



소입설  
멘토링 수업  
(9주차)

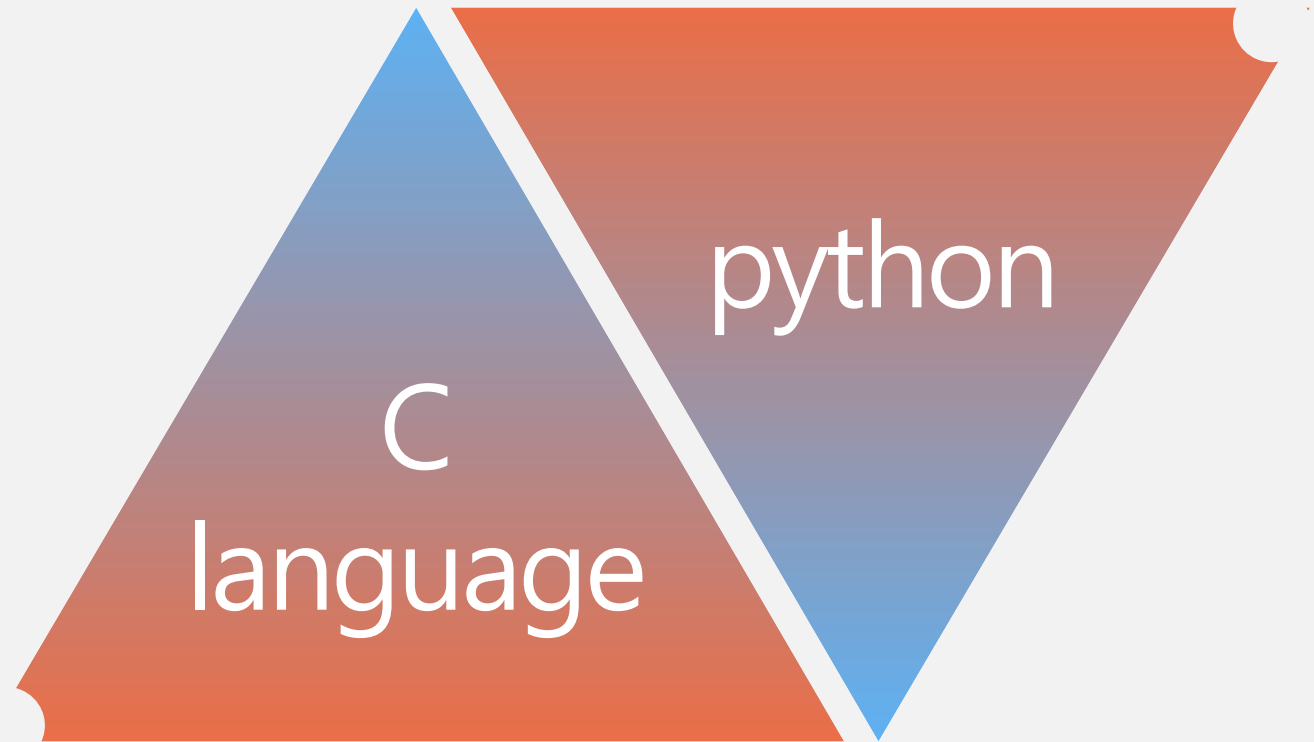


# CONTENTS

---

- 1 C language
  - 2 Windows vs Linux
  - 3 Pointer and Array
-

# 1. C language



1. operators, value, expression, statement (8)
  2. data types (8)
  3. printf, scanf (8)
  4. comment (8)
  5. function (8)
  6. #include (8)
  7. while statement, if statement, for statement (9)
  8. condition and Boolean (9)
  9. #define (9)
-

## 자료형 (data type)

### 1. 정수

- char, short, int, long

### 2. 실수

- double, float

### 3. 배열

- char[], int[], ...

---

## 자료형 (data type)

### 4. 포인터

- `char*`, `int*`, ...

### 5. 구조체

- `struct`

---

## 기본 자료형 종류와 데이터의 표현 범위

자료형(data type)		할당되는 메모리 크기	표현 가능한 데이터의 범위
정수형	char	1 바이트	-128 ~ +127
	short	2 바이트	-32768 ~ +32767
	int	4 바이트	-2147483648 ~ +2147483647
	long	4 바이트	-2147483648 ~ +2147483647
실수형	float	4 바이트	$3.4 \times 10^{-37} \sim 3.4 \times 10^{+38}$
	double	8 바이트	$1.7 \times 10^{-307} \sim 1.7 \times 10^{+308}$
	long double	8 바이트 혹은 그 이상	차이를 많이 보임

