R E P O R T

순천향대학교

과목명 l 이산수학

담당교수 l 박두순 교수님

학과 l 컴퓨터소프트웨어 공학과

학년 l 2학년

학번 l 20164006

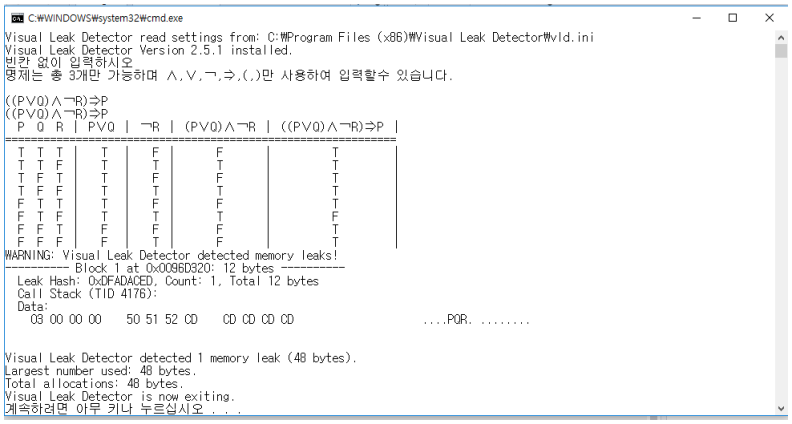
이름 l 박 훈

제출일 l 5월 12일 금요일

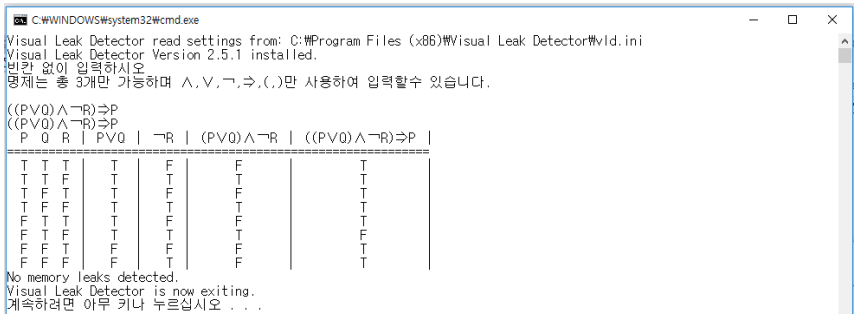
들어가는 글

2017년 4월 22일 ~5월 12일 이산수학에서 배운 연산자 우선 순위와 결합 법칙을 연구하여 진리표에 관하여 코딩을 하였다. 이번 레포트 코딩은 내가 단순히 생각했던 것보다 많은 경우의 수가 존재할 수 있었다. 하여 프로그램 코딩의 기간보다 여러가지 많은 합성 명제를 테스트 하는 데에 많은 기간이 사용되었던 것 같다.

또한 이번 레포트에는 구조체에 동적 할당을 사용한다. 동적 할당을 하여 힙 메모리를 사용할 시에는 반드시 메모리를 힙에 반환을 하여 메모리의 누수가 발생하지 않도록 해야한다. 동적 할당 해제 같은 경우 눈으로 반환이 되었는지 확인하기가 힘든 부분이다. 하여 vld의 확장 도구를 사용하여 메모리의 누수를 검사하도록 한다.



<메모리 누수가 존재할 경우>



<메모리가 누수가 존재하지 않을 경우>

수정사항 : 결론에 알고리즘에 접근에 대한 실행결과 목록 추가 진리표 알고리즘에 따른 문제 접근을 하여 알고리즘에 대한 처리가 실행 결과가 같은지 확인

얼마만큼 길이의 합성 명제 계산이 가능하지 측정

목차

Ⅰ. 서론

명제란?

명제는 참이나 거짓으로 구분할 수 있는 문장이나 수식을 한다.

논리 연산자(Logical Operation)의 종류

(1) 부정(Negation : NOT : ¬P) : 명제 P에 대해 P가 참(T)이면 ¬P 또는 ~P 거짓(F)가 되고, P가 거짓(F)이면 ¬P 또는 ~P는 참(T)가 된다.

(2) 논리곱(Conjunction : AND : p∧q) : 명제 p,q,에 대해 p,q의 진릿값이 모두 참(T)이면 p∧q는 참(T)이 되고, 그 외의 경우에는 거짓(F)가 된다.

(3) 논리합(Disjunction : OR : p∨q) : 명제 p,q에 대해 p,q의 진릿값 중 하나라도 참(T)이면 p∨q는 참(T)가 되고, p와 q의 진릿값이 모두 거짓(F)이면 p∨q는 거짓(F)이 된다.

합성명제(Compound Proposition)

AND, OR, NOT과 같은 논리 연산자에 의해 하나 이상의 명제들이 결합되어 만들어진 명제이다.

명제의 종류

(1) 항진명제(Tautology : T) : 합성명제를 구성하는 명제의 진릿값에 상관없이 진릿값이 항상 참(T)인 명제이다.

(2) 모순명제(Contradiction : F) : 합성명제를 구성하는 명제의 진릿값에 상관없이 진릿값에 항상 거짓(F)인 명제다.

(3) 사건명제(Contingency) : 항진명제도 모순명제도 아닌 명제다.

함축(Implication : p⇒q)

명제 p가 조건 또는 원인이 되고, 명제 q가 결론 또는 결과로 제시되는 명제로, "p는 q를 함축

한다." 또는 p면 q다"라고 읽는다. 명제 p가 참(T)이고 명제 q가 거짓(F)일 때만 p⇒q는 거짓(F)이 되고, 그 외의 경우에는 모두 참(T)이 된다.

진리표

합성 함수가 주어진 경우 논리적 추론에 맞게 연산 과정을 표로 작성한 표를 진리표라 한다.

프로젝트 목표

(1) 합성 명제는 많아야 3개로 제한한다.

(2) 논리 연산자는 and, or, not, implication 총 4가지 경우가 있다.

(3) 세개의 명제와 4개의 논리 연산자를 조합하여 합성 명제를 입력으로 주면 그에 대한 진리표가 출력되는 프로그램을 만들어라.

(4) 예제 2.13과 2.14를 적용하여 손으로 계산한 결과와 같은 결과가 나오는지 확인하여라.

Ⅱ. 본론

진리표 작성 방식

예제 2.13의 진리표

예제 2.13 ((P∨Q)∧ ¬R)⇒P						
P	Q	R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
T	T	T	T	F	F	T
T	T	F	T	T	T	T
T	F	T	T	F	F	T
T	F	F	T	T	T	T
F	T	T	T	F	F	T
F	T	F	T	T	T	F
F	F	T	F	F	F	T
F	F	F	F	T	F	T

P73 예제 2.13의 진리표는 다음과 같이 작성이 되어 있다. 위의 진리표를 통해 진리표의 작성 방식에 대하여 알아보겠다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P						
P	Q	R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
T	T	T	T	F	F	T

맨 첫번째 줄부터 살펴 보겠다. 명제로는 P Q R 3개의 명제가 존재하며 현재 P, Q, R값은 모두 참(T)값을 가지고 있다. 연산자 우선 순위에 맞춰서 표가 작성되어 있으며 P∨Q는 or 연산이므로 P와 Q중 둘 중 하나가 참(T)이면 참(T)이 되는 연산자이다. P와 Q가 모두 참(T)이므로 참(T)값을 출력하고 있다. 두번째 연산으로는 ¬R이다. 이것은 not연산이므로 R이 참(T)이면 거짓(F)를 반환하고 R이 거짓(F)이면 참(T)을 반환한다. 세번째 연산으로 (P∨Q)∧ ¬R는 첫번째 연산 결과와 두번째 연산 결과를 and연산을 한 결과를 반환한다. 하여 둘 중 하나라도 F이면 F이므로 T와 F and결과로 F가 반환되었다. 네번째 연산으로는 ((P∨Q)∧ ¬R)⇒P를 계산한다. 이 연산은 세번째 연산 결과와 P가 가지고 있는 값을 사용하여 implication연산을 실행한다. 이 연산 같은 경우 앞에 값이 T 뒤에 값이 F인 경우를 제외하고 전부 참(T)를 출력하므로 참(T)가 출력되

게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P				
P Q R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
T T F	T	T	T	T

P∨Q = p는 참(T)이고 Q는 참(T)이다. 참(T) or 참(T) 연산을 통해 참(T)가 출력된다.

