Professional 사전 과정

SW 문제 해결

내용

- **비트 연산**
- **진수**
- **실수**
- **기본 확률 이론 정리**
- **정수론**
  - 소수, 모듈라 연산
  - 최대공약수, 최소공배수
- **기계적 최적화**

비트 연산

비트 연산자

<table>
<tr><th>연산자</th><th>연사자의 기능</th></tr>
<tr><td>&</td><td>비트단위로 AND 연산을 한다.
예) num1 & num2</td></tr>
<tr><td>|</td><td>비트단위로 OR 연산을 한다.
예) num1 | num2</td></tr>
<tr><td>^</td><td>비트단위로 XOR 연산을 한다. (같으면 0 다르면 1)
예) num1 ^ num2</td></tr>
<tr><td>~</td><td>단항 연산자로서 피연산자의 모든 비트를 반전시킨다.
예) ~num</td></tr>
<tr><td><<</td><td>피연산자의 비트 열을 왼쪽으로 이동시킨다.
예) num << 2</td></tr>
<tr><td>>></td><td>피연산자의 비트 열을 오른쪽으로 이동시킨다.
예) num >> 2</td></tr>
</table>

1 << n

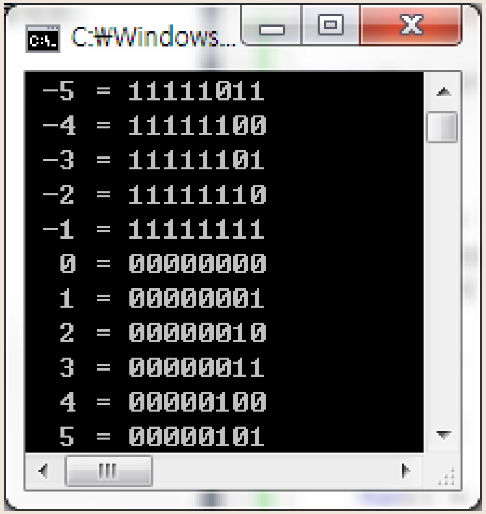
- 2^n 의 값을 갖는다.
- 원소가 n개일 경우의 모든 부분집합의 수를 의미한다.
- Power set (모든 부분 집합)
  - 공집합과 자기 자신을 포함한 모든 부분집합
  - 각 원소가 포함되거나 포함되지 않는 2가지 경우의 수를 계산하면 모든 부분집합의 수가 계산된다.

i & (1 << j)

- 계산 결과는 i의 j번째 비트가 1인지 아닌지를 의미한다.

비트 연산 예제1

```
void Bbit_print(char i)
{
    for(int j = 7; j >= 0; j--)
        putchar((i & (1 << j)) ? '1' : '0');
        // printf("%d", (i >> j) & 1 );
    putchar(' ');
}

main(void)
{
    char i;

    for(i = -5; i < 6;i++){
        printf("%3d = ", i);
        Bbit_print(i);
        putchar('\n');
    }
}
```



연습문제1

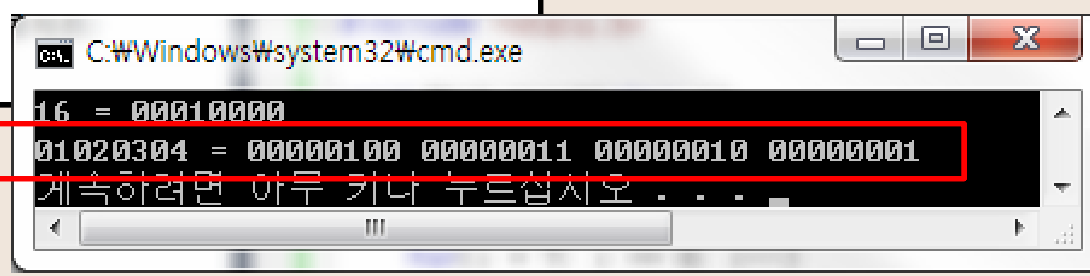
- **0과 1로 이루어진 1차 배열에서 7개 byte를 묶어서 10진수로 출력하기**
  - 예를 들어

0	0	0	0	0	0	1	0	0	0	1	1	0	1

  - 이면
    - 1, 13 을 출력한다.
  - 입력 예
    - 0000000111 1000000110 0000011110 0110000110 0001111001 1110011111 1001100111
    - 편의상 10개 단위로 간격을 두었음. 이어있는 데이터로 간주하시오.

비트 연산 예제2

```
void main(void)
{
        char *p;
        char a = 0x10;
        int x = 0x01020304, i;

        printf("%d = ", a);
        p = &a;
        Bbit_print(*p);
        putchar('\n');

        printf("0%X = ", x);
        p = (char *) &x;
        for(i = 0; i < 4;i++)
                Bbit_print(*p++);
        putchar('\n');
}
```



엔디안(Endianness)

- 컴퓨터의 메모리와 같은 1차원의 공간에 여러 개의 연속된 대상을 배열하는 방법을 의미하며 HW 아키텍처마다 다르다.
- 주의 : 속도 향상을 위해 바이트 단위와 워드 단위를 변환하여 연산 할 때 올바로 이해하지 않으면 오류를 발생 시킬 수 있다.
- 엔디안은 크게 두 가지로 나뉨
  - **빅 엔디안**(Big-endian)
    - 보통 큰 단위가 앞에 나옴. 네트워크.
  - **리틀 엔디안**(Little-endian)
    - 작은 단위가 앞에 나옴. 대다수 데스크탑 컴퓨터.

종류	0x1234의 표현	0x12345678의 표현
빅 엔디안	12 34	12 34 56 78
리틀 엔디안	34 12	78 56 34 12

비트 연산 예제3

- 엔디안 확인 코드

```
main(void)
{
        int n = 0x00111111;
        char *c = (char *)&n;

        if(c[0])
                printf("little endian");
        else
                printf("big endian");
}
```

비트 연산 예제4

- 엔디안 변환 코드

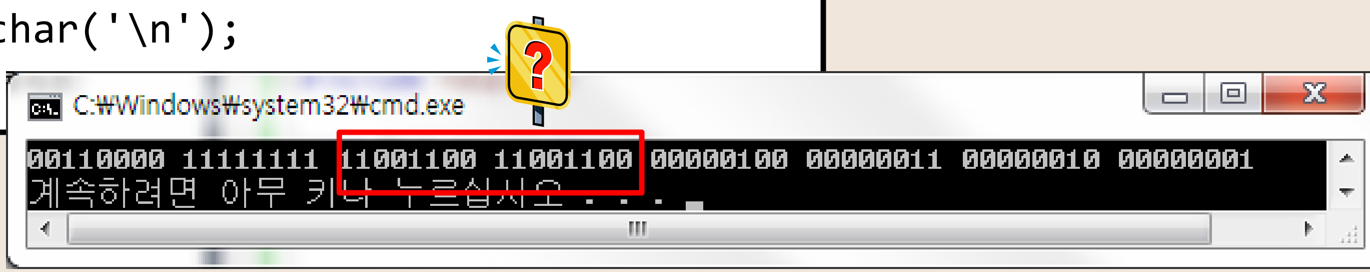
```
void ce(int * n) //change endian
{
        char *p = (char *) n;
        char t;
        t = p[0], p[0] = p[3], p[3] = t;
        t = p[1], p[1] = p[2], p[2] = t;
}
```

```
void ce1(int *n)
{   *n = (*n<<24) | ((*n<<8) & 0xff0000) | ((*n>>8) & 0xff00) | (*n>>24); }
```

```
main( )
{
        int x = 0x01020304;
        char *p = (char*) &x;

        printf("x = %d%d%d%d\n", p[0],p[1],p[2],p[3]);
        ce(&x);
        printf("x = %d%d%d%d\n", p[0],p[1],p[2],p[3]);
}
```