## 서식 문자의 종류와 그 의미

서식 문자	출력 형태
<b>%c</b>	단일 문자
<b>%d</b>	부호 있는 10진 정수
<b>%i</b>	부호 있는 10진 정수, %d와 같음
<b>%f</b>	부호 있는 10진 실수
<b>%s</b>	문자열
<b>%o</b>	부호 없는 8진 정수
<b>%u</b>	부호 없는 10진 정수
<b>%x, %X</b>	부호 없는 16진 정수, 소,대문자 사용
<b>%p, %P</b>	주소값(16진 정수), 소,대문자 사용
<b>%e</b>	e 표기법에 의한 실수
<b>%E</b>	E 표기법에 의한 실수
<b>%g</b>	값에 따라서 %f, %e 둘 중 하나를 선택
<b>%G</b>	값에 따라서 %f, %E 둘 중 하나를 선택
<b>%%</b>	% 기호 출력



```
// function
```

```
#include <stdio.h>
```

```
int function(int);
```

```
int main(void) {  
    int num,result;  
    scanf("%d",&num);  
    result=function(num);  
    printf("result: %d",result);  
    return 0;  
}
```

```
int function (int a){  
    int temp;  
    printf("function call\n");  
    scanf("%d",&temp);  
    return a+temp;  
}
```

```
// #include
```

```
#include <stdio.h>
```

```
int function(int);
```

```
int main(void) {  
    int num,result;  
    scanf("%d",&num);  
    result=function(num);  
    printf("result: %d",result);  
    return 0;  
}
```

```
int function (int a){  
    int temp;  
    printf("function call\n");  
    scanf("%d",&temp);  
    return a+temp;  
}
```

```
// #include
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int function(int);
```

```
int main(void) {  
    int num,result;  
    scanf("%d",&num);  
    result=function(num);  
    printf("result: %d",result);  
    return 0;  
}
```

```
int function (int a){  
    int temp;  
    printf("function call\n");  
    temp=rand()%10;  
    return a+temp;  
}
```

```
// #include
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <windows.h>
```

```
int function(int);
```

```
int main(void) {
```

```
    int num,result;
```

```
    scanf("%d",&num);
```

```
    result=function(num);
```

```
    printf("result: %d",result);
```

```
    return 0;
```

```
}
```

```
int function (int a){
```

```
    int temp;
```

```
    printf("function call\n");
```

```
    temp=rand()%10;
```

```
    Sleep(3000);
```

```
    return a+temp;
```

```
}
```

```
// while loop

#include<stdio.h>

int main(){
    int count=0,exit=3;
    while(count<exit){
        function();
        count++;
    }
    return 0;
}
```

```
void function (){
    printf("function call\n");
}
```

```
// if-else statement
```

```
#include<stdio.h>
```

```
int main(){  
    int count=0,exit=3;  
    if(count<exit){  
        function();  
        count++;  
    } else {  
        printf("0>=3\n");  
    }  
    return 0;  
}
```

```
void function (){  
    printf("function call\n");  
}
```

```
// for loop
```

```
#include<stdio.h>
```

```
int main(){  
    int i,exit=3;  
    for(i=0;i<exit;i++){  
        function();  
    }  
    return 0;  
}
```

```
void function (){  
    printf("function call\n");  
}
```

```
// #define
```

```
#include<stdio.h>
```

```
#define TRUE 1
```

```
#define FALSE 0
```

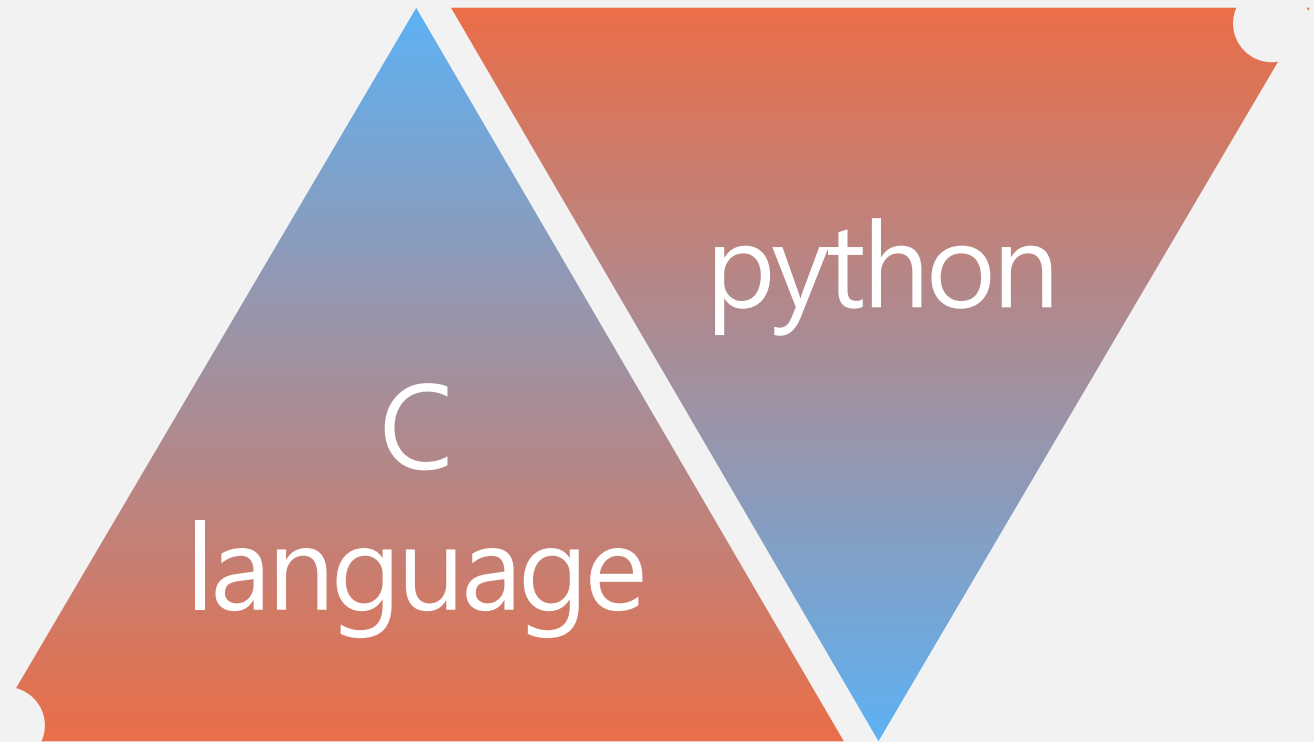
```
int main(){  
    if(TRUE) function1();  
    if(FALSE) function2();  
    return 0;  
}
```

```
void function1 (){  
    printf("It`s TRUE\n");  
}
```