¬R = R은 거짓(F)이기에 거짓(F)에 not연산을 하여 참(T)가 된다.

(P∨Q)∧ ¬R = 첫번째 계산 값(T)과 두번째 계산 값(T)을 이용하여 and연산을 하여 T가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(T)와 P의 값(T)을 사용하여 implication연산을 실행하여 T가 출력되게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P				
P Q R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
T F T	T	F	F	T

P∨Q = p는 참(T)이고 Q는 거짓(F)이다. 참(T) or 거짓(F) 연산을 통해 참(T)가 출력된다.

¬R = R은 참(T)이기에 참(T)에 not연산을 하여 거짓(F)가 된다.

(P∨Q)∧ ¬R = 첫번째 계산 값(T)과 두번째 계산 값(F)을 이용하여 and연산을 하여 F가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(F)와 P의 값(T)을 사용하여 implication연산을 실행하여 T가 출력되게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P				
P Q R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
T F F	T	T	T	T

P∨Q = p는 참(T)이고 Q는 거짓(F)이다. 참(T) or 거짓(F) 연산을 통해 참(T)가 출력된다.

¬R = R은 거짓(F)이기에 거짓(F)에 not연산을 하여 참(T)가 된다.

(P∨Q)∧ ¬R = 첫번째 계산 값(T)과 두번째 계산 값(T)을 이용하여 and연산을 하여 T가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(T)와 P의 값(T)을 사용하여 implication연산을 실행하여 T가 출력되게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P				
P Q R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
F T T	T	F	F	T

P∨Q = p는 거짓(F)이고 Q는 참(T)이다. 거짓(F) or 참(T) 연산을 통해 참(T)가 출력된다.

¬R = R은 참(T)이기에 참(T)에 not연산을 하여 거짓(F)가 된다.

(P∨Q)∧ ¬R = 첫번째 계산 값(T)과 두번째 계산 값(F)을 이용하여 and연산을 하여 F가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(F)와 P의 값(F)을 사용하여 implication연산을 실행하여 T가 출력되게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P				
P Q R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
F T F	T	T	T	F

P∨Q = p는 거짓(F)이고 Q는 참(T)이다. 거짓(F) or 참(T) 연산을 통해 참(T)가 출력된다.

¬R = R은 거짓(F)이기에 거짓(F)에 not연산을 하여 참(T)가 된다.

(P∨Q)∧ ¬R = 첫번째 계산 값(T)과 두번째 계산 값(T)을 이용하여 and연산을 하여 T가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(T)와 P의 값(F)을 사용하여 implication연산을 실행하여 F가 출력되게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P				
P Q R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P

F	F	T	F	F	F	T

P∨Q = p는 거짓(F)이고 Q는 거짓(F)이다. 거짓(F) or 거짓(F) 연산을 통해 참(F)가 출력된다.

¬R = R은 참(T)이기에 참(T)에 not연산을 하여 거짓(F)가 된다.

(P∨Q)∧ ¬R = 첫번째 계산(F)과 두번째 계산 값(F)을 이용하여 and연산을 하여 F가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(F)와 P의 값(F)을 사용하여 implication연산을 실행하여 T가 출력되게 된다.

예제 2.13 ((P∨Q)∧ ¬R)⇒P					
P	Q	R	P∨Q	¬R	(P∨Q)∧ ¬R	((P∨Q)∧ ¬R)⇒P
F	F	F	F	T	F	T

P∨Q = p는 거짓(F)이고 Q는 거짓(F)이다. 거짓(F) or 거짓(F) 연산을 통해 참(F)가 출력된다.

¬R = R은 거짓(F)이기에 거짓(F)에 not연산을 하여 참(T)가 된다.

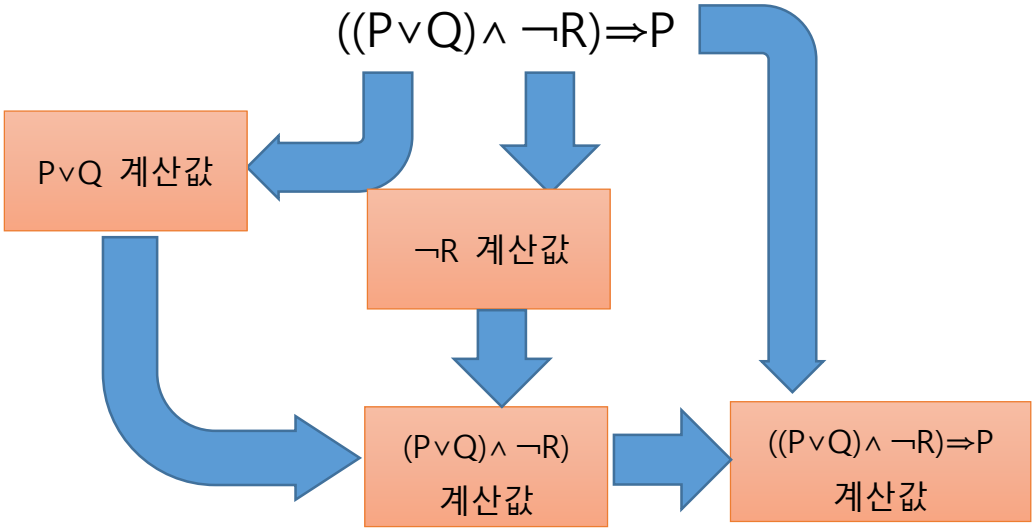
(P∨Q)∧ ¬R = 첫번째 계산(F)과 두번째 계산 값(T)을 이용하여 and연산을 하여 F가 출력 된다.

((P∨Q)∧ ¬R)⇒P = 세번째 계산 값(F)와 P의 값(F)을 사용하여 implication연산을 실행하여 T가 출력되게 된다.

프로그램 구상

1. 합성 명제를 입력 받는다.
2. 합성 명제 개수를 카운트한다.
3. 연산자 우선 순위로 연산 과정을 출력해준다.
4. 표의 칸을 맞춘다.

이상적인 진리표 알고리즘



문제 해결방식(1차)

연산 다음 나올 수 있는 것

¬ : (, 피연산

∨ : (, ¬, 피연산

∧ : (, ¬, 피연산

⇒ : (, ¬, 피연산

피연산자 경우 출력

하나의 피연산자에서 나올 수 있는 경우는 true, false 총 두가지가 존재하게 된다. 하여 2^피연산자의 개수 = 나올 수 있는 경우의 수가 된다. 하여 만약 경우의 수가 8이면 0~7을 이진수로 보았다.