비트 연산 예제5

```
main(void)
{
        struct {
           char a;
           char b;
           int j;
        } s;
        char *p;    int i;

        s.a = 0x30;
        s.b = 0xff;
        s.j = 0x01020304;

        p = (char *) &s;

        for(i = 0; i < sizeof(s); i++)
                Bbit_print(*p++);
        putchar('\n');
}
```



Byte Alignment

- 32bit machine 에서 32bit bus line 을 활용하여 메모리를 access 한다. 프로세서의 성능 향상을 위하여 주소버스가 4의 배수형태의 주소만 access 한다.
- 어떤 객체(4byte)가 4의 배수형 주소에 있지 않다면 메모리 access를 2 번 해야 한다. 이에 따라 각 변수는 저장 될 수 있는 주소의 번지 패턴이 있다.
  - 1byte 형은 어떠한 주소 번지에라도 기록
  - 2byte 형은 2byte boundary 에 정렬
  - 4byte 형은 4byte boundary 에 정렬
  - Double (8byte) 형은 windows 에서는 8byte, 리눅스에서는 4byte boundary 가 된다.
  - 배열은 그 형에 따라 boundary가 결정

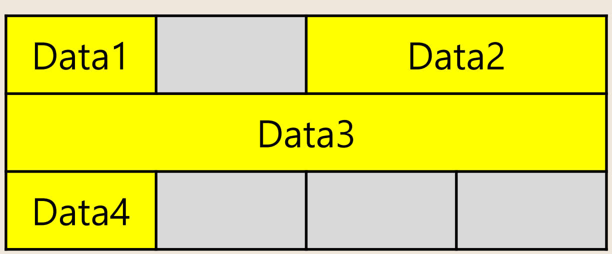
Structure Byte Padding

- 구조체의 멤버들이 byte alignment을 해야 하는 관계로 멤버들 사이에 임의의 공간이 생기는 현상(padding byte).
- 구조체의 경우 멤버 중 가장 큰 데이터 타입의 배수 값으로 크기가 결정

```
struct Message
{
        char Data1;      short Data2;
        int Data3;       char Data4;
}m;
void main(void)
{
        printf("%d \n", sizeof(struct Message));
}
```

결과 : 12

m



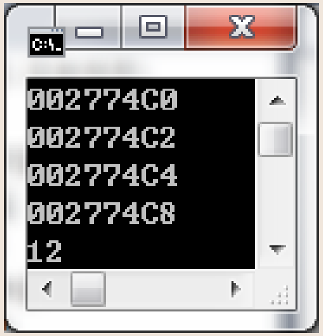
구조체의 padding과 크기 예

```
struct Message
{
        char Data1;
        short Data2;
        int Data3;
        char Data4;
}m;

void main(void)
{
        printf("%p\n", &m.Data1);
        printf("%p\n", &m.Data2);
        printf("%p\n", &m.Data3);
        printf("%p\n", &m.Data4);

        printf("%d \n",
                sizeof(struct Message));
}
```

```
/* after compilation */
struct MixedData
{
        char Data1;
        char Padding0[1]; // short는 2의 배수
        short Data2;
        int Data3;
        char Data4;
        char Padding1[3]; // 패딩
};
```

int 형이 가장 큰 데이터 형

int 배수 형태가 크기(12)로 결정



구조체의 padding 과 크기 예

```
struct Message
{
	char Data1;
	short Data2;
	int Data3;
	char Data4;
} m;

void main(void)
{
	printf("%d \n",
		sizeof(stru
}
```

결과 : 12

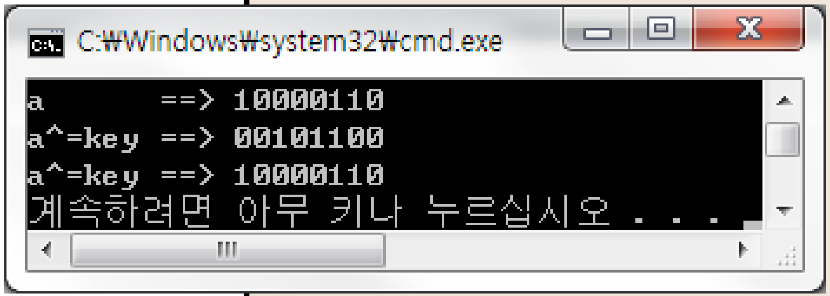
```
struct Message
{
	char Data1;
	char Data4;
	short Data2;
	int Data3;
} m;

void main(void)
{
	printf("%d \n",
		sizeof(struct Message));
}
```

결과 : 8

비트 연산 예제6

- 비트 연산자 ^를 두 번 연산하면 처음 값을 반환한다.

```
void main(void)
{
        char a = 0x86;
        char key = 0xAA;

        printf("a      ==> ");
        Bbit_print(a);   putchar('\n');

        printf("a^=key ==> ");
        a ^= key;
        Bbit_print(a);   putchar('\n');

        printf("a^=key ==> ");
        a ^= key;
        Bbit_print(a);   putchar('\n');
}
```



비트 연산 예제7

패킹과 언패킹 예

- 4 개의 문자를 하나의 int 형에 패킹하는 함수
- 32 비트 int 안에 있는 문자를 검색하는 함수 (마스크 사용)

```
int pack(char a, char b, char c, char d)
{
        int   p = a;
        p = (p << 8) | b;
        p = (p << 8) | c;
        p = (p << 8) | d;
        return p;
}
```

```
char unpack(int p, int k)
{
        int  n = k * 8;
        unsigned  mask = 255;
        mask <<= n;
        return ((p & mask) >> n);
}
```

진수

2진수, 8진수, 10진수, 16진수

10진수 → 타 진수로 변환

- 원하는 타진법의 수로 나눈 뒤 나머지를 거꾸로 읽는다.
- 예제) $(149)_{10} = (10010101)_2$

  $= (225)_8$

  $= (95)_{16}$

2	149	→ 1
2	74	→ 0
2	37	→ 1
2	18	→ 0
2	9	→ 1
2	4	→ 0
2	2	→ 0
	1	

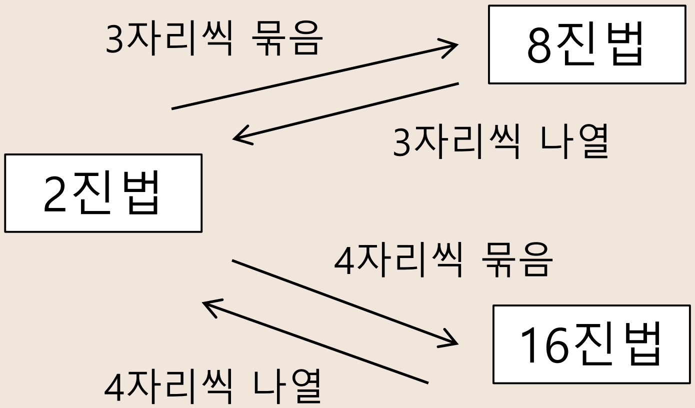
MSB(most significant bit)

타 진수 ➔ 10진수로 변환

- 예) $(135)_8 = 1*8^2 + 3*8^1 + 5*8^0 = 93_{10}$
- 소수점이 있을 때의 예)

  $(135.12)_8 = 1*8^2 + 3*8^1 + 5*8^0 + 1*8^{-1} + 2*8^{-2} = 93.15625_{10}$

2진수, 8진수, 16진수간 변환



컴퓨터에서의 음의 정수 표현 방법

- 1의 보수 : 부호와 절대값으로 표현된 값을 부호 비트를 제외 한 나머지 비트들을 0은 1로, 1은 0로 변환한다.
  - -6 : 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 : 부호와 절대값 표현.
  - -6 : 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 : 1의 보수 표현.
- 2의 보수 : 1의 보수방법으로 표현된 값의 최하위 비트에 1을 더한다.
  - -6 : 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 : 2의 보수 표현.

