Hub)	• 한 사무실이나 가까운 거리의 컴퓨터들을 연결하는 장치로, 각 회선을 통합적으로 관리하며, 신호 증폭 기능을 하는 리피터의 역할도 포함함 • 허브의 종류에는 더미 허브, 스위칭 허브가 있음
리피터 (Repeater)	전송되는 신호가 전송 선로의 특성 및 외부 충격 등의 요인으로 인해 원래의 형태와 다르게 왜곡되거나 약해질 경우 원래의 신호 형태로 재생하여 다시 전송하는 역할을 수행함
브리지 (Bridge)	• LAN과 LAN을 연결하거나 LAN 안에서의 컴퓨터 그룹(세그먼트)을 연결하는 기능을 수행함 • 네트워크를 분산적으로 구성할 수 있어 보안성을 높일 수 있음
스위치 (Switch)	• 브리지와 같이 LAN과 LAN을 연결하여 훨씬 더 큰 LAN을 만드는 장치 • 하드웨어를 기반으로 처리하므로 전송 속도가 빠름

네트워크 계층

라우터 (Router)	브리지와 같이 LAN과 LAN의 연결 기능에 데이터 전송의 최적 경로를 선택할 수 있는 기능이 추가된 것으로, 서로 다른 LAN이나 LAN과 WAN의 연결도 수행함
게이트웨이 (Gateway)	• 전 계층(1~7계층)의 프로토콜 구조가 다른 네트워크의 연결을 수행함 • LAN에서 다른 네트워크에 데이터를 보내거나 다른 네트워크로부터 데이터를 받아들이는 출입구 역할을 함

23.5, 21.8, 21.3

핵심 311 응용 계층의 주요 프로토콜

2416503

FTP (File Transfer Protocol)	컴퓨터와 컴퓨터 또는 컴퓨터와 인터넷 사이에서 파일을 주고받을 수 있도록 하는 원격 파일 전송 프로토콜
SMTP(Simple Mail Transfer Protocol)	전자 우편을 교환하는 서비스
TELNET	• 멀리 떨어져 있는 컴퓨터에 접속하여 자신의 컴퓨터처럼 사용할 수 있도록 해주는 서비스 • 프로그램을 실행하는 등 시스템 관리 작업을 할 수 있는 가상의 터미널(Virtual Terminal) 기능을 수행
SNMP(Simple Network Management Protocol)	TCP/IP의 네트워크 관리 프로토콜로, 라우터나 허브 등 네트워크 기기의 네트워크 정보를 네트워크 관리 시스템에 보내는 데 사용되는 표준 통신 규약
DNS (Domain Name System)	도메인 네임을 IP 주소로 매핑(Mapping)하는 시스템
HTTP(HyperText Transfer Protocol)	월드 와이드 웹(WWW)에서 HTML 문서를 송수신 하기 위한 표준 프로토콜

MQTT(Message Queuing Telemetry Transport) 발행-구독 기반의 메시징 프로토콜로, IoT 환경에서 자주 사용된다.

23.7, 22.4, 21.8, 21.5, 21.3, 20.9, 20.8, 20.6

핵심 312 전송 계층의 주요 프로토콜

2416504

TCP(Transmission Control Protocol)	• 양방향 연결(Full Duplex Connection)형 서비스를 제공함 • 스트림 위주의 전달(패킷 단위)을 함 • 신뢰성 있는 경로를 확립하고 메시지 전송을 감독함 • 순서 제어, 오류 제어, 흐름 제어 기능을 함 • TCP 프로토콜의 헤더는 기본적으로 20Byte에서 60Byte까지 사용할 수 있는데, 선택적으로 40Byte를 더 추가할 수 있으므로 최대 100Byte까지 크기를 확장할 수 있음
UDP (User Datagram Protocol)	• 데이터 전송 전에 연결을 설정하지 않는 비연결형 서비스를 제공함 • TCP에 비해 상대적으로 단순한 헤더 구조를 가지므로, 오버헤드가 적고, 흐름제어나 순서 제어가 없어 전송 속도가 빠름 • 실시간 전송에 유리하며, 신뢰성보다는 속도가 중요시되는 네트워크에서 사용됨
RTCP(Real-Time Control Protocol)	• RTP(Real-time Transport Protocol) 패킷의 전송 품질을 제어하기 위한 제어 프로토콜 • 세션(Session)에 참여한 각 참여자들에게 주기적으로 제어 정보를 전송함

23.7, 22.7, 22.3, 20.9, 20.6

핵심 313 인터넷 계층의 주요 프로토콜

2416505

IP(Internet Protocol)	• 전송할 데이터에 주소를 지정하고, 경로를 설정하는 기능을 함 • 비연결형인 데이터그램 방식을 사용하는 것으로 신뢰성이 보장되지 않음
ICMP (Internet Control Message Protocol, 인터넷 제어 메시지 프로토콜)	IP와 조합하여 통신중에 발생하는 오류의 처리와 전송 경로 변경 등을 위한 제어 메시지를 관리하는 역할을 하며, 헤더는 8Byte로 구성됨
IGMP(Internet Group Management Protocol, 인터넷 그룹 관리 프로토콜)	멀티캐스트를 지원하는 호스트나 라우터 사이에서 멀티캐스트 그룹 유지를 위해 사용됨
ARP (Address Resolution Protocol, 주소 분석 프로토콜)	호스트의 IP 주소를 호스트와 연결된 네트워크 접속 장치의 물리적 주소(MAC Address)로 바꿈
RARP(Reverse Address Resolution Protocol)	ARP와 반대로 물리적 주소를 IP 주소로 변환하는 기능을 함

22.7, 21.3

핵심 314 네트워크 액세스 계층의 주요 프로토콜

2416506

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