```
/*
실패 요인
내가 구현한 함수에서  index out of range error


비트 배열 첨자 인덱스 계산
        int idIdx1 = (int)( (id-1) / 32);
        int idIdx2 = (int)(id % 32);
        if( idIdx2 == 0)
                idIdx2 = 32;

*/


#include <stdio.h>
enum REPEAT_TYPE {
        NONE,
        DAILY,
        WEEKLY,
        MONTHLY,
        YEARLY,
        EVERY3DAYS,
        EVERY5DAYS,
        EVERY10DAYS,
        TYPE_MAX
};

struct DATE {
        int year;
        int month;
        int day;
};

struct REPEAT {
        REPEAT_TYPE type;
        int cnt;
};

extern int daysOfMonth(int y, int m);

/* My info ------------------------------------------------------------------*/
int daysPerYYMM[2200][14];

struct SCINFO {
        DATE date;
        REPEAT rp;
};
SCINFO scheduleInfo[11000];


struct IDS
{
        unsigned int ids[320];   //320*32 = 10240
};


IDS dayScheduleCount[37001];


int bit_mask[33];


/* function definition------------------------------------------------- */
inline void setIDToDaySchedule(int dayIndex, int id);
inline void remmoveIDToDaySchedule(int dayIndex, int id);
inline int getScheduleCount(int dayIndex);

int searchSchedule(DATE from, DATE to);
```

```
inline int bitCount(unsigned int i);
int getIndexByYYMMDD(int yy, int mm,int dd);


int firstCheck = 0;
int prevmaxids=0;

void init()
{
        register int i,j;
        int sum=0;
        //make table
        if( firstCheck ==0)
        {
                for(i=2000 ; i<2100 ; i++)
                {
                        for(j=1 ; j<=12 ; j++)
                        {
                                daysPerYYMM[i][j]=sum;
                                //printf(" %d,",daysOfMonth(i,j) );
                                //printf(" [%d][%d] : [%d], ",i,j,daysPerYYMM[i][j] );
                                sum +=daysOfMonth(i,j);
                        }
                        //printf("Wn");
                }

                for(i=1 ; i<=32 ; i++)
                {
                        bit_mask[i] = (1<<(i-1));
                }
                firstCheck =1;
        }


        for(i=0 ; i<=10000 ; i++)
        {
                scheduleInfo[i].date.day=0;
                scheduleInfo[i].date.month=0;
                scheduleInfo[i].date.year=0;
                scheduleInfo[i].rp.cnt=0;
                scheduleInfo[i].rp.type=(REPEAT_TYPE)0;
        }

#if 1
        for(i=0 ; i<=37000 ; i++)
        {
                for(j=0 ; j<320 ; j++)
                {
                        dayScheduleCount[i].ids[j]=0;
                }
        }
#endif

}


/*
일정을 추가한다.
id: 1~10,000 값을 가지며 중복이 없다.
date: 일정 시작일. DATE 구조를 가지며 구조는 다음과 같다.
struct DATE {
  int year; // 2000~2099
  int month; // 1~12
  int day; // 1~31 월별로 다름.
};
repeat: REPEAT 구조를 가지며 구조는 다음과 같다.
struct REPEAT {
  REPEAT_TYPE type; // 반복 형태
```

```
  int cnt; // 1~100 반복 횟수
};
type은 반복 형태이며 다음 값들 중 하나를 가진다.
enum REPEAT_TYPE {
  NONE,
  DAILY, // 1매일
  WEEKLY, // 2매주 시작일과 같은 요일
  MONTHLY, // 3매월 동일한 날짜
  YEARLY, // 4매년 동월 동일
  EVERY3DAYS, // 5시작일부터 3일마다
  EVERY5DAYS, // 6시작일부터 5일마다
  EVERY10DAYS, // 7시작일부터 10일마다
  TYPE_MAX
};
type이 NONE인 경우 cnt는 반드시 1이다.
일정이 반복되는 경우 2099년 12월 31일을 이후까지 반복되는 일정은 주어지지 않는다.
type이 MONTHLY인 경우 day는 28 이하 값을 가진다.
type이 YEARLY인 경우 month가 2라면 day값은 29를 가지는 경우는 없다.