연습문제2

- 16진수 문자로 이루어진 1차 배열이 주어질 때 앞에서부터 7bit씩 묶어 십진수로 변환하여 출력해 보자
  - 예를 들어

0	F	9	7	A	3

  - 일 경우
    - 000011111001011110100011

      ➔ 0000111 1100101 1110100 011
    - 7, 101, 116, 3을 출력한다.

  - 입력 예
    - 01D06079861D79F99F

실수

실수의 표현

소수점 이하 4자리를 10진수로 나타내보면

2진수	10진수 값
0.0000	0
0.0001	0.0625
0.0010	0.125
0.0011	0.1875
0.0100	0.25
0.0101	0.3125
0.0110	0.375
0.0111	0.4375
0.1000	0.5
0.1001	0.5625
0.1010	0.625
0.1011	0.6875
0.1100	0.75
0.1101	0.8125
0.1110	0.875
0.1111	0.9375

- 2진 실수를 10진수로 변환하는 방법
  - 예) 1001.0011

$1 \times 2^{-4} = 1 \times 0.0625 = 0.0625$

$1 \times 2^{-3} = 1 \times 0.125 = 0.125$

$0 \times 2^{-2} = 0 \times 0.25 = 0.25$

$0 \times 2^{-1} = 0 \times 0.5 = 0.5$

$1 \times 2^{0} = 1 \times 1 = 1$

$0 \times 2^{1} = 0 \times 2 = 0$

$0 \times 2^{2} = 0 \times 4 = 0$

$1 \times 2^{3} = 1 \times 8 = 8$

9.1875

실수의 표현

- 컴퓨터는 실수를 표현하기 위해 부동 소수점(floating-point) 표기법을 사용한다
- 부동 소수점 표기 방법은 소수점의 위치를 고정시켜 표현하는 방식이다
  - 소수점의 위치를 왼쪽의 가장 유효한 숫자 다음으로 고정시키고 밑수의 지수승으로 표현

$$1001.0011 \rightarrow 1.0010011 \times 2^3$$

실수를 저장하기 위한 형식

- 단정도 실수(32비트)
- 배정도 실수(64비트)

단정도 실수

부호1비트	지수 8비트	가수 23비트

배정도 실수

부호1비트	지수 11비트	가수 52비트

- 가수부(mantissa) : 실수의 유효 자릿수들을 부호화된 고정 소수점으로 표현한 것
- 지수부(exponent) : 실제 소수점의 위치를 지수 승으로 표현한 것

단정도 실수의 가수 부분을 만드는 방법

- 예 : 1001.0011
  - 정수부의 첫 번째 자리가 1이 되도록 오른쪽으로 시프트
  - 소수점 이하를 23비트로 만든다
  - 소수점 이하만을 가수 부분에 저장
  - 지수 부분은 시프트 한 자릿수 만큼 증가 또는 감소

0001.0010011

0001.00100110000000000000000

00100110000000000000000

→ 1.0010011×2^{3}

단정도 실수의 지수 부분을 만드는 방법

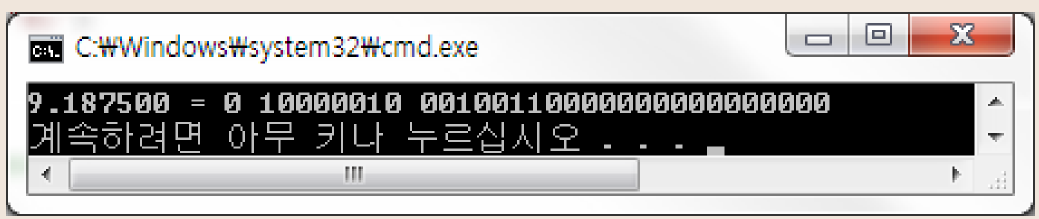
- 지수부에는 8비트가 배정(256개의 상태를 나타낼 수 있음)
- 숫자로는 0-255까지 나타낼 수 있지만, 음수 값을 나타낼 수 있어야 하므로 익세스(excess) 표현법을 사용
  - 익세스 표현법 : 지수부의 값을 반으로 나누어 그 값을 0으로 간주하여 음수지수와 양수지수를 표현하는 방법

단정도 표현에서의 지수부 익세스 표현

실제 지수	2진수	10진수 값
128	11111111	255
127	11111110	254
………………………………		
3	10000010	130
2	10000001	129
1	10000000	128
0	01111111	127
-1	01111110	126
………………………………		
-126	00000001	1
-127	00000000	0

✔ 예 : 1001.0011을 단정도 실수로 표현한 예

0	10000010	00100110000000000000000



- **컴퓨터는 실수를 근사적으로 표현한다.**
  - 이진법으로 표현 할 수 없는 형태의 실수는 정확한 값이 아니라 근사 값으로 저장되는데 이때 생기는 작은 오차가 계산 과정에서 다른 결과를 가져온다.
- **실수 자료형의 유효 자릿수를 알아 두자.**
  - 32 비트 실수형 유효자릿수(십진수) ➔ 6
  - 64 비트 실수형 유효자릿수(십진수) ➔ 15

연습문제3

- **16진수 문자로 이루어진 1차 배열이 주어질 때 암호비트패턴을 찾아 차례대로 출력하시오. 암호는 연속되어있다.**
  - 예를 들어

0	**D**	**E**	**C**

  - 일 경우
    - 00 001101 111011 00
    - 0, 2가 출력된다.
  - 입력 예
    - 0269FAC9A0

암호비트패턴

0	001101
1	010011
2	111011
3	110001
4	100011
5	110111
6	001011
7	111101
8	011001
9	101111

기본 확률 이론 정리

문제 제시 : 다음 물음의 답은 무엇인가?

- 다섯 개의 문자 a, a, a, b, b 를 모두 일렬로 배열하는 방법의 수는?
- 숫자 1, 2, 3 을 사용하여 만들 수 있는 네 자리 정수의 개수는? (단, 한 숫자는 여러 번 사용할 수도 있다.)
- 서로 다른 5통의 편지를 A, B, C 의 세 우체통에 넣는 방법의 수는?
- 남자 5명, 여자 3명 중에서 남자 3명, 여자 2명의 임원을 선출하는 방법의 수는?
- 6명의 선거인이 2명의 후보자에게 무기명으로 투표하는 방법의수는?

경우의 수

합의 법칙

- 두 사건 *A, B* 가 **동시에 일어나지 않을 때**, *A, B* 가 일어나는 경우를 각각 *m*가지, *n*가지라고 하면 ***A* 또는 *B*** 가 일어나는 경우의 수는 ***m+n*** 가지이다.

곱의 법칙

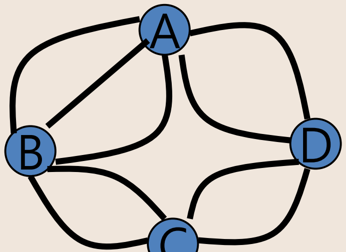
- 한 사건 A 가 m 가지로 일어나고 그 각각에 대하여 다른 사건 B 가 n가지로 일어날 때 **A 와 B 가 동시에 일어나는 경우의 수는** $m \times n$ 이다.