0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	0	1	0
5	1	0	1
6	1	1	0

7	1	1	1

여기서 0을 T로 1을 F로 출력을 하였다.

0	T	T	T
1	T	T	F
2	T	F	T
3	T	F	F
4	F	T	T
5	F	T	F
6	F	F	T
7	F	F	F

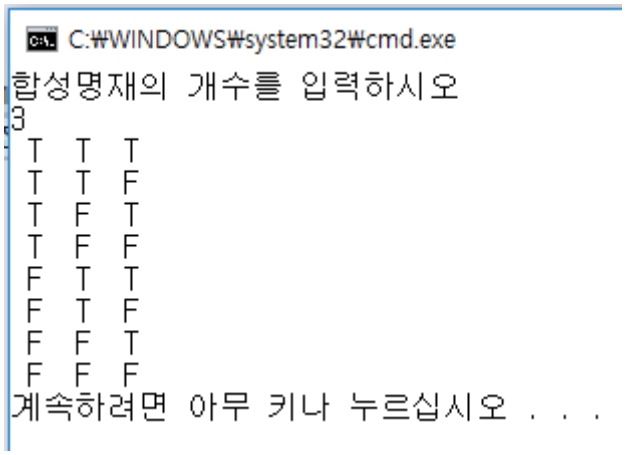
이런 방식을 사용하여 피연산의 경우를 출력하도록 하였다.

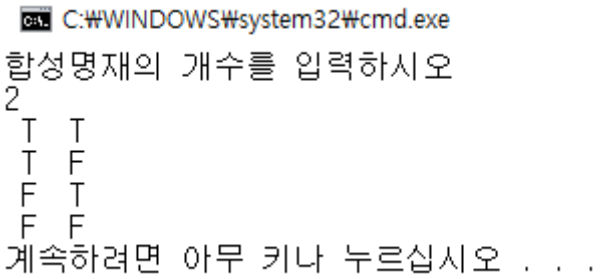
피연산자의 경우 출력 소스

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
int main()
{
    int i = 0,k,temp1;
    int x;
    printf("합성명재의 개수를 입력하시오₩n");
    scanf("%d", &x);
    bool *array1 = (bool*)malloc(sizeof(bool)*x);
    for (i = 0; i < (1<<x); i++)
    {
        temp1 = i;
        for (k = x-1; k >= 0; k--)
        {
            if (temp1 / (1 << k))
            {
                array1[x - k - 1] = false;
                printf(" F ");
            }
```

```
21              else
22              {
23                  array1[x - k - 1] = true;
24                  printf(" T ");
25              }
26              temp1 = temp1%(1 << k);
27          }
28          printf("₩n");
29      }
30      free(array1);
31      return 0;
32  }
```

디버깅 결과





다음과 같이 피연산의 경우가 잘 작동함을 볼 수가 있다.

hash_map

앞에서 피연산자의 경우를 찾는 방식의 알고리즘에 대하여 설명을 하였다. 이러한 방식을 사용하여 진리표를 작성하기 위해서는 명제에 알맞은 진릿값을 넣고 찾아야 한다. 하여 명제의 알맞은 피연산자 값을 찾는 탐색 알고리즘이 요구 되어진다. 이에 대한 탐색 알고리즘으로는 순차 탐색 방식과 map방식을 생각을 할 수가 있었다. 순차 탐색은 하나하나씩 비교를 하면서 알맞은 값을 탐색하는 알고리즘이다. 하지만 이러한 방식은 명제의 개수가 많을 경우 실행속도가 매우 느려질 수 있는 단점이 있다. 하여 지난주 스터디 때 공부한 c++의 map을 생각하게 되었다. #include<map>을 선언하여 사용할 수 있으며 c++표준 라이브러리 함수이다. map에는 key값과 value값이 필요로 한다. key값을 사용하여 탐색하면 얼마나 편리할까? 이 키 값은 문자열이 될 수도 있고 다양한 타입형이 될 수도 있다. 예를 들면 key값에 "이산수학"을 넣고 value값에 "3학점"이라는 것을 넣으면 배열의 index 접근처럼 key값에 "이산수학"을 입력했을 시에 value값으로 "3학점"이라는 값을 도출할 수 있다. Map의 알고리즘은 옛날엔 이진 트리 방식(BST)를 사용했지만 현재는 레드 블랙 트리를 사용한다고 들었다. 하여 c언어에는 map와 비슷한 함수가 있는지 찾아보다가 hashmap이란 것을 찾아 볼 수가 있었다. 이것은 vc(visual-studio)에는 #include<hash_map>으로 존재는 하지만 c++표준라이브러리는 아니라고 한다. 맵과 해시 맵의 차이는 맵 같은 경우는 자료를 정렬하여 저장을 하기에 정렬된 자료가 필요할 경우 아주 유용하다. 해시 맵의 경우는 정렬하여 저장을 하지 않는다. 그리고 무엇보다 가장 큰 차이는 실행 속도의 차이 인 것 같다. 확실히 해시 맵이 배열로 접근하기 때문에 많이 빠른 것을 느낀다. 하지만 현실적으로 탐색 키들이 문자열이거나 매우 큰 숫자이면 배열의 index로 사용하기에는 많은 무리가 있으므로 탐색 키를 작은 정수로 사상 시키기 위한 사상 함수를 사용한다. 이 사상 함수는 key들의 특징이나 성질을 이용하여 key값이 충동이 일어나지 않도록 하여야 한다. 하여 A~Z까지의 명제를 아스키 코드값을 이용하여 알파벳의 아스키코드값이 가장 작은 A를 빼서 A~Z까지의 순서를 key값으로 사상시키는 함수를 작성하였다.

hash_map 사상함수

```
1 int hash_key(char x)
2 {
3     if ('A' <= x&&'Z' >= x)
4         return x - 'A';
5     else if ('a' <= x&&'z' >= x)
6         return x - 'a';
```

```
    return -1;
}
```

함수를 사용한 괄호 연산

괄호란 연산의 순서를 우선순위로 하고 싶은 곳에 표시하는 기법이다. 다시 말해 괄호를 만나게 될 경우 괄호 안에 있는 연산을 먼저 실행을 해야한다. 이번 프로그램에서 제일 어려웠던 부분 중에 하나였다. 하여 괄호를 만나면 닫는 괄호까지의 범위를 따로 호출하여 계산하면 되지 않을까? 라는 생각을 하게 되었습니다. 하여 이를 함수의 범위 호출로 만들고 그에 대한 부분의 연산 결과를 함수 반환하는 형태로 코딩을 하였습니다. 그리고 이미 실행하여 계산된 괄호는 다시 계산할 필요가 없기에 visit의 배열에 연산한 연산 기호를 숫자로 저장하여 방문한 곳을 다시 계산하지 않도록 하였습니다.

코딩 소스

```
#include<stdio.h>
#include<stdbool.h>
#include<string.h>
#include<stdlib.h>
#include<vld.h>
#define and(x,y) x&y//∧일때
#define or(x,y) x|y//∨일때
#define not(x) !x// ¬일때
#define implication(x,y) !((x==true)&&(y==false))//⇒일때
typedef struct alphabet
{
    int alphabet_count;
    char alphabet[3];
    bool logic[3];
}Alphabet;
bool find_alphabet(char alphabet[], char c);
void print_check_logic(Alphabet*proposition, char result[], int i);
void alphabet_save(Alphabet *proposition, char result[]);
void draw(Alphabet*proposition, char result[]);
```