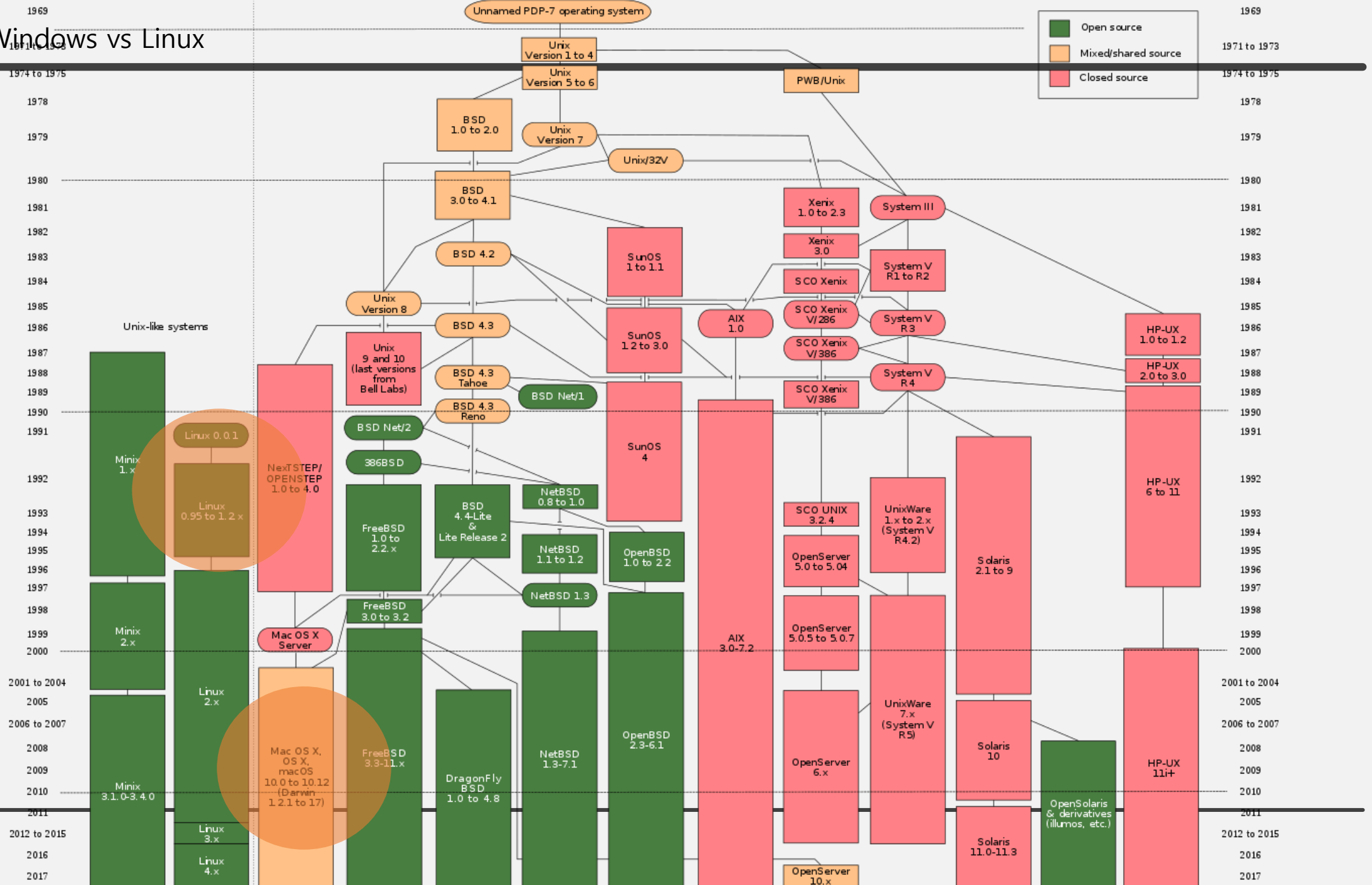
```
void function2 (){  
    printf("It`s FALSE!\n");  
}
```



## 2. Windows vs Linux



## 2 Windows vs Linux





Linux

The diagram consists of three orange circles arranged horizontally. The first circle on the left contains the word 'Linux'. The middle circle contains the word 'Windows'. The third circle on the right contains the word 'macOS'. The circles are evenly spaced and have a solid orange fill.