연습문제

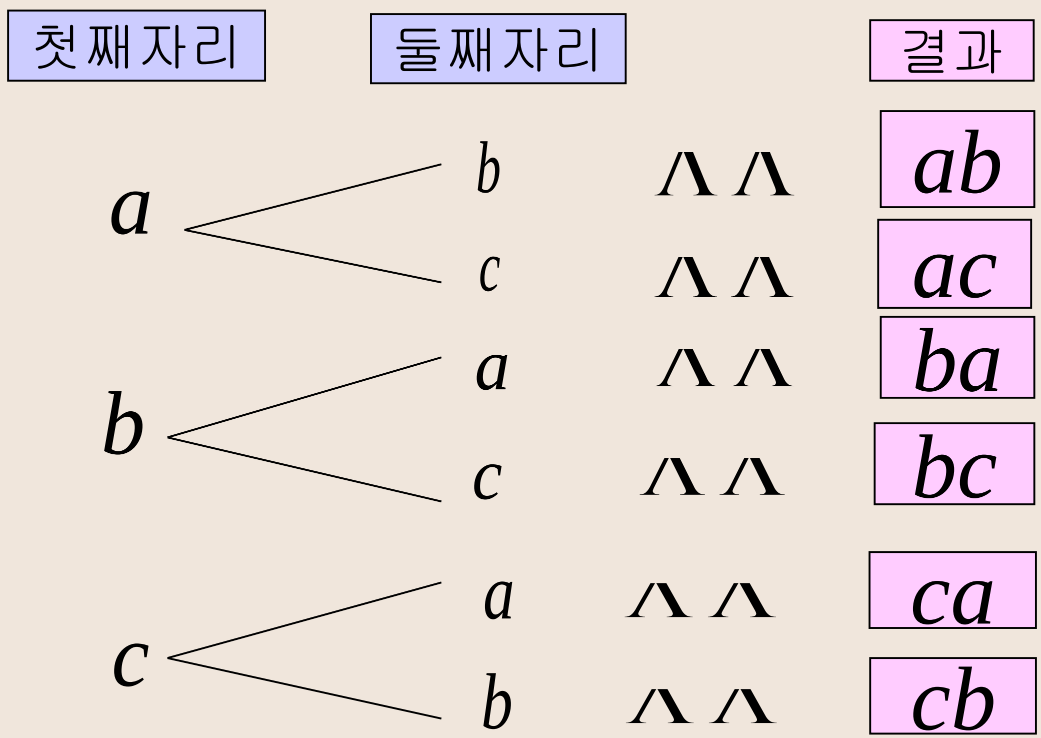
- A, B, C, D 의 네 지점을 잇는 도로망이 오른쪽 그림과 같을 때, 다음 물음에 답하여라.

(1) B에서 A를 지나 C로 가는 방법의 수

(2) B에서 C 로 가는 방법의 수를 구하라.



세 개의 문자 *a*, *b*, *c* 중 두 개를 택하여 일렬로 배열하는 방법의 수는?



- 따라서, 앞의 결과는 첫째 자리에 들어갈 수 있는 경우의 수에다 둘째 자리에 들어갈 수 있는 경우의 수를 곱의 법칙에 의하여 곱한 가지 수가 된다.

$$\therefore 3 \times 2 = 6\,(\text{가지})$$

- 이것을 서로 다른 3개에서 2개를 택하는 순열 이라 한다.

순 열

- 서로 다른 n 개에서 r 개를 택하여 일렬로 나열하는 방법을 ***n* 개에서 *r* 개를 택하는 순열**이라 하고, 이 순열의 수를 ${}_{n}P_{r}$ 로 나타낸다.

$$ {}_{n}P_{r} = n(n-1)(n-2)\Lambda\ (n-r+1) \quad (\text{단, } r \le n \ ) $$

- ${}_{n}P_{r}$ 에서 $r = n$ 이면,

$$ {}_{n}P_{n} = n\cdot(n-1)\cdot(n-2)\Lambda\ 2\cdot 1 = n! $$

$$ {}_{n}P_{r} = \frac{n!}{(n-r)!} \qquad 0! = 1 \qquad {}_{n}P_{0} = 1 $$

- 1, 2, 3, 4, 5 의 다섯 개의 숫자 중에서 서로 다른 세 숫자를 이용하여 만들 수 있는 세 자리의 자연수는 모두 몇 개인가?

순열의 점화식

$$ {}_{n}P_{r} = n \cdot {}_{n-1}P_{r-1} $$

- 1, 2, 3, 4 네 개의 숫자에서 세 개를 뽑는 순열을 생각해 보자.
- 먼저 4가 마지막에 있는 경우 (X, Y, 4)는 나머지 1, 2, 3에서 두 개의 위치를 채우면 된다.
- 그런데 4이외에 1, 2, 3가 맨 마지막에 있는 경우 (X, Y, 1), (X, Y, 2), (X, Y, 3)도 생각해야 하므로 결국 마지막에 있는 수를 제외한 나머지 세 개의 숫자에서 두 개의 순열을 뽑으면 된다.
- 이 경우의 수가 위의 점화식이다.

1, 2, 3, 4 네 개의 숫자에서 세 개를 뽑는 순열

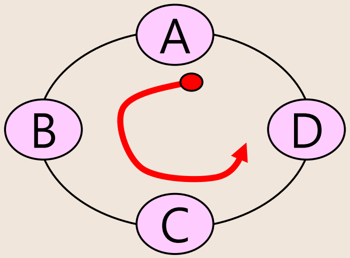
```
int t[10];
int a[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

void Perm(int n, int r, int q)
{
        if(r == 0) print(q);
        else {
                for(int i = n-1; i >= 0; i--) {
                        swap(&a[i], &a[n-1]);
                        t[r-1] = a[n-1];
                        Perm(n-1, r-1, q);
                        swap(&a[i], &a[n-1]);
                }
        }
}
void main(void)
{       Perm(4, 3, 3);     }
```

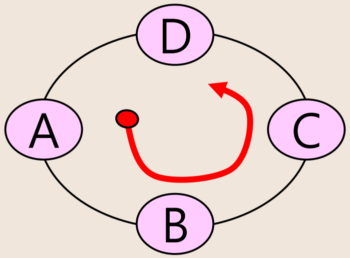
```
void print(int q)
{
        while(q) printf(" %d", t[--q]);
        printf("₩n")
}
```

원순열

- 서로 다른 n개의 원소를 **원형으로 배열하는** 것을 **원순열** 이라 한다.
- 또 이를 계산하는 방법은 $(n-1)!$

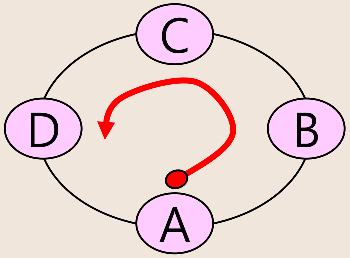
ABCD

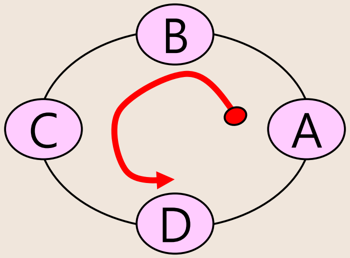




DABC

CDAB





BCDA

중복 순열

- 서로 다른 n 개의 **중복을 허용하여** r 개를 택하여 일렬로 나열하는 방법을 n 개에서 r 개를 택하는 **중복순열** 이라 한다.

$$ {}_{n}\Pi_{r} = n^{r} $$

- 서로 다른 3 개의 과일 사과, 배, 수박 이 있다. 2개를 택하여 일렬로 배열할 때 중복을 허용하여 나열한 순열의 개수는?
  - (사과, 사과), (배,배), (수박,수박) 의 3가지를 더 생각할 수 있다.
- 중복 순열의 점화식

$$ {}_{n}\Pi_{r} = n \cdot {}_{n}\Pi_{r-1} $$

중복 순열 구하기 코드 예

```
void PI(int n, int r, int q)
{
        if(r == 0) print(q);
        else {
                for(int i = n-1; i >= 0; i--) {
                        swap(&a[i], &a[n-1]);
                        t[r-1] = a[n-1];
                        PI(n, r-1, q);
                        swap(&a[i], &a[n-1]);
                }
        }
}
```

조합

- 서로 다른 n 개에서 **순서를 생각하지 않고** r 개 (0 <= r <= n)를 택하는 것을 **조합**이라 한다

$$ {}_nP_r = {}_nC_r \times r! $$

$$ {}_nC_r = \frac{{}_nP_r}{r!} = \frac{n!}{r!(n-r)!}, \quad {}_nC_0 = 1 $$

$$ {}_nC_r = {}_nC_{n-r} \qquad {}_nC_p \times {}_{n-p}C_q \times {}_rC_r $$

서로 다른 n 개의 물건을 p개, q개, r개 ($p+q+r = n$) 의 3개조로 나누는 방법의 수

- 서로 다른 종류의 꽃 15송이를 다섯 송이씩 세 묶음으로 나누는 방법의 수는?