```
int move;
int visit[1000];
bool hash_map[27];
int p_lenth[1000];
int pos_len;
int cow_len;
int hash_key(char x)
{
    if ('A' <= x&&'Z' >= x)
        return x - 'A';
    else if ('a' <= x&&'z' >= x)
        return x - 'a';
    return -1;
}
void print_check_logic(Alphabet*proposition, char result[], int i)
{
    int len = strlen(result);
    int j = i;
    int o = i;
    int k;
    int p;
    char name[3];
    for (; i < len; i++)
    {
        name[0] = result[i];
        name[1] = result[i + 1];
        name[2] = '₩0';
        if (result[i] == '('&&visit[i] != 1)
        {
            visit[i] = 1;
            print_check_logic(proposition, result, i + 1);
        }
        if (!strcmp(name, "¬") && visit[i] != 3 && visit[i] != 6)
        {
            p = i;
            if (visit[p] == 4)
            {
                visit[p] = 6;
```

```
58          }
59          else
60              visit[p] = 3;
61          k = i;
62          i = i + 2;
63          if (result[i] == '('&&visit[i] != 1)
64          {
65              cow_len = cow_len + 2;
66              visit[i] = 1;
67              print_check_logic(proposition, result, i + 1);
68              p_lenth[pos_len] = move - k;
69              cow_len = cow_len + p_lenth[pos_len] + 4;
70              pos_len++;
71              for (; k <= move; k++)
72                  printf("%c", result[k]);
73              printf("  |  ");
74          }
75          else
76          {
77              p_lenth[pos_len] = 3;
78              cow_len = cow_len + p_lenth[pos_len] + 4;
79              pos_len++;
80              printf("%s%c  |  ", name, result[i]);
81          }
82          if (visit[p] == 6)
83          {
84              return 0;
85          }
86      }
87      if (!strcmp(name, "v") && visit[i] != 7)
88      {
89          visit[i] = 7;
90          k = i;
91          i = i + 2;
92          name[0] = result[i];
93          name[1] = result[i + 1];
94          name[2] = '₩0';
95          if (result[i] == '('&&visit[i] != 1)
```

```
96          {
97              visit[i] = 1;
98              print_check_logic(proposition, result, i + 1);
99              p_lenth[pos_len] = move - k;
10              cow_len = cow_len + p_lenth[pos_len] + 4;
0               pos_len++;
10              for (; k <= move; k++)
1                   printf("%c", result[k]);
10              printf("  |  ");
2           }
10          if (!strcmp(name, "¬") && visit[i] != 4)
3           {
10              visit[i] = 4;
4               print_check_logic(proposition, result, i);
10          }
5       }
10      if (!strcmp(name, "∧") && visit[i] != 5)
6       {
10          visit[i] = 5;
7           k = i;
10          i = i + 2;
8           name[0] = result[i];
10          name[1] = result[i + 1];
9           name[2] = '₩0';
11          if (result[i] == '('&&visit[i] != 1)
0           {
11              visit[i] = 1;
1               print_check_logic(proposition, result, i + 1);
11              p_lenth[pos_len] = move - k;
2               cow_len = cow_len + p_lenth[pos_len] + 4;
11              pos_len++;
3               for (; k <= move; k++)
11                  printf("%c", result[k]);
4               printf("  |  ");
11          }
5           if (!strcmp(name, "¬") && visit[i] != 4)
11          {
6               visit[i] = 4;
```

```
11              print_check_logic(proposition, result, i);
7           }
11      }
8       if (!strcmp(name, "⇒") && visit[i] != 8)
11      {
9           //printf("ads");
12          visit[i] = 8;
0           k = i;
12          i = i + 2;
1           name[0] = result[i];
12          name[1] = result[i + 1];
2           name[2] = '₩0';
12          if (result[i] == '('&&visit[i] != 1)
3           {
12              visit[i] = 1;
4               print_check_logic(proposition, result, i + 1);
12              p_lenth[pos_len] = move - o;
5               cow_len = cow_len + p_lenth[pos_len] + 4;
12              pos_len++;
6               for (o; o <= move; o++)
12                  printf("%c", result[o]);
7               printf("  |  ");
12          }
8           else if (!strcmp(name, "¬") && visit[i] != 4)
12          {
9               k = i + 2;
13              visit[i] = 4;
0               print_check_logic(proposition, result, i);
13              p_lenth[pos_len] = k - o;
1               cow_len = cow_len + p_lenth[pos_len] + 4;
13              pos_len++;
2               for (o; o <= k; o++)
13                  printf("%c", result[o]);
3               printf("  |  ");
13          }
4           else
13          {
5               k = i++;
```

```
13                  p_lenth[pos_len] = k - o;
6                   cow_len = cow_len + p_lenth[pos_len] + 4;
13                  pos_len++;
7                   for (o; o <= k; o++)
13                      printf("%c", result[o]);
8                   printf("  |  ");
13              }
9           }
14          if (result[i + 1] == ')'&&visit[i] != 2)
0           {
14              move = i + 1;
1               visit[i] = 2;
14              p_lenth[pos_len] = i - j;
2               cow_len = cow_len + p_lenth[pos_len] + 4;
14              pos_len++;
3               for (j; j <= i; j++)
14                  printf("%c", result[j]);
4               printf("  |  ");
14              return 0;
5           }
14      }
6       memset(&visit, 0, 1000);
14      move = 0;
7       pos_len = 0;
14      return 0;
8   }
14  bool result_check_logic(Alphabet*proposition, char result[], int i)
9   {
15      int len = strlen(result);
0       int j = i;
15      int k;
1       int p, l;
15      char name[3];
2       bool ref = 1, ref1;
15      int lenth = 0;
3       if (find_alphabet(proposition->alphabet, result[i]))
15      {
4           ref = hash_map[hash_key(result[i])];
```

```
15      }
5       ////////////////////////////////
15      for (; i < len; i++)
6       {
15          name[0] = result[i];
7           name[1] = result[i + 1];
15          name[2] = '₩0';
8           ////////////////////
15          if (result[i] == '('&&visit[i] != 1)
9           {
16              visit[i] = 1;
0               ref = result_check_logic(proposition, result, i + 1);
16          }
1           ///////////////
16          if (!strcmp(name, "¬") && visit[i] != 3 && visit[i] != 6)
2           {
16              p = i;
3               if (visit[p] == 4)
16              {
4                   visit[p] = 6;
16              }
5               else
16                  visit[p] = 3;
6               k = i;
16              i = i + 2;
7               if (result[i] == '('&&visit[i] != 1)
16              {
8                   visit[i] = 1;
16                  ref = result_check_logic(proposition, result, i + 1);
9                   ref = not(ref);
17                  lenth = p_lenth[pos_len];
0                   pos_len++;
17                  if (lenth % 2 == 1)
1                       for (l = 0; l < lenth / 2 + 1; l++)printf(" ");
17                  else
2                       for (l = 0; l < lenth / 2; l++)printf(" ");
17                  if (ref)
3                       printf("T");
```