Windows

macOS

---



**Ubuntu**

redhat

centOS

---

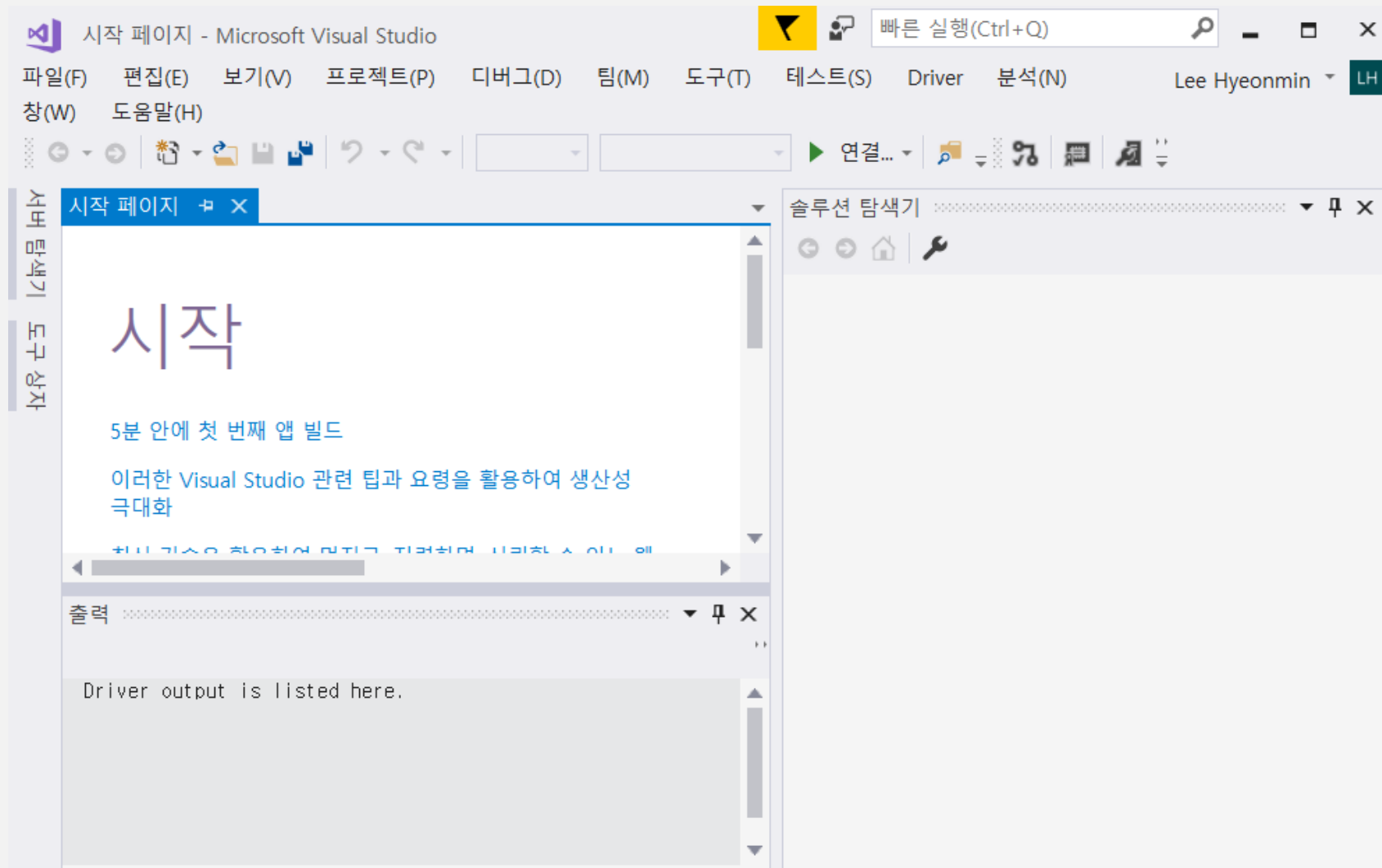
## 2 Windows vs Linux

---

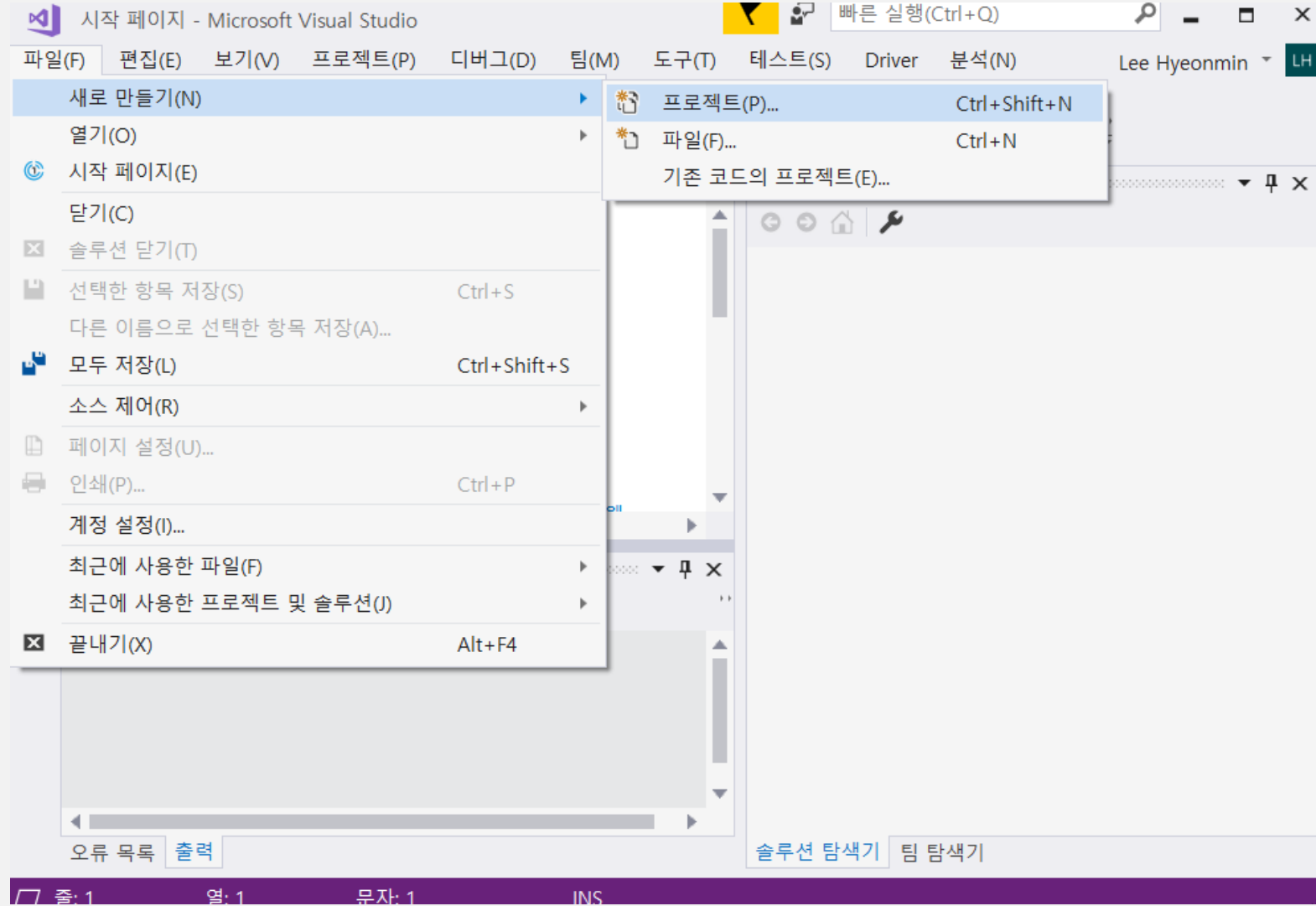
비교	Windows	Linux
커널 소스코드	비공개	공개
점유율	82%	2%
사용 용도	거의 모든 업무	서버유지, 개발
속도	느림	빠름
용량	큼	작음
커널 수정	거의 불가능	가능
C compiler	Visual Studio 내장 컴파일러	gcc
User Interface	GUI	거의 CUI