다음 식을 생각해 보자

$$(a+b)^4 = (a+b)(a+b)(a+b)(a+b)$$
$$= a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

- 4개의 인수 (a+b) 로 부터 각각 a 또는 b 를 하나씩 택하여 곱한 것이다.
- 예를 들어, ab^3 항은 4개의 (a+b) 중에서 3개로부터 b를 택하고, 나머지 하나는 a를 택하여 곱한 것. 즉, ${}_4C_3 \times {}_1C_1 = 4$

이항정리

$$(a+b)^n = \boxed{{}_nC_0} a^n + \boxed{{}_nC_1} a^{n-1}b + \boxed{{}_nC_2} a^{n-2}b^2 +$$

$$\Lambda + \boxed{{}_nC_r} a^{n-r}b^r + \Lambda + \boxed{{}_nC_n} a^0 b^n$$

$$= \sum_{r=0}^{n} {}_nC_r a^{n-r} b^r$$

일반항

이항계수

파스칼의 삼각형

$(a+b)$ 1 1

$(a+b)^2$ 1 2 1

$(a+b)^3$ 1 3 3 1

$(a+b)^4$ 1 4 6 4 1

$(a+b)^5$ 1 5 10 10 5 1

Λ Λ Λ Λ Λ Λ Λ Λ Λ

조합의 점화식

$$ {}_{n}C_{r} = {}_{n-1}C_{r-1} + {}_{n-1}C_{r} \qquad {}_{n}C_{0} = 1 $$

```
void Comb(int n, int r, int q)
{
        if(r == 0) print(q);
        else if (n < r) return;
        else
        {
                t[r-1] = a[n-1];
                Comb(n-1, r-1, q);
                Comb(n-1, r, q);
        }
}
```

중복조합

- 서로 다른 n 개에서 **중복을 허락** 하여 r 개를 택하는 조합을 **중복조합** 이라 하고, 이 중복조합의 수를 ${}_{n}H_{r}$ 로 나타낸다

$$ {}_{n}H_{r} = {}_{n+r-1}C_{r} $$

- 중복조합의 점화식

$$ {}_{n}H_{r} = {}_{n}H_{r-1} + {}_{n-1}H_{r} $$

- 예 : 숫자 1, 2에서 중복을 허락하여 3개를 택하는 조합은?
  - (1,1,1), (1,1,2), (1,2,20, (2,2,2)

순열, 중복순열, 조합, 중복조합의 차이점

(i) ${}_nP_r$: 중복을 허락하지는 않지만, 순서는 생각한다.

(ii) ${}_n\Pi_r$: 중복을 허락하고 순서도 생각한다.

(iii) ${}_nC_r$: 중복을 허락하지 않고 순서도 생각하지 않는다.

(iv) ${}_nH_r$: 중복을 허락하고 순서는 생각하지 않는다.

연습 문제

- 다섯 개의 문자 *a, a, a, b, b* 를 모두 일렬로 배열하는 방법의 수는?
- 숫자 1, 2, 3 을 사용하여 만들 수 있는 네 자리 정수의 개수는? (단, 한 숫자는 여러 번 사용할 수도 있다.)
- 서로 다른 5통의 편지를 A, B, C 의 세 우체통에 넣는 방법의 수는?
- 남자 5명, 여자 3명 중에서 남자 3명, 여자 2명의 임원을 선출하는 방법의 수는?

확률에 관한 여러 용어 정리

- 시행 : 동일한 조건에서 여러 차례 반복할 수 있는 실험이나 관찰.
- 표본공간 : 어떤 시행에서 일어날 수 있는 모든 가능한 결과의 전체집합
- 사건 : 표본공간의 부분집합
- 전사건 : 표본공간 자신의 집합 (반드시 일어나는 사건)
- 공사건 : 결코 일어나지 않는 사건
- 합사건 : ***A* 또는 *B*** 가 일어날 사건
- 곱사건 : ***A* 와 *B*** 가 동시에 일어날 사건
- 배반사건 : 두 사건 $A \cap B = \phi$ 인 A 와 B 를 서로 배반사건이라 한다.
- 여사건 : A^c 사건 A 에 대하여 A가 일어나지 않는 사건을 A의 여사건이라 한다.

확률의 정의

- 하나의 사건이 일어날 수 있는 가능성을 수치로 나타낸 것 사건 A가 일어날 확률을 *P(A)* 로 나타낸다.

확률의 기본 성질

- 임의의 사건 A 에 대하여 $0 \le P(A) \le 1$ 이며,
- 특히, $P(U) = 1, P(\phi) = 0$ 이다.

여사건의 확률

- A의 여사건을 A^c , 사건 A가 일어날 확률을 P(A) 라고 하면

$$P(A^c) = 1 - P(A)$$

수학적 확률

- 어떤 시행에서 얻어지는 근원사건이 모두 같은 정도로 일어날 것이라고 기대될 때, 전사건 S에 속하는 총수를 n(S) , 사건 A 에 속하는 근원사건의 개수를 n(A) 라 하면, 사건 A가 일어날 확률 P(A) 는

$$P(A) = \frac{n(A)}{n(S)} = \frac{\text{사건}A\text{가 일어날 경우의 수}}{\text{모든 경우의 수}}$$

통계적 확률

- 한 사건 A 가 일어날 확률을 P 라 할 때, n 번의 반복시행에서 사건 A 가 일어난 횟수를 r 이라 하면, 상대돗수 $\frac{r}{n}$ 은 n 이 커짐에 따라 확률 P 에 가까워 진다.

$$\therefore \quad \lim_{n\to\infty} \frac{r}{n} = P \quad \Longleftrightarrow \quad P: \text{사건 } A \text{ 의 통계적 확률}$$

연습문제

- 20장의 복권 중에 당첨 복권이 4장 들어있다. 이 중에서 2장의 복권을 샀을 때 2장 모두 당첨될 확률은?
- 남자 6명, 여자 4명 중에서 위원 4명을 뽑을 때 남자 2명, 여자 2명이 뽑힐 확률은?
- 한 줄로 6명이 설 때, 특정한 3사람이 이웃하게 될 확률은?
- 흰 구슬 8개와 붉은 구슬 5개가 들어 있는 주머니에서 3 개를 꺼낼 때, 모두 같은 색일 확률은?
- 10개의 제비 중에서 당첨이 4개 들어있다. 처음에 갑이 1개를 꺼내고, 다음에 을이 한 개를 꺼낼 때 을이 당첨을 뽑을 확률은?

확률의 덧셈정리

- 두 사건 A, B 에 대하여

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

- 두 사건 A, B 가 **동시에 일어나지 않을 때**,

$$P(A \cup B) = P(A) + P(B)$$

- 한 개의 주사위를 던질 때 2의 배수 또는 3의 배수의 눈이 나올 확률은?
- 흰 공 4개, 검은 공 5개가 들어 있는 주머니에서 3개의 공을 꺼낼 때 3개 모두 같은 색의 공일 확률은?