*/


inline int bitCount(unsigned int i)
{
    i = i - ((i >> 1) & 0x55555555);
    i = (i & 0x33333333) + ((i >> 2) & 0x33333333);
    i = (i + (i >> 4)) & 0x0f0f0f0f;
    i = i + (i >> 8);
    i = i + (i >> 16);
    return i & 0x3f;
}


void addSchedule(int id, DATE date, REPEAT repeat)
{
        scheduleInfo[id].date = date;
        scheduleInfo[id].rp = repeat;

        int yy=scheduleInfo[id].date.year;
        int mm=scheduleInfo[id].date.month;
        int dd=scheduleInfo[id].date.day;

        int startIndex = getIndexByYYMMDD(yy,mm,dd);
        REPEAT_TYPE type = scheduleInfo[id].rp.type;
        int cnt=scheduleInfo[id].rp.cnt;

        register int i=0,j=0;
        if( type == NONE)
        {
                setIDToDaySchedule(startIndex,id);
        }
        else if( type == DAILY)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+1)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(i,id);
                        //printf("[addSchedule]  i:%d  %d Wn",i,dayScheduleCount[i]);
                        c_count++;
                }
        }
        else if( type == WEEKLY)
        {
                int c_count=0;
```

```
                for(i=startIndex ; ; i=i+7)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(i,id);
                        //printf("[addSchedule]  i:%d  %d ₩n",i,dayScheduleCount[i]);
                        c_count++;
                }
        }
        else if( type == MONTHLY)
        {
                int c_count=0;
                int n_yy=scheduleInfo[id].date.year;
                int n_mm=scheduleInfo[id].date.month;
                int n_dd=scheduleInfo[id].date.day;
                int n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
                while(1)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(n_startIndex,id);
                    //printf("₩t[Debug]  n_startIndex:%d  id:%d
%d;%d;%d₩n",n_startIndex,id,n_yy,n_mm,n_dd);

                        c_count++;
                        n_mm++;
                        if( n_mm>12)
                        {
                                n_yy++;
                                n_mm=1;
                        }
                        n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
                }
        }
        else if( type == YEARLY)
        {
                int c_count=0;
                int n_yy=scheduleInfo[id].date.year;
                int n_mm=scheduleInfo[id].date.month;
                int n_dd=scheduleInfo[id].date.day;
                int n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
                while(1)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(n_startIndex,id);
                        c_count++;

//printf("[addSchedule]  n_startIndex:%d  %d ₩n",n_startIndex,dayScheduleCount[n_startIndex]);

                        n_yy++;
                        if( n_yy>2099)
                        {
                                break;
                        }
                        n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
                }
        }
        else if( type == EVERY3DAYS)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+3)
```

```
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(i,id);
                                //printf("[addSchedule]  i:%d  %d Wn",i,dayScheduleCount[i]);
                        c_count++;
                }
        }
        else if( type == EVERY5DAYS)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+5)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(i,id);
                    //printf("[addSchedule]  i:%d  %d Wn",i,dayScheduleCount[i]);
                        c_count++;
                }
        }
        else if( type == EVERY10DAYS)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+10)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        setIDToDaySchedule(i,id);
                //printf("[addSchedule]  i:%d  %d Wn",i,dayScheduleCount[i]);
                        c_count++;
                }
        }
}

/*
void deleteScheduleByID(int id);
id로 반복되는 모든 일정을 삭제한다.