---

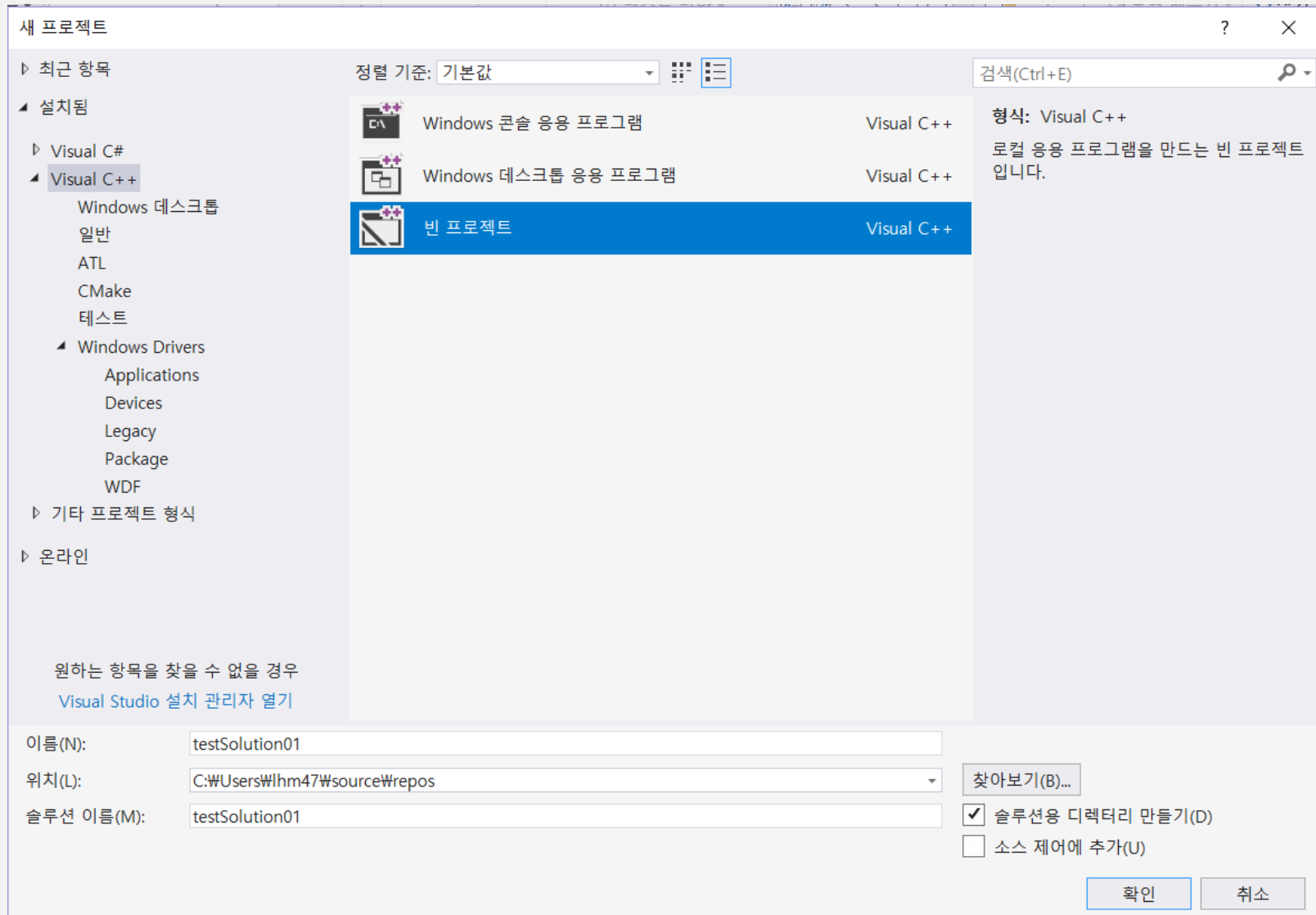
## 2 Windows vs Linux



## 2 Windows