조건부확률

- A 안에서 $A \cap B$ 가 일어날 확률 $\dfrac{n(A\cap B)}{n(A)}$ 는 A 가 일어났을 때의 B 의 조건부 확률이다.

$$P(B|A) = \frac{n(A\cap B)}{n(A)} = \frac{P(A\cap B)}{P(A)}$$

$$P(A) = \frac{n(A)}{n(S)} \qquad P(A\cap B) = \frac{n(A\cap B)}{n(S)}$$

- 주머니 속에 흰 공 4개, 검은 공 3개가 들어 있다 한 개씩 두 번 꺼 낼 때 두 개가 모두 흰 공일 확률은? 처음 꺼낸 공을 다시 주머니에 넣었을 때 두 개 모두 흰 공 일 확률은?

독립사건, 종속사건

- $P(B|A) = P(B|A^c) = P(B)$ 이면, 사건 A와 B는 서로 독립 이라고 하고 이 때, 두 사건을 독립사건 이라 한다. 또, 서로 독립이 아닌 두 사건 즉, $P(B|A) \neq P(B)$ 일 때, 두 사건 A, B를 종속사건 이라 한다.

- 한 개의 주사위와 한 개의 동전을 동시에 던질 때 주사위의 3의 눈과 동전의 앞면이 나올 확률은?
- 10개의 제비 중에서 3개의 당첨 제비가 들어 있다. 이 제비를 A, B 두 사람이 A, B 순으로 하나씩 뽑을 때, 각자가 당첨할 확률은?

확률의 곱셈정리

$$P(A \cap B) = P(A) \cdot P(B|A) = P(B) \cdot P(A|B)$$

독립사건의 곱셈정리

- 두 사건 A 와 B 가 서로 독립 이면

$$P(A \cap B) = P(A) \cdot P(B)$$

흰 구슬 5개, 붉은 구슬 2개가 들어 있는 주머니에서 구슬을 꺼낼 때, 처음에는 붉은 구슬, 두 번째에는 흰 구슬이 나올 확률은?

독립 시행

- 동전이나 주사위를 여러 번 던질 때와 같이 어떤 시행을 계속해서 되풀이할 때, 매번 일어나는 사건이 서로 독립인 경우 즉, 각 시행의 결과가 그 이전의 시행의 결과에 영향을 미치지 않는 시행

독립시행의 확률

- 어떤 시행에서 사건 A 가 일어날 확률이 p 이고 그 여사건 이 일어날 확률이 q (q=1-p)일 때, n 번의 독립시행에서 사건 A 가 r 번 일어날 확률은 $P_r = {}_nC_r\, p^r q^{n-r} \ (r = 0, 1, 2, \Lambda\ , n)$

- 5개의 동전을 동시에 던질 때, 이 중에서 2개의 동전만 앞면이 나올 확률을 구하면?

정수론

정수론

- 컴퓨터의 동작과 이산수학은 매우 밀접한 관계를 가진다. 이와 관련된 정수론에 관련된 문제들도 프로그램을 개발하다 보면 등장한다. 물론 자주 나오지는 않지만 가끔씩 꼭 필요한 일이 있기 때문에 학습해 두는 것도 의미가 있다.

- 정수론에서 많이 거론되는 소수, 모듈러 연산에 대해 학습한다.
- 최대공약수, 최소공배수

약수(Divisor)

- n 과 d 가 정수, $d \neq 0$.
- n=dq 를 만족하는 정수 q가 존재하면 d가 n을 나눈다(divides).
- q 는 몫(quotient), d 는 n의 약수(divisor) 또는 인수(factor).
- d 가 n 을 나누면 $d|n$ 으로, 그렇지 않으면 $d\nmid n$으로 표기한다.

<u>정리</u>

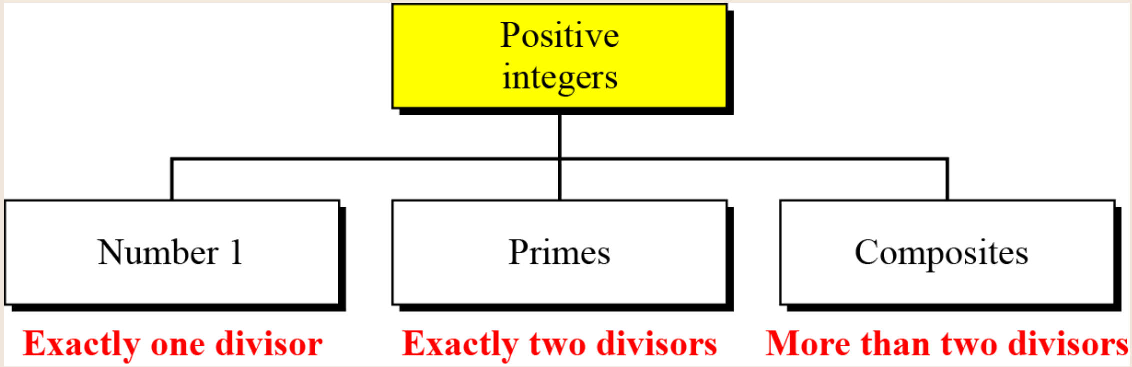
m, n, d 이 정수일 때

- $d|m$ 이고 $d|n$ 이면, $d|(m+n)$ 이다.
- $d|m$ 이고 $d|n$ 이면, $d|(m-n)$ 이다.
- $d|m$ 이고 $d|n$ 이면, $d|mn$ 이다.

소수

- 1보다 큰 어떤 정수가 1과 자신만을 양의 약수로 가진다면 이 정수는 소수
- 합성수
  - 소수가 아닌 1보다 큰 정수는 합성수
- 주의 : 가장 작은 소수는 무엇인가? 2 (1은 소수가 아니다)

양정수의 3 부류



소수 검사 알고리즘

```
is_prime1(n)
    FOR d in 2 → n-1
        IF n % d == 0 : RETURN FALSE
    RETURN TRUE
```

```
is_prime2(n)
    FOR d in 2 → ⌊√n⌋
        IF n % d == 0 : RETURN FALSE
    RETURN TRUE
```

```
is_prime2_1(n)
    IF n == 2 : RETURN TRUE
    IF n % 2 == 0 : RETURN FALSE
    FOR d in 3 → ⌊√n⌋ step 2
        IF n % d == 0 : RETURN FALSE
    RETURN TRUE
```

에라토스테네스의 체 (Sieve of Eratosthenes)

- 지워 지지 않은 수의 배수를 찾아 순회하며 지운다.

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

서로소(relatively prime number)

- 어떤 두 수가 공통적인 소인수를 갖지 못할 때 두 수를 **서로소** 라고 한다.
- 공통적인 소인수 => 최대 공약수 => GCD
- 양의 정수 c가 다음의 조건을 만족한다면 c는 a와 b의 최대 공약수
  - c는 a와 b의 약수
  - a와 b에 대한 어떠한 약수는 c의 약수
- GCD(a,b)= max[k, 이때 k는 k|a이고 k|b]

유클리드 알고리즘

- 문헌에 기록된 최초의 알고리즘
- 두 정수의 최대 공약수를 계산하는 효과적인 알고리즘
- gcd(a, b) = gcd(b, a mod b)
- 예
  - a = 105, b = 30
  - gcd(105, 30) = gcd(30,105 mod 30) = gcd(30, 15)
    = gcd(15, 30 mod 15) = gcd(15, 0)
  - gcd(15, 0) = 15
  - ∴ gcd(105,30) = 15

최소공배수 (LCM:Least Common Multiple)

- 공배수
  - m, n: 양의 정수
  - m 과 n 의 공배수는 m 과 n에 의해 나누어지는 정수
- LCM (최소 공배수)
  - *lcm(m,n)* : m 과 n의 최소 공배수

