```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "generate_function.h"

#define TRUE 1
#define FALSE 0
#define TABLESIZE 128 //hashtable 크기
#define BUFFSIZE 1000

int NUMBEROFLEVEL; //logic의 최대 level
double start, logic_start, logic, end; //시간 측정 변수

struct NET* headnet = NULL; //net의 제일 앞 부분
struct NETTABLE* table = NULL; //hashtable
struct QUEUE* primary_queue = NULL; //primary input이 들어있는 queue

struct NET* Search_net (char netname[BUFNAME], int location); //hash table에서 netname을 가진 net 검색
void Init_Table (void); //net table 초기화
void Build_Circuit (char* filename); //회로 생성
int Read_a_word (char* str, FILE* fp); //파일에서 단어 단위로 읽음

void Level_Decision (void); //net의 레벨 결정
int Logic_Calculation (char* in_vec); //레벨별 상태 및 스위칭 횟수 연산

int Logic_Value (struct NET* searchnet); //입력값에 의한 레벨 상태 출력
void Print_Net (int total_lv); //연산 결과 출력
void Insert_table (int hashcalc, struct NET* newnet, char* input_string); //hash table 확장
struct NET* net_stack (void); //스택형식으로 net을 새로 생성하는 함수
struct NETLIST* link_netlist (struct NET* newnet); //net의 출력을 netlist와 연결하는 부분

int main (void)
{
    char filename[BUFNAME];
    char input_vector[BUFNAME];
    int total_level;
    int * a;
    int *head_a;
    int temp;

    printf ("Input file name ? ");
    scanf ("%s", filename);

    printf ("Input vector file name ? ");
    scanf ("%s", input_vector);

    start = clock(); //시간을 측정 시작
    Init_Table ();
    Build_Circuit (filename);

    logic_start = clock();
    Level_Decision ();

    total_level = Logic_Calculation (input_vector);
```

```
    logic_start = clock();
    Level_Decision ();

    total_level = Logic_Calculation (input_vector);
    logic = clock();

    Print_Net (total_level);

    return 0;
}

//net table 초기화
void Init_Table (void)
{
    int i;

    table = (struct NETTABLE*) malloc (TABLESIZE * sizeof(struct NETTABLE));

    for (i = 0; i < TABLESIZE; i++){
        (table + i)->net = NULL;
        (table + i)->next = NULL;
    }
}

//파일로부터 circuit을 생성
void Build_Circuit (char* filename)
{
    FILE* fp;
    char in;
    int hashcalc, i;
    char instring[BUFNAME];
    struct NET* newnet;
    struct NET* newnet2;
    struct NETLIST* newlist;
    struct NETLIST* newlist2;
    struct QUEUE* newqueue;

    //파일 열기
    if ((fp = fopen (filename, "r")) == NULL)
    {
        printf ("ERROR : file can't open!!!₩n");
        exit (0);
    }

    //한 단어씩 읽으면서 처리
    while(Read_a_word (instring, fp) == TRUE)
    {
        //# 또는 OUTPUT이라는 문자를 만나면 해당 라인 skip
        if((strcmp (instring, "#") == 0) || (strcmp (instring, "OUTPUT") == 0)) {
            while((in = getc (fp)) != EOF && in != '₩n') {
                // 그 문장이 끝날 때까지 한개씩 문자를 받음
                if(in == '₩n') {
                    ungetc (in, fp);
                    break;
                }
```

```
            if(in == '₩n') {
                ungetc (in, fp);
                break;
            }
        }
    }

    //INPUT 이라는 문자를 만나면 입력이 되는 netline의 번호를 받음
    else if(strcmp(instring, "INPUT") == 0)
    {
        Read_a_word (instring, fp); //input net 읽기

        hashcalc = 0; //hash table 위치값은 0으로 초기화
        for(i = 0; instring[i] != '₩0'; i ++){
            hashcalc = hashcalc + atoi(&instring[i]); //입력 숫자를 받아서 전부 더함
        }

        hashcalc = hashcalc % TABLESIZE; //TABLESIZE로 나눠서 남는 나머지(테이블상의 위치를 나타냄)

        newnet = net_stack(); //net을 새로 생성

        //input을 나타내기 위한 primary_queue를 등록
        newqueue = Getqueue(); //queue 생성
        newqueue->link = primary_queue;
        primary_queue = newqueue; //queue끼리 계속 연결