vs Linux

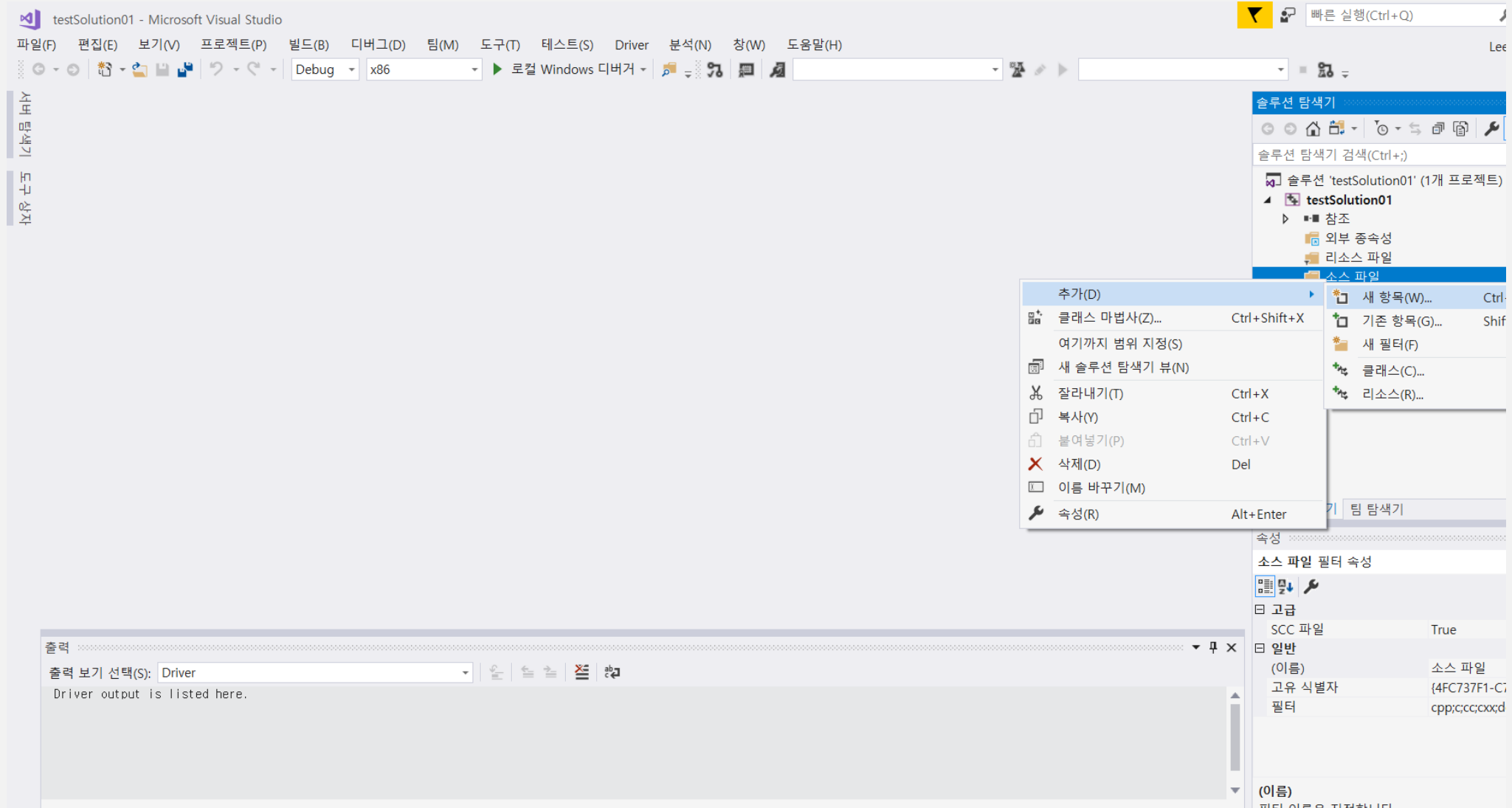


## 2 Windows vs Linux