<u>정리1</u>

정수 m>1, n>1에 대한 각각의 소인수분해는

$$m = p_1^{a_1} p_2^{a_2} \ldots p_l^{a_l} \text{ 와 } \quad n = p_1^{b_1} p_2^{b_2} \ldots p_l^{b_l}$$

(p_i 가 $m(n)$의 소인수가 아니면, $a_i(b_i)=0$)

그러면 $lcm(m,n) = p_1^{\max(a_1,b_1)} p_2^{\max(a_2,b_2)} \ldots p_l^{\max(a_l,b_l)}$

Example

- $82320 = 2^4 \cdot 3^1 \cdot 5^1 \cdot 7^3 \cdot 11^0$
- $950796 = 2^2 \cdot 3^2 \cdot 5^0 \cdot 7^4 \cdot 11^1$
- $\mathrm{lcm}(82320, 950796) = 2^4 \cdot 3^2 \cdot 5^1 \cdot 7^4 \cdot 11^1 = 19015920$

<u>정리2</u>

양의 정수 m, n에 대해서,

$$gcd(m,n) \cdot lcm(m,n) = mn$$

Example

- gcd(30, 105) = 15
- lcm(30, 105) = 210
- gcd(30, 105) · lcm(30, 105) = 15 · 210 = 3150 = 30 · 105

정리 2를 사용하여

- $\mathrm{lcm}(m,n) = \dfrac{mn}{\mathrm{gcd}(m,n)}$

모듈러 연산

- 어떤 양의 정수 n과 어떤 정수 a가 주어지고, 만약 a를 n으로 나눈다면 다음과 같은 관계를 가지는 몫 q와 나머지 r을 얻는다

$$a = qn + r \qquad 0 <= r < n$$

$$q = \lfloor a/n \rfloor \qquad a \equiv r \bmod n$$

- 합동
  - (a mod n) = (b mod n), 두 정수 a와 b는 modulo n에 대해 합동
  - $a \equiv b \bmod n$ 으로 표기
  - $a \equiv 0 \bmod n$ 이라면 그때 $n|a$ 이다

모듈러 연산자의 특성

- 만약 $n|(a-b)$ 라면, $a \equiv b \bmod n$
- $(a \bmod n) = (b \bmod n)$은 $a \equiv b \bmod n$
- $a \equiv b \bmod n$은 $b \equiv a \bmod n$ 을 의미
- $a \equiv b \bmod n$과 $b \equiv c \bmod n$ 은 $a \equiv c \bmod n$ 을 의미

모듈러 산술 연산

- mod n 연산
  - 정수들의 범위 => {0, 1, ..., (n-1)}으로 표현가능
  - 즉, 이러한 집합의 범위 내에서 산술 연산이 가능

  - 모듈러 연산의 특징

1. [(a mod n)+(b mod n)] mod n = (a+b) mod n
2. [(a mod n)-(b mod n)] mod n = (a-b) mod n
3. [(a mod n)*(b mod n)] mod n = (a*b) mod n

예) a = 11, b = 15, n = 8

1. (a + b) mod n = ((11 mod 8) + (15 mod 8)) mod 8 = 3 + 7 mod 8

   = 10 mod 8 = 2

2. (a - b) mod n = ((11 mod 8) - (15 mod 8)) mod 8 = 3 - 7 mod 8

   = -4 mod 8 = 4

3. (a * b) mod n = ((11 mod 8) * (15 mod 8)) mod 8 = 3 * 7 mod 8

   = 21 mod 8 = 5

지수 연산 => 곱셈의 반복으로 수행가능

- 예) $11^7 \bmod 13$

$11^2 \bmod 13 = 121 \bmod 13 = (130 – 9) \bmod 13$

$= 0 + (- 9) \bmod 13 = 4 \bmod 13$

$11^4 \bmod 13 = 4^2 \bmod 13 = (13 \bmod 13) + (3 \bmod 13)$

$= 3 \bmod 13$

$11^7 \bmod 13 = (11 \bmod 13) (4 \bmod 13) (3 \bmod 13)$

$= (11 \bmod 13) (12 \bmod 13)$

$= 132 \bmod 13$

$= 2 \bmod 13$

N ^ A mod 1000000007 알고리즘은?

기계적 최적화

문제제시:어느 코드가 수행 시간이 빠른가?

```
for(i = 0; i < 1000; i++)
    i == 0;
```

```
for(i = 0; i < 1000; i++)
    i > 0;
```

홀짝의 판별을 빠르게 할 수 있는 방법은 없나?

```
if(i % 2 == 1)
    // 홀수 처리
else
    // 짝수 처리
```

정수

- unsigned 는 signed 연산 보다 빠르다
- unsigned int > int
- unsigned long > long long

floating point 연산은 매우 느리다.

- 만약 소수점 2자리까지의 정확도를 유지하는 프로그램을 만든다면, 모든 값에 x100을 해서 int 형으로 바꾼 다음 연산을 하도록 한다.

- **최적화란 프로그램을 좀 더 빨리, 좀 더 작게 개선하는 과정을 의미한다.**
- **일반적으로 최적화는 컴파일러의 소관이다. 좀 더 나은 컴파일러는 좀더 빠르고, 좀 더 작은 실행 코드를 만들어 내는 최적화 기능이 뛰어나기 때문이다.**
- **그러나 컴파일러가 수행하는 최적화는 어느 정도의 한계가 있다.**
  - 컴파일러의 최적화는 프로그래머의 의도를 어느 정도로 잘 해석하느냐는 것이 그 척도인데 비해서, 만약에 프로그래머의 의도가 잘못된 것이라면 컴파일러는 이에 대한 아무런 대책이 없기 때문이다.
- **그래서 소스코드 수준의 최적화를 위한 기법에 대해 알아본다.**

나눗셈을 피하자

- 분모와 분자의 32bit 나눗셈은 20~140의 실행 사이클을 가지고 있다.
- 나눗셈은 가능하면 곱셈으로 대체해서 사용하면 빠르다.
- 예를 들어 b * c가 integer 범위 안이라는 것을 안다면
  - (a/b) > c ➔ a > (c*b)로 다시 쓸 수 있다.

1 부터 n 까지 더하기

- 간단한 수학 공식은 기억해 두자.

```
for (i = 0; i < n; i++)
    sum += i;
```

```
n(n+1) * 0.5
```

짝수 홀수 확인

- 비트 연산을 이용하자. % 연산 보다 빠르다.

```
1234 & 1 ? printf("홀수\n"):printf("짝수\n");
// OR
1234 << 31 ?  printf("홀수\n"):printf("짝수\n");
```

2의 제곱로 나누기

- 나누기를 할 때 2의 제곱수를 분자로 함으로써, 코드를 더 효율적으로 만들 수 있다.

```
unsinged int div32u (unsinged int a) {
   return a / 32;
 }
```

```
unsigned int a = 1024;
unsigned b, c;
b = a/32;    // --- 1
c = a >> 5;  // --- 2
// 1과 2는 동일한 결과, 컴파일러에 의해 생성된 코드도 동일
```

Binary Breakdown

```
if(a==1) {
} else if(a==2) {
} else if(a==3) {
} else if(a==4) {
} else if(a==5) {
} else if(a==6) {
} else if(a==7) {
} else if(a==8) {
}
```

```
if(a<=4) {
    if(a==1)     {
    }  else if(a==2)  {
    }  else if(a==3)  {
    }  else if(a==4)  {
    }
}
else {
    if(a==5)  {
    } else if(a==6)  {
    } else if(a==7)  {
    } else if(a==8)  {
    }
}
```