        newqueue->net = newnet; //input을 나타내는 net과 queue를 연결

        Insert_table (hashcalc, newnet, instring); //hashtable에 net 등록
    }

    // 입력받은 값이 0~9 이거나 대문자 A~Z 이면
    else if ((instring[0] >= '0' && instring[0] <= '9') || (instring[0] >= 'A' && instring[0] <= 'Z'))
    {
        hashcalc = 0;

        for(i = 0; instring[i] != '₩0'; i ++) {
            hashcalc = hashcalc + atoi (&instring[i]);
        }

        hashcalc = hashcalc % TABLESIZE;

        newnet = Search_net(instring, hashcalc); //새로 생성할 net이 중복인지 검색

        //중복되지 않으면 새로 생성
        if(newnet == NULL) {
            newnet = net_stack ();
            Insert_table (hashcalc, newnet, instring);
        }

        //net의 type 지정
        Read_a_word (instring, fp);
        if (strcmp (instring, "AND") == 0) newnet->type = AND;
        else if (strcmp (instring, "OR") ==0) newnet->type = OR;
        else if (strcmp (instring, "NOT") == 0) newnet->type = NOT;
        else if (strcmp (instring, "NAND") == 0) newnet->type = NAND;
```

```
        if (strcmp (instring, "AND") == 0) newnet->type = AND;
        else if (strcmp (instring, "OR") ==0) newnet->type = OR;
        else if (strcmp (instring, "NOT") == 0) newnet->type = NOT;
        else if (strcmp (instring, "NAND") == 0) newnet->type = NAND;
        else if (strcmp (instring, "NOR") == 0) newnet->type = NOR;
        else if (strcmp (instring, "XOR") == 0) newnet->type = XOR;
        else if (strcmp (instring, "XNOR") == 0) newnet->type = XNOR;
        else if (strcmp (instring, "BUFF") == 0) newnet->type = BUFF;
        else newnet->type = UNDEFINED;

        //해당 net의 입력 부분 연결
        while(TRUE)
        {
            Read_a_word (instring, fp);
            hashcalc = 0;

            for (i = 0; instring[i] != '₩0'; i ++)
            {
                hashcalc = hashcalc + atoi (&instring[i]);
            }
            hashcalc = hashcalc % TABLESIZE;

            newlist = Getlist (); //net list 생성
            newlist->next = newnet->prev; //새로 생성한 netlist를 net의 입력 부분(netlist)과 연결
            newnet->prev = newlist; //새로 생성한 netlist를 net의 입력 부분으로 선언
            newlist->net = Search_net(instring, hashcalc); //새로 생성한 netlist를 해당 net과 연결

            //해당 net이 없을 경우 새로 생성
            if(newlist->net == NULL)
            {
                newnet2 = net_stack ();
                newlist2 = link_netlist(newnet2); //생성된 net에 netlist 연결
                newlist2->net = newnet2;
                Insert_table (hashcalc, newnet2, instring);
                newlist->net = newnet2;
            }
            //해당 net이 있을 경우 바로 연결
            else
            {
                newnet2 = newlist->net;
                newlist2 = link_netlist(newnet2);
                newlist2->net = newnet;
            }

            fseek (fp, -1L, SEEK_CUR);
            if ((in = getc (fp)) == ')') break;
        }
    }

    else
    {
        continue;
    }
}
```

```
        else
        {
            continue;
        }
    }

    fclose(fp);
}

void Print_Net(int total_lv)
{
    FILE *fp2;
    struct NET *temp=NULL;
    struct NETLIST *tempnet=NULL;
    int *logic_value=NULL;
    int i, end;

    fp2 = fopen("output.txt", "w");
    logic_value = (int *)malloc(total_lv+sizeof(int));
    for(i=0;i<=total_lv;i++)
        *(logic_value+i) = 0;

    for(temp=headnet;temp!=NULL;temp=temp->next) {
        fprintf(fp2, "₩nlogic name : %s₩n", temp->name);
        fprintf(fp2, "logic type : %d₩n", temp->type);
        fprintf(fp2, "level : %d₩n", temp->level);
        fprintf(fp2, "sw : %d₩n", temp->sw);