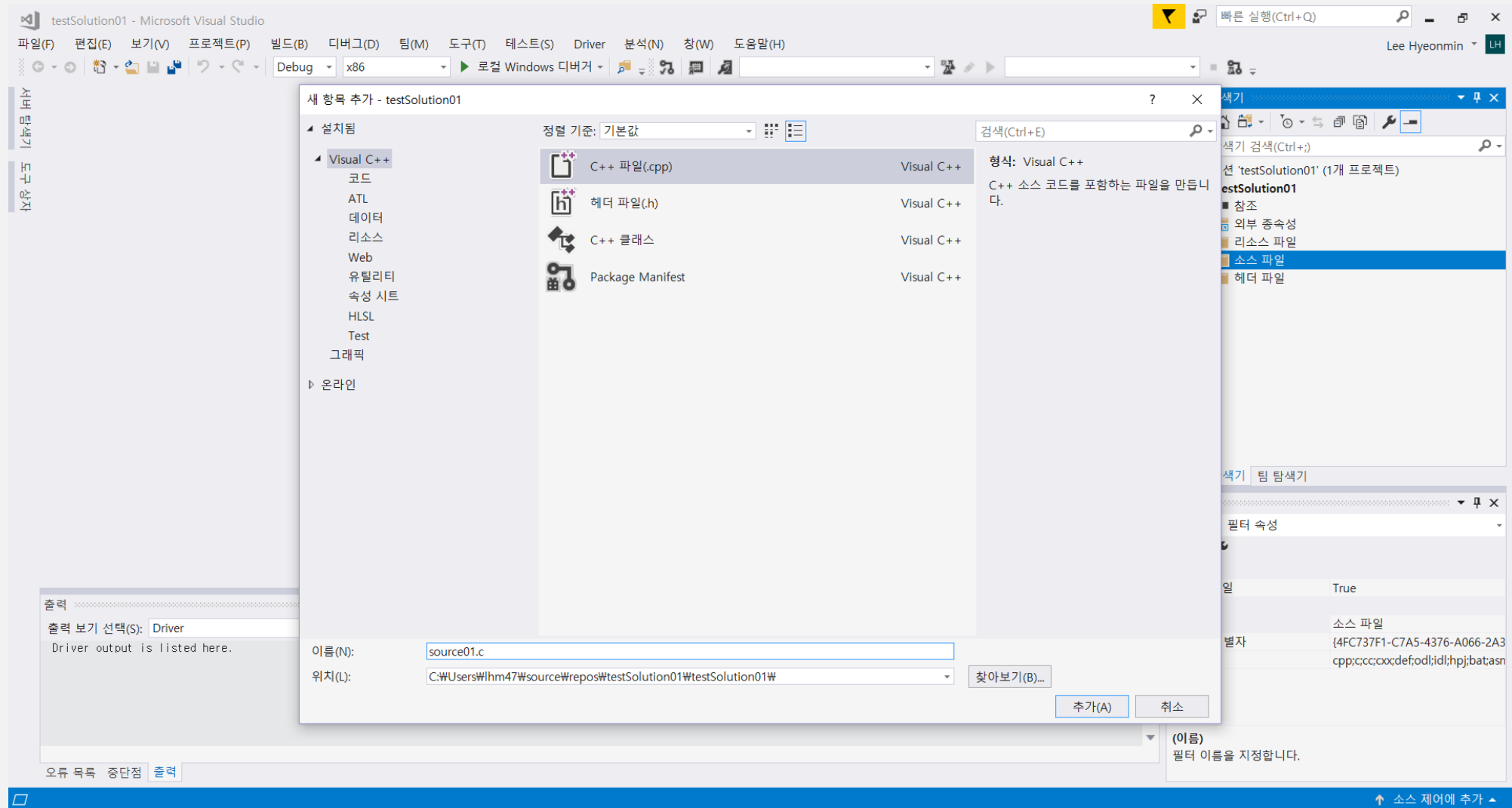




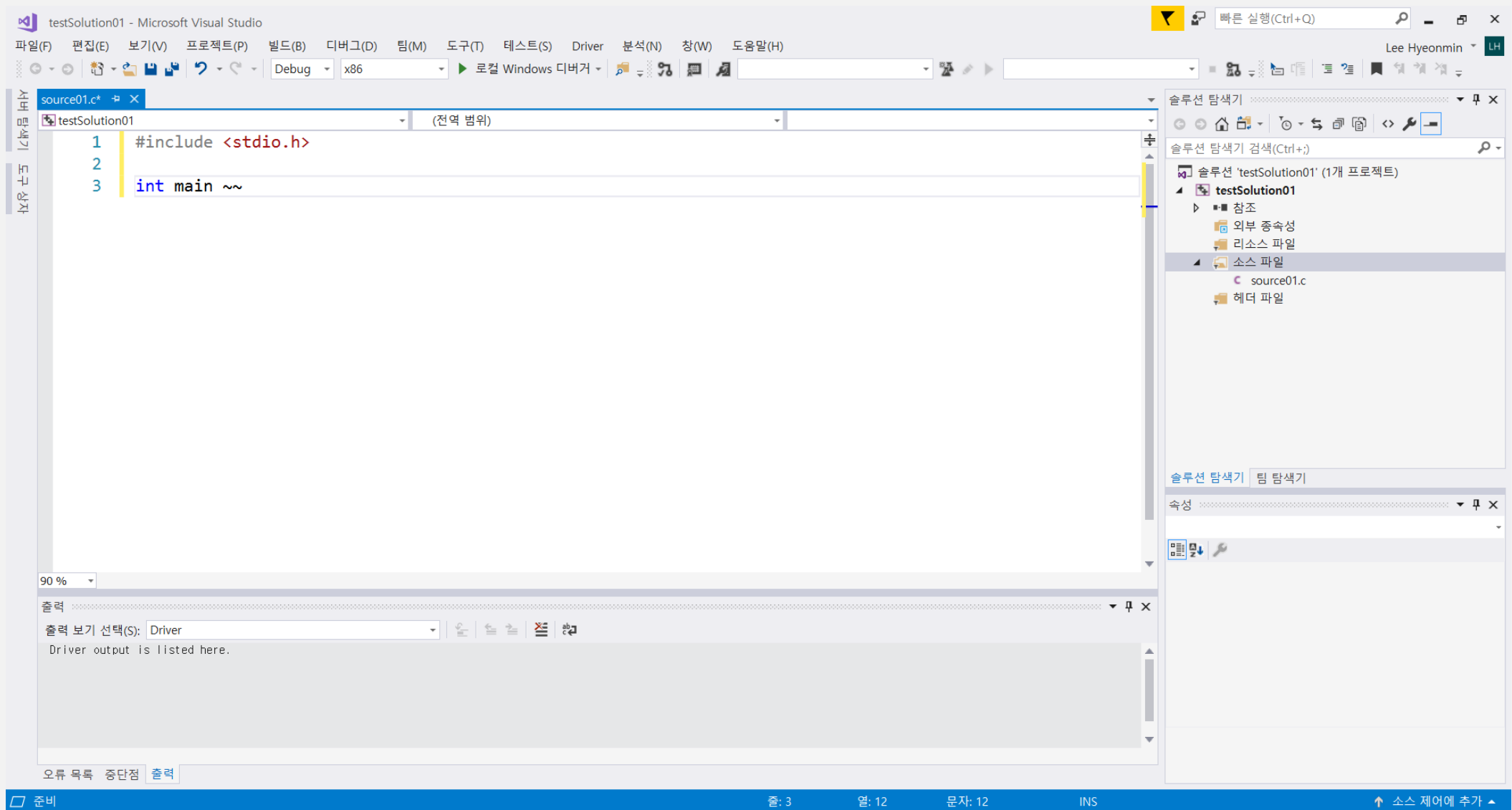