id: 삭제될 일정의 id
*/
void deleteScheduleByID(int id)
{
        int yy=scheduleInfo[id].date.year;
        int mm=scheduleInfo[id].date.month;
        int dd=scheduleInfo[id].date.day;

        int startIndex = getIndexByYYMMDD(yy,mm,dd);
        REPEAT_TYPE type = scheduleInfo[id].rp.type;
        int cnt=scheduleInfo[id].rp.cnt;

        //printf("[deleteScheduleByID] [yy][mm][dd] %d %d %d [startIndex] %d [type] %d [cnt]
%dWn",yy,mm,dd,startIndex,type,cnt);
        register int i=0,j=0;
        if( type == NONE)
        {
                remmoveIDToDaySchedule(startIndex,id);
        }
        else if( type == DAILY)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+1)
                {
                        if( c_count >= cnt)
```

```
                {
                        break;
                }
                remmoveIDToDaySchedule(i,id);
                //printf("[addSchedule]  i:%d  %d ₩n",i,dayScheduleCount[i]);
                c_count++;
        }
}
else if( type == WEEKLY)
{
        int c_count=0;
        for(i=startIndex ; ; i=i+7)
        {
                if( c_count >= cnt)
                {
                        break;
                }
                remmoveIDToDaySchedule(i,id);
                c_count++;
        }
}
else if( type == MONTHLY)
{
        int c_count=0;
        int n_yy=scheduleInfo[id].date.year;
        int n_mm=scheduleInfo[id].date.month;
        int n_dd=scheduleInfo[id].date.day;
        int n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
        while(1)
        {
                if( c_count >= cnt)
                {
                        break;
                }
                remmoveIDToDaySchedule(n_startIndex,id);
                c_count++;
                n_mm++;
                if( n_mm>12)
                {
                        n_yy++;
                        n_mm=1;
                }
                n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
        }
}
else if( type == YEARLY)
{
        int c_count=0;
        int n_yy=scheduleInfo[id].date.year;
        int n_mm=scheduleInfo[id].date.month;
        int n_dd=scheduleInfo[id].date.day;
        int n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
        while(1)
        {
                if( c_count >= cnt)
                {
                        break;
                }
                remmoveIDToDaySchedule(n_startIndex,id);
                c_count++;

                n_yy++;
                if( n_yy>2099)
                {
                        break;
                }
                n_startIndex = getIndexByYYMMDD(n_yy,n_mm,n_dd);
        }
```

```
        }
        else if( type == EVERY3DAYS)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+3)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        remmoveIDToDaySchedule(i,id);
                        c_count++;
                }
        }
        else if( type == EVERY5DAYS)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+5)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        remmoveIDToDaySchedule(i,id);
                        c_count++;
                }
        }
        else if( type == EVERY10DAYS)
        {
                int c_count=0;
                for(i=startIndex ; ; i=i+10)
                {
                        if( c_count >= cnt)
                        {
                                break;
                        }
                        remmoveIDToDaySchedule(i,id);
                        c_count++;
                }
        }
}


/*
void deleteScheduleByDate(DATE date);
해당일의 일정을 모두 삭제한다.
date: DATE 형태를 가진다.
*/
void deleteScheduleByDate(DATE date)
{
        int dateIndex = getIndexByYYMMDD(date.year,date.month,date.day);
        register int j;
        for(j=0 ; j<320 ; j++)
        {
                dayScheduleCount[dateIndex].ids[j]=0;
        }

        //printf("[deleteScheduleByDate] [yy][mm][dd] %d %d %d [startIndex] %d
₩n",date.year,date.month,date.day,dateIndex);
}


/*
일정 개수를 반환한다.
from: 시작일자
to: 끝일자
시작일자와 끝일자를 포함한 사이에 있는 모든 일정들의 수를 계산한다.
from에서 to까지 최대 일수는 30일이다.
```

```
*/
int searchSchedule(DATE from, DATE to)
{
        int startIndex = getIndexByYYMMDD(from.year,from.month,from.day);
        int endIndex = getIndexByYYMMDD(to.year,to.month,to.day);

        register int i,sum=0;
        for(i=startIndex ; i<=endIndex ; i++)
        {
                sum = sum+ getScheduleCount(i);
        }

        //printf("[searchSchedule] [%d/%d/%d]=%d ~ [%d/%d/%d]=%d  / sum [%d]
₩n",from.year,from.month,from.day,startIndex,
          //to.year,to.month,to.day,endIndex,sum);

        return sum;
}


int getIndexByYYMMDD(int yy, int mm,int dd)
{
        return (daysPerYYMM[yy][mm] + dd);
}

inline void setIDToDaySchedule(int dayIndex, int id)
{
        int idIdx1 = (int)( (id-1) / 32);
        int idIdx2 = (int)(id % 32);
        if( idIdx2 == 0)
                idIdx2 = 32;

        dayScheduleCount[dayIndex].ids[idIdx1] = dayScheduleCount[dayIndex].ids[idIdx1] |
bit_mask[idIdx2];
}
inline void remmoveIDToDaySchedule(int dayIndex, int id)
{
        int idIdx1 = (int)( (id-1) / 32);
        int idIdx2 = (int)(id % 32);
        if( idIdx2 == 0)
                idIdx2 = 32;

        dayScheduleCount[dayIndex].ids[idIdx1] = dayScheduleCount[dayIndex].ids[idIdx1] & (~
(bit_mask[idIdx2] ) );
}

inline int getScheduleCount(int dayIndex)
{
        register int i=0,sum=0;
        for(i=0 ; i<320 ; i++)
        {
                sum = sum+ bitCount(dayScheduleCount[dayIndex].ids[i]);
        }

        return sum;
}
```