```
            else
                printf("F");
            for (l = 0; l < lenth / 2; l++)printf("  ");

            printf("  |  ");
        }
        else
        {
            ref = hash_map[hash_key(result[i])];
            ref = not(ref);
            pos_len++;
            if (ref)
                printf("  T");
            else
                printf("  F");
            //printf("%s%c  |  ", name, result[i]);
            printf("  |  ");
        }

        if (visit[p] == 6)
        {
            return ref;
        }
    }
    ////////////////////
    if (!strcmp(name, "v") && visit[i] != 4)
    {
        visit[i] = 4;
        k = i;
        i = i + 2;
        name[0] = result[i];
        name[1] = result[i + 1];
        name[2] = '₩0';
        if (result[i] == '('&&visit[i] != 1)
        {
            visit[i] = 1;
            ref1 = result_check_logic(proposition, result, i + 1);
            /*for (; k <= move; k++)
```

```
19              printf("%c", result[k]);*/
3               //printf("  |  ");
19          }
4           if (!strcmp(name, "¬") && visit[i] != 4)
19          {
5               visit[i] = 4;
19              ref1 = result_check_logic(proposition, result, i);
6           }
19          else
7           {
19              ref1 = hash_map[hash_key(result[i])];
8           }
19          //printf("%d %d", pos_len,p_lenth[pos_len]);
9           ref = or (ref, ref1);
20          lenth = p_lenth[pos_len];
0           pos_len++;
20          if (lenth % 2 == 1)
1               for (l = 0; l < lenth / 2 + 1; l++)printf(" ");
20          else
2               for (l = 0; l < lenth / 2; l++)printf(" ");
20          if (ref)
3               printf("T");
20          else
4               printf("F");
20          for (l = 0; l < lenth / 2; l++)printf(" ");
5
20          printf("  |  ");
6       }
20      //////////////////
7
20      //////////////
8       if (!strcmp(name, "∧") && visit[i] != 5)
20      {
9           visit[i] = 5;
21          k = i;
0           i = i + 2;
21          name[0] = result[i];
1           name[1] = result[i + 1];
```

```
            name[2] = '₩0';
            if (result[i] == '('&&visit[i] != 1)
            {
                visit[i] = 1;
                ref1 = result_check_logic(proposition, result, i + 1);
                /*for (; k <= move; k++)
                printf("%c", result[k]);*/
                //printf("  |  ");
            }
            else if (!strcmp(name, "¬") && visit[i] != 4)
            {
                visit[i] = 4;
                ref1 = result_check_logic(proposition, result, i);
            }
            else
            {
                ref1 = hash_map[hash_key(result[i])];
            }
            ref = and (ref, ref1);
            lenth = p_lenth[pos_len];
            pos_len++;
            if (lenth % 2 == 1)
                for (l = 0; l < lenth / 2 + 1; l++)printf(" ");
            else
                for (l = 0; l < lenth / 2; l++)printf(" ");
            if (ref)
                printf("T");
            else
                printf("F");
            for (l = 0; l < lenth / 2; l++)printf(" ");

            printf("  |  ");
        }
        ////////////////////////
        if (!strcmp(name, "⇒") && visit[i] != 8)
        {
            visit[i] = 8;
            k = i;
```

```
            i = i + 2;
            name[0] = result[i];
            name[1] = result[i + 1];
            name[2] = '₩0';
            if (result[i] == '('&&visit[i] != 1)
            {
                visit[i] = 1;
                ref1 = result_check_logic(proposition, result, i + 1);
                ref = implication(ref, ref1);

            }
            else if (!strcmp(name, "¬") && visit[i] != 4)
            {
                visit[i] = 4;
                ref1 = result_check_logic(proposition, result, i);
                ref = implication(ref, ref1);
            }
            else
            {
                ref1 = hash_map[hash_key(result[i])];
                ref = implication(ref, ref1);
            }
            lenth = p_lenth[pos_len];
            pos_len++;
            if (lenth % 2 == 1)
                for (l = 0; l < lenth / 2 + 1; l++)printf(" ");
            else
                for (l = 0; l < lenth / 2; l++)printf(" ");
            if (ref)
                printf("T");
            else
                printf("F");
            for (l = 0; l < lenth / 2; l++)printf(" ");

            printf("  |  ");

        }
        ////////////////////
```

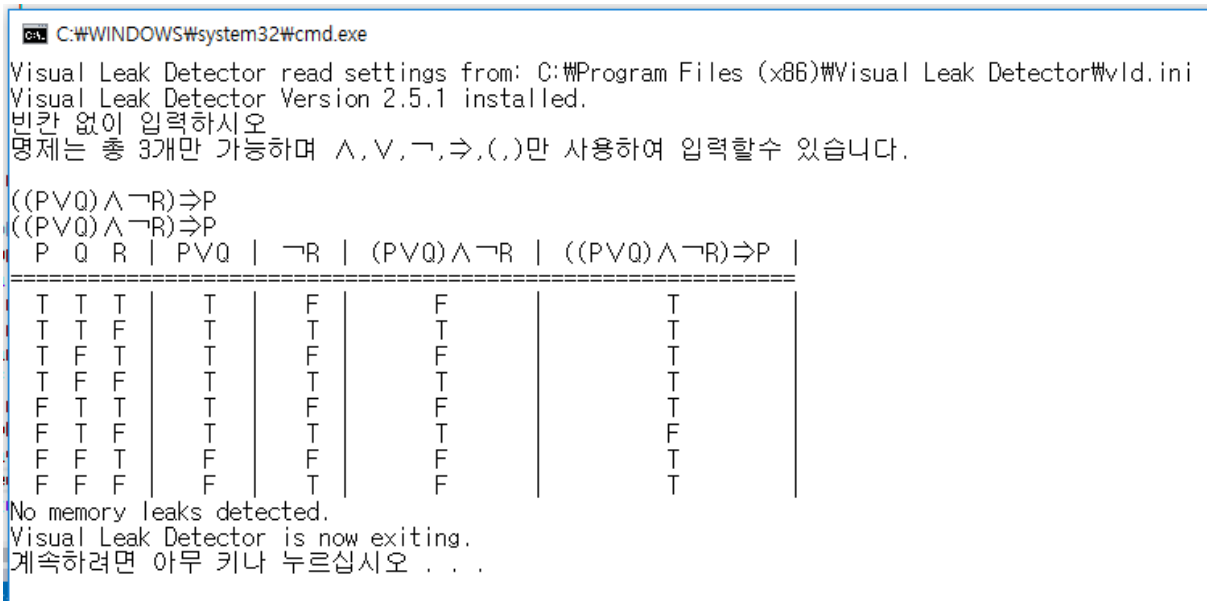
```
25          if (result[i + 1] == ')'&&visit[i] != 2)
0           {
25              move = i + 1;
1               visit[i] = 2;
25              /*for (j; j <= i; j++)
2               printf("%c", result[j]);*/
25              return ref;
3           }
25      }
4       memset(&visit, 0, 1000);
25      move = 0;
5       pos_len = 0;
25      return true;
6   }
25  bool find_alphabet(char alphabet[], char c)
7   {
25      int i;
8       for (i = 0; i < strlen(alphabet); i++)
25          if (alphabet[i] == c)
9               return true;
26      return false;
0   }
26  void file_read(char result[])
1   {
26      FILE*fp = NULL;
2       fp = fopen("data.txt", "rt");
26      if (fp == NULL)
3       {
26          printf("실패");
4           return -1;
26      }
5       fscanf(fp, "%s", result);
26      fclose(fp);
6   }
26  void alphabet_save(Alphabet *proposition, char result[])
7   {
26      int i;
8       int len = strlen(result);
```