        fprintf(fp2, "swcnt : %d₩n", temp->swcnt);
        tempnet = temp->prev;
        while(tempnet!=NULL) {
            fprintf(fp2, "prev net : %s₩n", tempnet->net->name);
            tempnet=tempnet->next;
        }
        tempnet = temp->succ;
        while(tempnet!=NULL) {
            fprintf(fp2, "succ net : %s₩n", tempnet->net->name);
            tempnet=tempnet->next;
        }
        *(logic_value + temp->level) = *(logic_value + temp->level) + temp->swcnt;
        fprintf(fp2, "*(logic_value + temp->level) = %d₩n",*(logic_value + temp->level)); //test
    }
    fprintf(fp2, "₩n************************* result **********************₩n");
    for(i=0;i<total_lv;i++)
        fprintf(fp2, "level[%d] total switching counts = %d₩n",i,*(logic_value+i));

    fclose(fp2);
    end = clock();
    printf("실행시간(로직) : %f sec ₩n실행시간(전체) : %f sec  ", (logic-logic_start)/1000, (end-start)/1000);

    while(1) {
        printf("do you want end??? (yes=1)  ");
        scanf("%d",&end);
        if(end==1)
           break;
```

```
        printf("do you want end??? (yes=1)  ");
        scanf("%d",&end);
        if(end==1)
           break;
    }
}

void Level_Decision(void)
{
    struct QUEUE *current = NULL;
    struct QUEUE *next = NULL;
    struct QUEUE *newqueue = NULL;
    struct NETLIST *tempnetlist = NULL;

    struct QUEUE *compared;
    int level = 0;
    int flag;

    //primary queue(input) 부터 시작
    current = primary_queue;/* write a code here */

    while(current != NULL)
    {
        //current queue의 link를 따라가면서 level을 결정
        for(current; current != NULL; current = current->link   /* write a code here */)
        {
            //현재 queue가 가르키는 net의 레벨 결정
            /* write a code here */current->net->level = level;

            //current queue가 가르키는 net의 출력 부분을 새로운 next queue로 등록하고, fanout이 하나 이상일 경우 고려
            for(tempnetlist=current->net->succ; tempnetlist != NULL; tempnetlist = tempnetlist->next)
            {
                if(next == NULL)
                {
                    //next queue에 새로운 queue 생성
                    next = Getqueue();

                    //next queue의 net이 current que가 가리키는 net의 출력 net에 연결되도록 지정
                    next->net = tempnetlist->net;
                }

                //next queue가 존재할 경우 새롭게 생성한 queue를 연결
                else
                {
                    flag=0;
                    for(compared=next;compared!=NULL;compared=compared->link){
                        if(tempnetlist->net==compared->net){
                            flag=1;
                        }
                    }
                    if(flag==0){
                        newqueue = Getqueue();
                        newqueue->link = next;
                        next = newqueue;
                        newqueue->net = tempnetlist->net;
```

```
                    newqueue->net = tempnetlist->net;
                }
            }
        }
    }
    //current queue를 next queue로 지정
    current = next;/* write a code here */
    next = NULL;
    level++;
}
NUMBEROFLEVEL = level;
}

/* c17_input.txt에 Input Vector가 들어 있습니다. */
int Logic_Calculation (char* in_vec)
{
    /* 레벨별 로직값 계산과 스위칭 계산하는 부분 */
    FILE* fp;

    char in[BUFFSIZE]; /* 파일에서 읽어들인 Input Vector를 한 글자씩 저장                *
                        * 예) Input Vector가 01010 이렇게 들어오면 in[0] = 0, in[1] = 1, *
                        *     in[2] = 0, in[3] = 1, in[4] = 0 이런 식으로 저장됨         */
    struct QUEUE* current;
    int in_num[BUFFSIZE]; /* in[]에서 읽어 들인 값(char 타입)을 int 타입으로 저장 처음character값을 int로 변환 */

    struct NET* head;