배열을 이용한 index 생성

```
switch ( queue ) {
  case 0 :   letter = 'W';
     break;
  case 1 :   letter = 'S';
     break;
  case 2 :   letter = 'U';
     break;
}
```

```
if ( queue == 0 )
  letter = 'W';
else if ( queue == 1 )
  letter = 'S';
else
  letter = 'U';
```

```
static char *classes="WSU";
letter = classes[queue];
```

나머지 연산자의 대체

```
unsigned int modulo_func1 (unsigned int count)
{
   return (++count % 60);
}
```

```
unsigned int modulo_func1 (unsigned int count)
{
   if (++count >= 60)
      count = 0;
   return (count);
}
```

Using Aliases

```
void func1( int *data )
{
   int i;

   for(i=0; i<10; i++)
   {
      anyfunc( *data, i);
   }
}
```

```
void func1( int *data )
{
   int i;
   int localdata;

   localdata = *data;
   for(i=0; i<10; i++)
   {
     anyfunc ( localdata, i);
   }
}
```

*data가 결코 변하지 않는다고 하더라도, anyfunc 함수를 호출하는 컴파일러는 이걸 알 수가 없다. 그래서 변수가 사용될 때마다 메모리로부터 다시 읽어 들이게 된다. 이 문제는 지역변수를 하나 더 둠으로써 해결할 수 있다.

전역 변수

- 전역 변수는 절대 레지스터에 할당할 수 없다. 포인터를 사용하여 간접적으로 할당하거나 함수호출을 이용해서 전역변수를 변환할 수 있다.
- 따라서 컴파일러는 전역변수의 값을 레지스터에 올려서 캐쉬 할 수 없게 되고 때문에 전역변수를 이용할 때마다 다시 읽어 들이는 오버로드가 생기게 된다.
- 그러므로 가능하면 전역변수를 직접 호출하는 대신에, 지역변수를 이용해서 필요한 연산을 하고 그 결과를 전역변수에 할당하는 방법을 사용해야 한다.

```c
int f(void);
int g(void);
int h(void);
int errs;
void test1(void)
{
  errs += f();
  errs += g();
  errs += h();
}
void test2(void)
{
  int localerrs = errs;
  localerrs += f();
  localerrs += g();
  localerrs += h();
  errs = localerrs;
}
```

지역변수

- 가능하면 지역변수로 char 이나 short를 사용하지 않도록 한다.
- char와 short가 사용될 경우 컴파일러는 값을 저장하기 위해서 8bit 혹은 16bit를 할당한 후, 남는 크기를 줄이는 작업을 하게 된다.
- 이는 24bit, 16bit 만큼을 shift 시키는 연산을 하게 됨을 의미한다.
- 그러므로 입력되는 데이터가 8 혹은 16 비트라고 하더라도, 32bit로 연산을 하도록 함수를 만들 필요가 있다.

```
int wordinc (int a)//가장 빠름
{
   return a + 1;
}

short shortinc (short a)
{
    return a + 1;
}

char charinc (char a)
{
    return a + 1;
}
```

포인터

- 구조체를 그대로 넘길 경우 구조체의 모든 값이 스택에 올라가기 때문에 느리게 작동한다. 이런 경우 포인터를 쓰도록 하자.
- 포인터를 통해서 구조체를 넘길 때, 구조체의 멤버를 수정 할 일이 없다면 상수로 선언해서 넘기도록 하자.
- 아래 예를 살펴보면, 값이 사용될 때마다 다시 읽혀질 필요가 없어지게 된다. 또한 이러한 코드는 실수로 구조체 멤버의 변수를 바꾸는 것과 같은 실수를 하지 않도록 해준다.

```
void print_data_of_a_structure ( const Thestruct  *data_pointer)
{
   ...printf contents of the structure...
}
```

Pointer chains

```
typedef struct { int x, y, z; } Point3;
typedef struct { Point3 *pos, *direction; } Object;

void InitPos1(Object *p)
{
   p->pos->x = 0;
   p->pos->y = 0;
   p->pos->z = 0;
}
```

```
void InitPos2(Object *p)
{
   Point3 *pos = p->pos;
   pos->x = 0;
   pos->y = 0;
   pos->z = 0;
}
```

- p->pos 가 캐쉬되므로 좀더 효율적으로 작동하게 된다.

Switch 대신 lookup table 를 사용하자

```
char * Condition_String1(int condition) {
  switch(condition) {
     case 0: return "EQ";
     case 1: return "NE";
     case 2: return "CS";
     case 3: return "CC";
     case 4: return "MI";
     case 5: return "PL";
     case 6: return "VS";
     case 7: return "VC";
     case 8: return "HI";
     case 9: return "LS";
     case 10: return "GE";
     case 11: return "LT";
     case 12: return "GT";
     case 13: return "LE";
     case 14: return "";
     default: return 0;
  }
}
```

```
char * Condition_String2(int condition) {
   if ((unsigned) condition >= 15) return 0;
      return
      "EQ\0NE\0CS\0CC\0MI\0PL\0VS\0VC\0HI\0LS\0GE\0LT\0GT\0LE\0\0" +
       3 * condition;
}
```

Loop termination

- 루프를 종료시키기 위한 검사는 항상 count-down-to-zero 방식을 사용하도록 한다. 이것은 좀더 적은 시간을 소비한다.

```
int i, fact = 1;
for (i = 1; i <= n; i++)
   fact *= i;
return (fact);
```

```
int i, fact = 1;
for (i = n; i != 0; i--)
   fact *= i;
return (fact);
```

더욱 빠른 for 문

```
for (i = 0; i < 10; i++) {...}
```

```
for (i = 10; i--;) {...}
```

```
for (i = 10; i ; i--) {...}
// OR
for (i = 10; i!=0; i--) {...}
```

함수 루프

- 함수는 호출되기 위한 분명한 오버헤드가 존재한다. 루프에서 함수를 호출하는 등의 코드는 작성하지 않는 게 좋다. 이런 류의 코드는 반대로 함수에서 루프를 수행하도록 변경하는걸 추천한다.

```
for(i=0 ; i<100 ; i++)
{
    func(t,i);
}
-
-
-
void func(int w,d)
{
    lots of stuff.
}
```

```
func(t);
-
-
-
void func(w)
{
    for(i=0 ; i<100 ; i++)
    {
        //lots of stuff.
    }
}
```

Population count – 비트 계수하기

- 주어진 값에 1bit가 몇 개인지를 검사하는 코드
- 4만큼 쉬프트 하는 식으로 바꿔서, 성능을 높일 수 있다.

```
int countbit1(int n)
{
    int bits = 0;
    while (n != 0)
    {
        if (n & 1) bits++;
        n >>= 1;
    }
    return bits;
}
```

```
int countbit2(int n)
{
    int bits = 0;
    while (n != 0)
    {
        if (n & 1) bits++;
        if (n & 2) bits++;
        if (n & 4) bits++;
        if (n & 8) bits++;
        n >>= 4;
    }
    return bits;
}
```

Loop 사용하지 않기

- 몇 번만 순환하는 루프의 경우 풀어 쓰면 성능을 향상시킬 수 있다
- 루프를 사용하지 않게 되면, 카운터를 유지하고 업데이트하고 비교하는 작업이 그만큼 줄어들게 된다.

```
for(i=0; i<3; i++)
{
    something(i);
}
```

```
something(0);
something(1);
something(2);
```

example

```
int a, b, c;
a = b / c;
a = b / 10;
a = (unsigned int)b / 10;
a = b / 16;
a = (unsigned int)b / 16;

int a, b, c;
a = b % c;
a = b % 10;
a = (unsigned int)b % 10;
a = b % 16;
a = (unsigned int)b % 16;
```

- 최적화를 위한 방법의 제일 첫째는 좋은 알고리즘을 선택하는 것이다.
- 최적화의 단계는 항상 프로그램 작성 시 최후의 단계여야 한다.