```
    int cnt = 0;
    for (i = 0; i < len; i++)
    {
        if (((result[i] >= 'a'&&result[i] <= 'z') || (result[i] >= 'A'&&result[i] <= 'Z')) && !find_alphabet(proposition->alphabet, result[i]))
        {
            proposition->alphabet[proposition->alphabet_count] = result[i];
            proposition->alphabet_count++;
        }
        if (proposition->alphabet_count > 3)
        {
            printf("오류 : 합성명제 개수 많음₩n");
            exit(1);
        }
        if (result[i] == '(')
        {
            cnt++;
        }
        if (result[i] == ')')
        {
            cnt--;
        }
    }
    if (cnt != 0)
    {
        printf("괄호 짝안맞음₩n");
        exit(1);
    }
}
void draw(Alphabet*proposition, char result[])
{
    int i = 0, k, temp1, j = 0;
    for (i = 0; i < proposition->alphabet_count; i++)
        printf("%3c", proposition->alphabet[i]);
    printf("  |  ");
    print_check_logic(proposition, result, 0);
    //check_logic(proposition, result, 0);
    printf("₩n====");
```

```
28      for (i = 0; i < proposition->alphabet_count; i++)printf("===");
8       for (i = 0; i <= cow_len; i++)printf("=");
28      printf("₩n");
9       for (i = 0; i < (1 << proposition->alphabet_count); i++)
29      {
0           temp1 = i;
29          j = 0;
1           for (k = proposition->alphabet_count - 1; k >= 0; k--)
29          {
2               if (temp1 / (1 << k))
29              {
3                   hash_map[hash_key(proposition->alphabet[j])] = false;
29                  printf("%3c", 'F');
4               }
29              else
5               {
29                  hash_map[hash_key(proposition->alphabet[j])] = true;
6                   printf("%3c", 'T');
29              }
7               temp1 = temp1 % (1 << k);
29              j++;
8           }
29          printf("  |  ");
9           //print_check_logic(proposition, result, 0);
30          result_check_logic(proposition, result, 0);
0           printf("₩n");
30      }
1   }
30  int main(void)
2   {
30      Alphabet *proposition = (Alphabet*)malloc(sizeof(Alphabet));
3       proposition->alphabet_count = 0;
30      char result[1000];
4       printf("빈칸 없이 입력하시오₩n");
30      printf("명제는 총 3개만 가능하며 ∧,∨,¬,⇒,(,)만 사용하여 입력할수 있습니다.₩n₩n");
5       scanf("%s", result);
30      //file_read(result);//파일로 값을 불러옴
6       printf("%s₩n", result);
```