    struct QUEUE** lt;
    struct QUEUE* newq;
    int i, j, k, m;
    int flag;

    /* 파일을 여는 부분 구현 */
    fp = fopen (in_vec, "r");

    head = headnet;

    lt = (struct QUEUE**) malloc (NUMBEROFLEVEL * sizeof(struct lt*));
    for (k = 0; k <= NUMBEROFLEVEL; k ++) {
        lt[k] = NULL;
    }

    while (head != NULL) {
        newq = Getqueue( );
        newq->link = lt[head->level];
        lt[head->level] = newq;
        lt[head->level]->net = head;

        head = head->next;
    }

    flag = 0; /* flag 값, 초기화할 때 일어나는 switching은 제외함 */

    /* 입력된 Value를 한 단어씩 읽어 들인다.  이미 있는 함수 이용*/
    while (Read_a_word (in, fp) == TRUE)
    {
```

```
/* 입력된 Value를 한 단어씩 읽어 들인다. 이미 있는 함수 이용*/
while (Read_a_word (in, fp) == TRUE)
{
    i = 0;
    j = 0;
    /* Input Vector 안에 들어온 값을 입력 */
    while (in[i] != '₩0')
    {
        if(in[i]=='1')
        {
            in_num[i]=1;
        }
        else if(in[i]=='0')
        {
            in_num[i]=0;
        }
        i++;
        /* 불러 들인 값이 1이면 1을 저장 아니면 0을 저장한다. */
        /* Hint1: in[]에 저장된 값을 이용하세요. */
    }

    current = lt[j];
    for (k = 0; k < i; k ++) {
        m=current->net->sw;
        if(m != in_num[k])
        {
            current->net->sw=in_num[k];
            if(flag==1)
            {current->net->swcnt++;}
        }
        current=current->link;
        /* 처음 입력 된 값으로 초기화 함 */
        /* Hint1: 임시 변수에 현재 Value(=0)를 저장하고 비교 구문을 통해서       *
         * 입력 된 값에 의해 변화된 값과 비교 한다음 switching count를 증가한다. */
        /* Hint2: Queue를 잘 이용하세요 */
    }
    j ++;
    current = lt[j];

    while (j <= NUMBEROFLEVEL)
    {
        while (current != NULL)
        {

            m=current->net->sw;
            current->net->sw=Logic_Value(current->net);

            if(m != current->net->sw)
            {
                if(flag==1)
                {current->net->swcnt++;}
            }

            current=current->link;
```

```
                current=current->link;
                /* Logic Calculation 구현 //이미 구현한 한수를 사용해야 한다. 위에서// */
                /* Hint1: 초기화 부분(바로 위)랑 거의 흡사합니다. */
                /* Hint2: 이미 구현된 함수를 참조하세요. */
            }
            j ++;
            current = lt[j];
        }
        flag = 1;
    }

    /* 파일을 닫는부분 구현 */
    fclose(fp);
    return(NUMBEROFLEVEL);
}

int Logic_Value (struct NET* searchnet)
{
    struct NETLIST* templist;
    int sw_buffer =- 1;

    for (templist = searchnet->prev; templist != NULL; templist = templist->next)
    {
        if(sw_buffer == -1)
        {
            sw_buffer = templist->net->sw;
        }
        else
        {
            switch (searchnet->type){
                case AND:
                    sw_buffer = sw_buffer & templist->net->sw;
                    break;
                case OR:
                    sw_buffer = sw_buffer | templist->net->sw;
                    break;
                case NAND:
                    sw_buffer = sw_buffer & templist->net->sw;
                    break;
                case NOR:
                    sw_buffer = sw_buffer | templist->net->sw;
                    break;
                case XOR:
                    sw_buffer = sw_buffer ^ templist->net->sw;
                    break;
                case XNOR:
                    sw_buffer = sw_buffer ^ templist->net->sw;
                    break;
                case NOT:
                    sw_buffer = templist->net->sw;
                    break;
                case BUFF:
                    sw_buffer = templist->net->sw;
                    break;
                default:
```

```
                case BUFF:
                    sw_buffer = templist->net->sw;
                    break;
                default:
                    printf ("The Type Error of the %s net !!!\n", searchnet->name);
                    break;
            }
        }
    }

    if(searchnet->type == NAND || searchnet->type == NOR || searchnet->type == XNOR ||searchnet->type == NOT)
    {
        sw_buffer = !(sw_buffer);
    }

    return(sw_buffer);
}

void Insert_table (int hashcalc, struct NET* newnet, char* instring)
{
    //해당 위치에 table이 비었을 경우
    if ((table + hashcalc)->net == NULL)
    {
        sprintf((table + hashcalc)->name, "%s", instring); // netname 입력
        (table + hashcalc)->net = newnet; //생성된 net을 table에 등록
        newnet->name = (table + hashcalc)->name; //net 이름을 table 이름과 동기화
    }
    //table에 이미 net이 있을경우 table 확장
    else
    {
        struct NETTABLE* newtable=NULL;
        newtable = Gettable (); //table 새로 생성
        sprintf (newtable->name, "%s", instring); //netname 입력
        newtable->net = newnet; //table에 net 등록

        newtable->next = (table + hashcalc)->next; //확장된 table과 기존의 table 연결
        (table + hashcalc)->next = newtable;
        newnet->name = newtable->name; //net 이름을 table 이름과 동기화
    }
}

struct NET* Search_net(char* netname, int location)
{
    struct NETTABLE* temp;

    //해당 테이블이 가르키는 net이 netname과 일치할때
    if (strcmp ((table + location)->name, netname) == 0)
    {
        return ((table + location)->net); // 테이블 위치의 net값을 리턴
    }

    else
    {
        //확장된 테이블이 가르키는 net중에 netname과 일치하는 것이 있는지 검색
        for (temp = (table + location)->next; temp != NULL; temp = temp->next)
```

```
    else
    {
        //확장된 테이블이 가르키는 net중에 netname과 일치하는 것이 있는지 검색
        for (temp = (table + location)->next; temp != NULL; temp = temp->next)
        {
            if(strcmp (temp->name, netname) == 0)
            {
                return (temp->net);
            }
        }
    }
    return (NULL); //해당 net을 찾지 못했을 경우 null 포인터를 반환
}


struct NET* net_stack (void)
{
    struct NET* newnet = NULL;
    newnet = Getnet(); //새로운 net 생성

    if (newnet == NULL)
    {
        printf ("₩n Out of memory...");
        exit(1);
    }

    newnet->type = INPUT; //INPUT이라는 문자는 1로 맵핑
    newnet->next = headnet; //새로 생성된 net의 next가 headnet을 가리킴
    headnet = newnet; //새로 생성한 net을 headnet으로 다시 선언

    return (newnet);
}

struct NETLIST* link_netlist (struct NET* newnet)
{
    struct NETLIST* newlist=NULL;
    newlist = Getlist();
    newlist->next = newnet->succ; //netlist를 net의 기존 출력(netlist)와 연결
    newnet->succ = newlist; //net의 출력을 netlist와 연결
    return (newlist);
}

int Read_a_word (char* str, FILE* fp) {
    int k;
    char ch;

    while(1)
    {
        //파일을 끝까지 읽었으면 false 반환
        if (fscanf (fp, "%c", &ch) == EOF)
        {
            return (FALSE);
        }

        //다음과 같은 문자는 skip
```

```
        //다음과 같은 문자는 skip
        if (ch == '₩n' || ch == ' ' || ch == '₩t' || ch == '(' || ch == ')' || ch == ',' || ch== '=') {
            continue;
        }

        //그 밖의 문자를 만나면 break
        else {
            break;
        }
    }

    //문자열을 받아 str에 저장
    k = 0;
    str[k] = ch;
    while (1) {
        if(fscanf (fp, "%c", &ch) == EOF)
        {
            //입력이 없으면
            return (FALSE);
        }
        if(ch == '₩n' || ch == ' ' || ch == '₩t' || ch == '(' || ch == ')' || ch == ',' || ch == '=')
        {
            break;
        }
        k ++;
        str[k] = ch;
    }
    k ++;

    str[k] = '₩0'; //문자열의 마지막에 null 문자 삽입

    return (TRUE);
}
```