```
30      alphabet_save(proposition, result);
7       draw(proposition, result);
30      free(proposition);
8       return 0;
30  }
```

Vld 디버깅 결과



Ⅲ. 결론

알고리즘 접근에 대한 실행 결과

((P∨Q)∧¬R)⇒¬P를 가지고 알고리즘으로 접근을 해보겠다. P=T, Q=T, R=T로 가정을 하고 연산을 해보겠다.

(를 만났으므로 괄호 안의 계산을 수행하기 위해 괄호 안의 범위부터 함수를 다시 호출을 한다.

두번째 괄호를 만났으므로 괄호안의 계산을 수행하기 위해 괄호 안의 범위로부터 함수를 다시 호출한다.

P ∨Q를 연산을 수행하여 T ∨ T이므로 T가 나오게 된다.

) 닫는 괄호를 만났으므로 두번째 괄호에서 진행했던 연산 값(T)을 반환한다.

두번째 괄호에서 반환된 값 ∧¬R을 연산을 수행을 하기 전에 ¬연산이 있으므로 ¬R을 먼저 계산한다. 이를 계산하기 위해 ¬R부분을 재귀 호출을 하여 먼저 계산을 실행을 한 다음에 그에 대한 결과를 반환하도록 한다. R이 T이므로 ¬R의 결과는 F이다.

두번째 괄호에서 반환된 값 ∧¬R을 연산을 수행한다. T ∧ F 이므로 F가 나오게 된다.

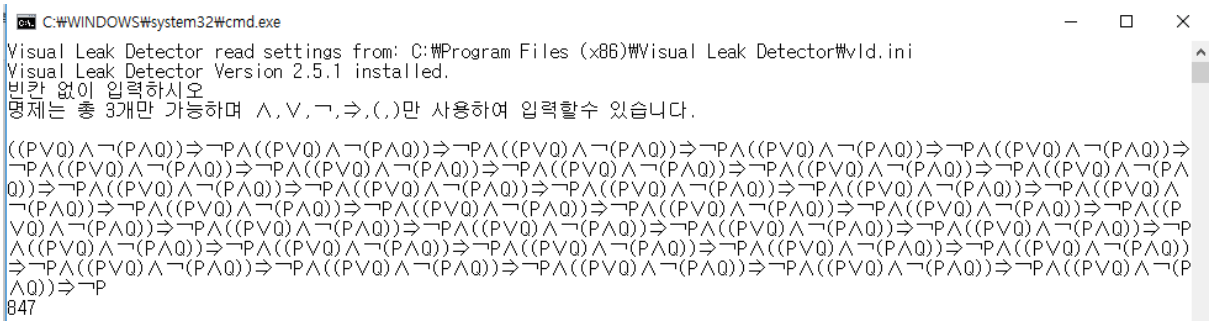
) 닫는 괄호를 만났으므로 첫번째 괄호에서 진행했던 연산 값(F)을 반환한다

반환된 F값과 ⇒¬P의 연산을 진행한다. ¬연산자를 만났으므로 재귀 호출로 먼저 계산을 진행하게 된다. P의 값은 T이므로 ¬P값을 F가 나오게 된다.

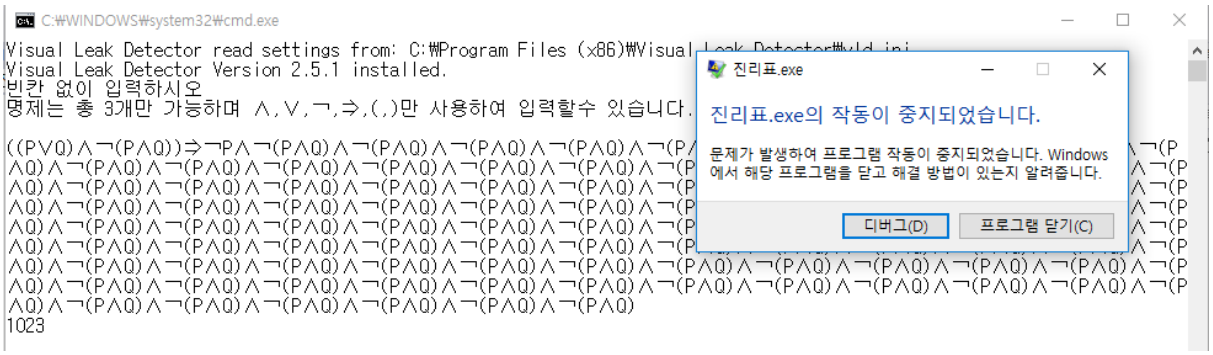
첫번째 괄호의 반환된 값과 ⇒F값을 계산하여 T라는 값을 낸다. 연산의 끝에 도달했으므로 계산을 끝낸다.

얼마만큼 길이의 합성 명제의 계산이 가능할까?

입력된 문자열의 길이를 출력하도록 하여 프로그램이 진행이 되도록 해보았다.

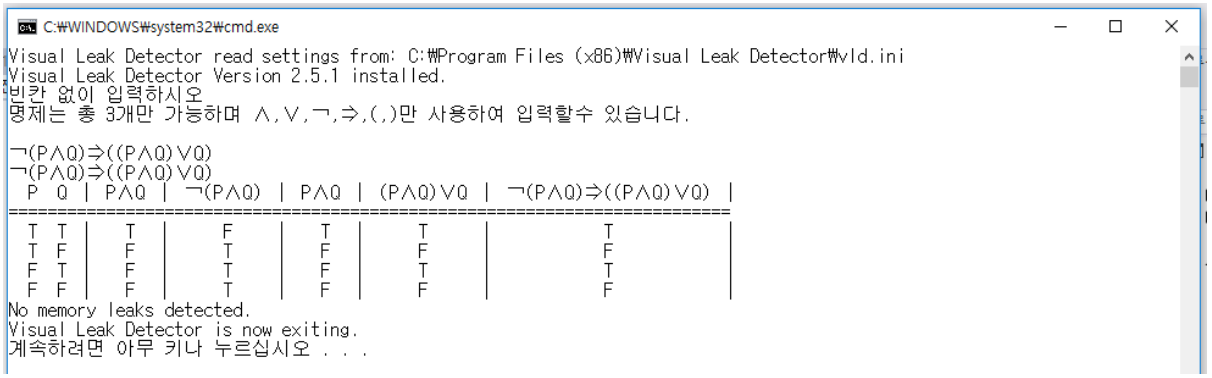


847길이의 합성 명제는 오버라이딩이 발생하지 않았다.



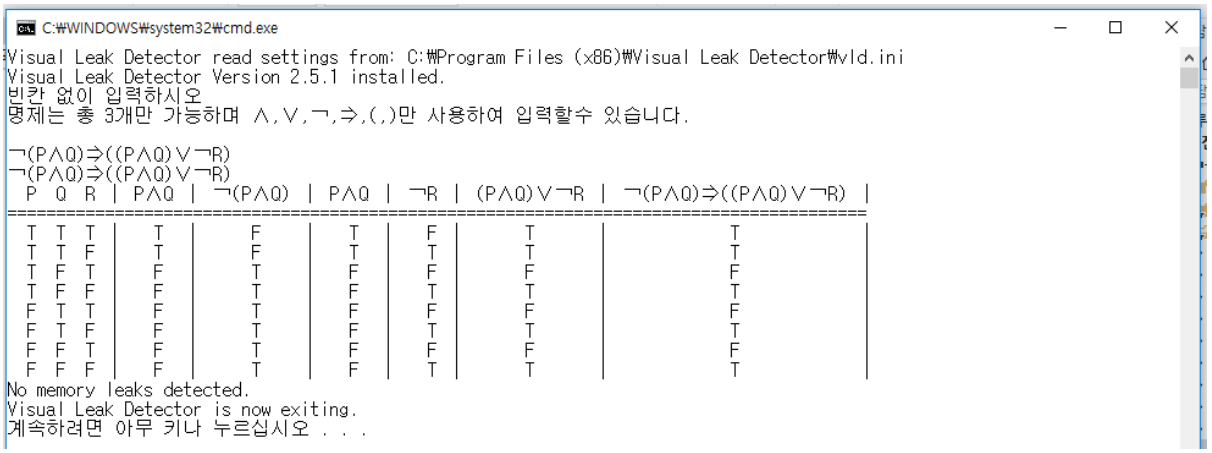
1023길이의 합성 명제에서 오버라이딩이 발생하였음을 확인할 수가 있었다. 현재 합성 명제를 문자열을 입력 받는 배열의 크기는 배열의 길이를 1000으로 설정 하였으므로 char(1 Byte)* 1000이므로 1000byte가 된다. 여기서 1000Byte의 문자열 배열에 합성 명제를 저장하는 부분에서 문자열의 끝을 구분하기 위한 '₩0'문자에 의하여 999Byte가 저장이 가능한데 유니코드는 2Byte의 공간이 필요로 한다. 하여 999길이만큼의 합성 문자도 읽지 못하게 되는 것이다. 이러한 오버라이딩 문제를 해결하기 위해 물론 배열사이즈를 더 크게 선언하는 방법이 있다. 하지만 단점으로는 합성 명제의 길이가 작은 경우 많은 불필요한 메모리가 생길 수 있다. 이러한 불필요한 메모리는 프로그램이 종료가 될 때까지 많은 메모리를 사용하게 된다.

테스트 결과



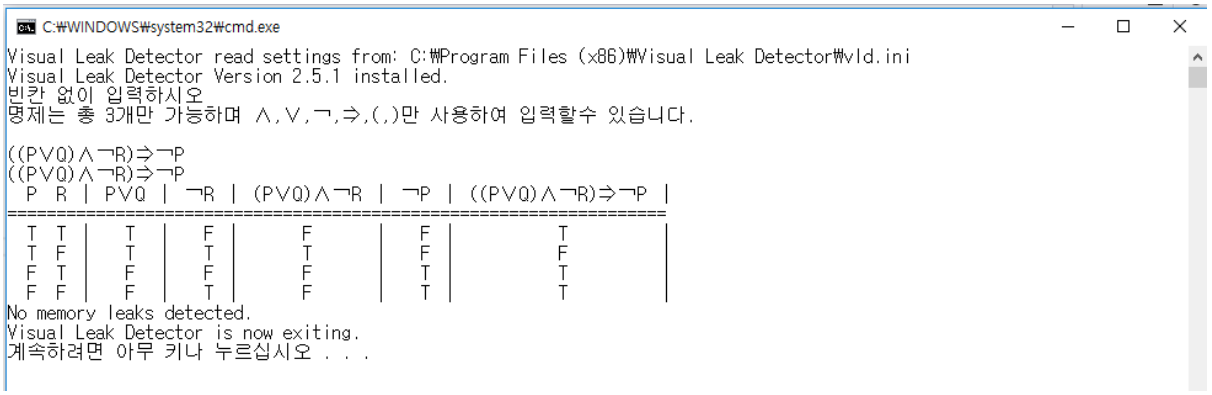
```
C:₩WINDOWS₩system32₩cmd.exe
Visual Leak Detector read settings from: C:₩Program Files (x86)₩Visual Leak Detector₩vld.ini
Visual Leak Detector Version 2.5.1 installed.
빈칸 없이 입력하시오
명제는 총 3개만 가능하며 ∧,∨,¬,⇒,(,)만 사용하여 입력할수 있습니다.

¬(P∧Q)⇒((P∧Q)∨Q)
¬(P∧Q)⇒((P∧Q)∨Q)
  P  Q | P∧Q | ¬(P∧Q) | P∧Q | (P∧Q)∨Q | ¬(P∧Q)⇒((P∧Q)∨Q) |
=======================================================================
  T  T |   T |     F |   T |       T |                 T |
  T  F |   F |     T |   F |       F |                 F |
  F  T |   F |     T |   F |       T |                 T |
  F  F |   F |     T |   F |       F |                 F |
No memory leaks detected.
Visual Leak Detector is now exiting.
계속하려면 아무 키나 누르십시오 . . .
```



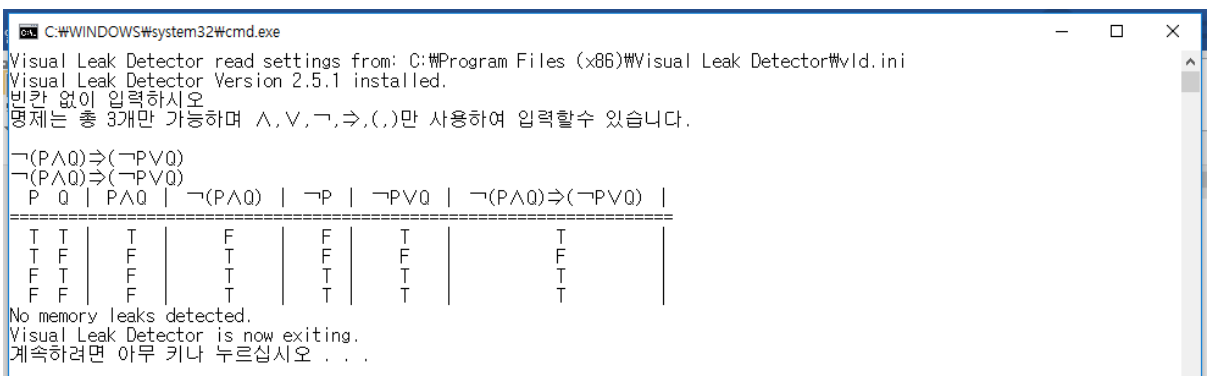
```
C:₩WINDOWS₩system32₩cmd.exe
Visual Leak Detector read settings from: C:₩Program Files (x86)₩Visual Leak Detector₩vld.ini
Visual Leak Detector Version 2.5.1 installed.
빈칸 없이 입력하시오
명제는 총 3개만 가능하며 ∧,∨,¬,⇒,(,)만 사용하여 입력할수 있습니다.

¬(P∧Q)⇒((P∧Q)∨¬R)
¬(P∧Q)⇒((P∧Q)∨¬R)
  P  Q  R | P∧Q | ¬(P∧Q) | P∧Q | ¬R | (P∧Q)∨¬R | ¬(P∧Q)⇒((P∧Q)∨¬R) |
=============================================================================
  T  T  T |   T |     F |   T |  F |       T |                 T |
  T  T  F |   T |     F |   T |  T |       T |                 T |
  T  F  T |   F |     T |   F |  F |       F |                 F |
  T  F  F |   F |     T |   F |  T |       T |                 T |
  F  T  T |   F |     T |   F |  F |       F |                 F |
  F  T  F |   F |     T |   F |  T |       T |                 T |
  F  F  T |   F |     T |   F |  F |       F |                 F |
  F  F  F |   F |     T |   F |  T |       T |                 T |
No memory leaks detected.
Visual Leak Detector is now exiting.
계속하려면 아무 키나 누르십시오 . . .
```



```
C:₩WINDOWS₩system32₩cmd.exe
Visual Leak Detector read settings from: C:₩Program Files (x86)₩Visual Leak Detector₩vld.ini
Visual Leak Detector Version 2.5.1 installed.
빈칸 없이 입력하시오
명제는 총 3개만 가능하며 ∧,∨,¬,⇒,(,)만 사용하여 입력할수 있습니다.

((P∨Q)∧¬R)⇒¬P
((P∨Q)∧¬R)⇒¬P
  P  R | P∨Q | ¬R | (P∨Q)∧¬R | ¬P | ((P∨Q)∧¬R)⇒¬P |
=================================================
  T  T |   T |  F |       F |  F |           T |
  T  F |   T |  T |       T |  F |           F |
  F  T |   F |  F |       F |  T |           T |
  F  F |   F |  T |       F |  T |           T |
No memory leaks detected.
Visual Leak Detector is now exiting.
계속하려면 아무 키나 누르십시오 . . .
```

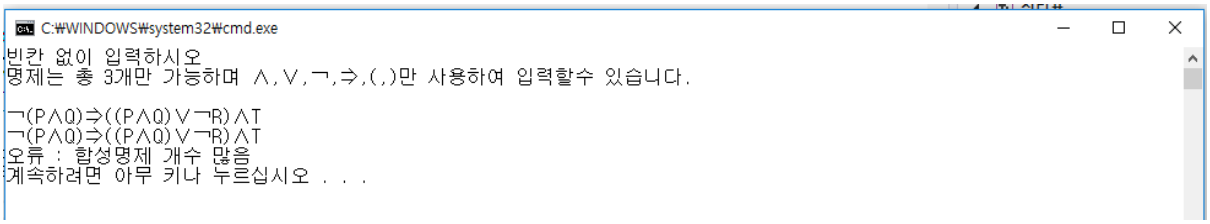


```
C:₩WINDOWS₩system32₩cmd.exe
Visual Leak Detector read settings from: C:₩Program Files (x86)₩Visual Leak Detector₩vld.ini
Visual Leak Detector Version 2.5.1 installed.
빈칸 없이 입력하시오
명제는 총 3개만 가능하며 ∧,∨,¬,⇒,(,)만 사용하여 입력할수 있습니다.

¬(P∧Q)⇒(¬P∨Q)
¬(P∧Q)⇒(¬P∨Q)
  P  Q | P∧Q | ¬(P∧Q) | ¬P | ¬P∨Q | ¬(P∧Q)⇒(¬P∨Q) |
=================================================
  T  T |   T |     F |  F |    T |           T |
  T  F |   F |     T |  F |    F |           F |
  F  T |   F |     T |  T |    T |           T |
  F  F |   F |     T |  T |    T |           T |
No memory leaks detected.
Visual Leak Detector is now exiting.
계속하려면 아무 키나 누르십시오 . . .
```

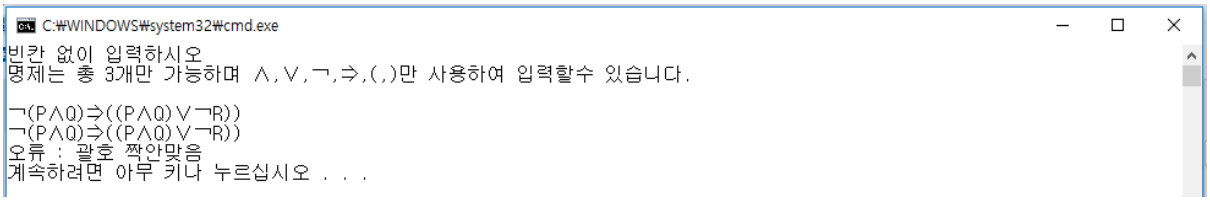
테스트로 직접 합성 명제를 만들어서 다양한 테스트를 하였다 다음과 같이 진리표가 잘 출력되는 것을 볼 수가 있었다.

합성 명제의 개수가 3개를 초과할 경우



명제의 개수가 많음을 알려주는 오류 문구를 출력해 주며 프로그램을 종료한다.

괄호의 짝이 안 맞는 경우



괄호 짝이 안 맞다는 오류를 출력해주며 프로그램을 종료한다.

느낀 점

이번 프로그램을 짜면서 지금까지 겪어보지 못한 많은 예외 사항을 겪을 수가 있었습니다. 이러한 예외 사항을 줄이고 사용자들에게 쫌 더 나은 프로그램을 제공하기 위해 테스트의 과정이 얼마나 중요한지를 깨달았습니다. 앞으로 사용자들이 프로그램을 쓰는데 불편함이 없도록 테스트의 중요성을 높여서 코딩을 하겠습니다.