## 2 Windows vs Linux



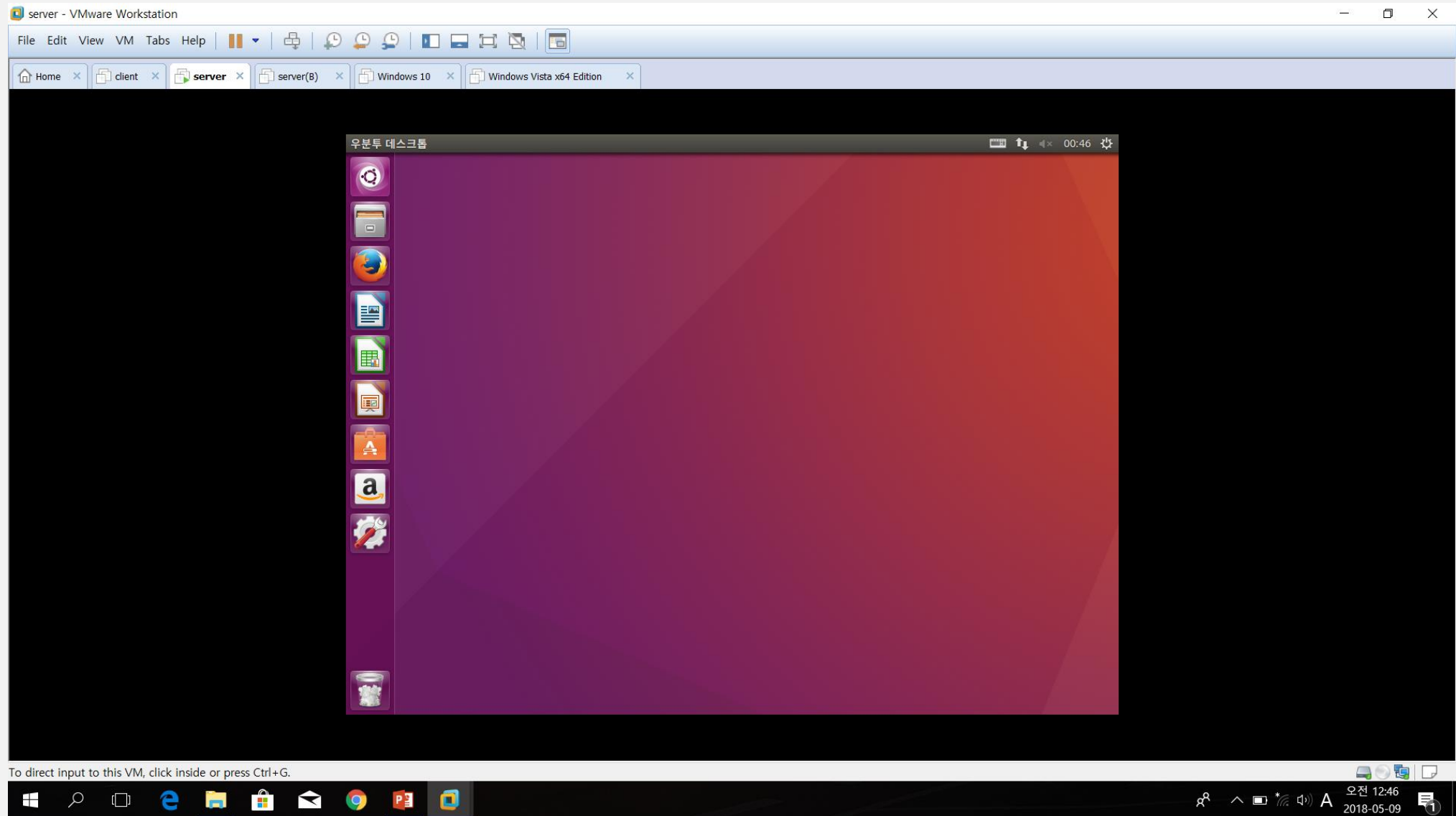
## 2 Windows vs Linux



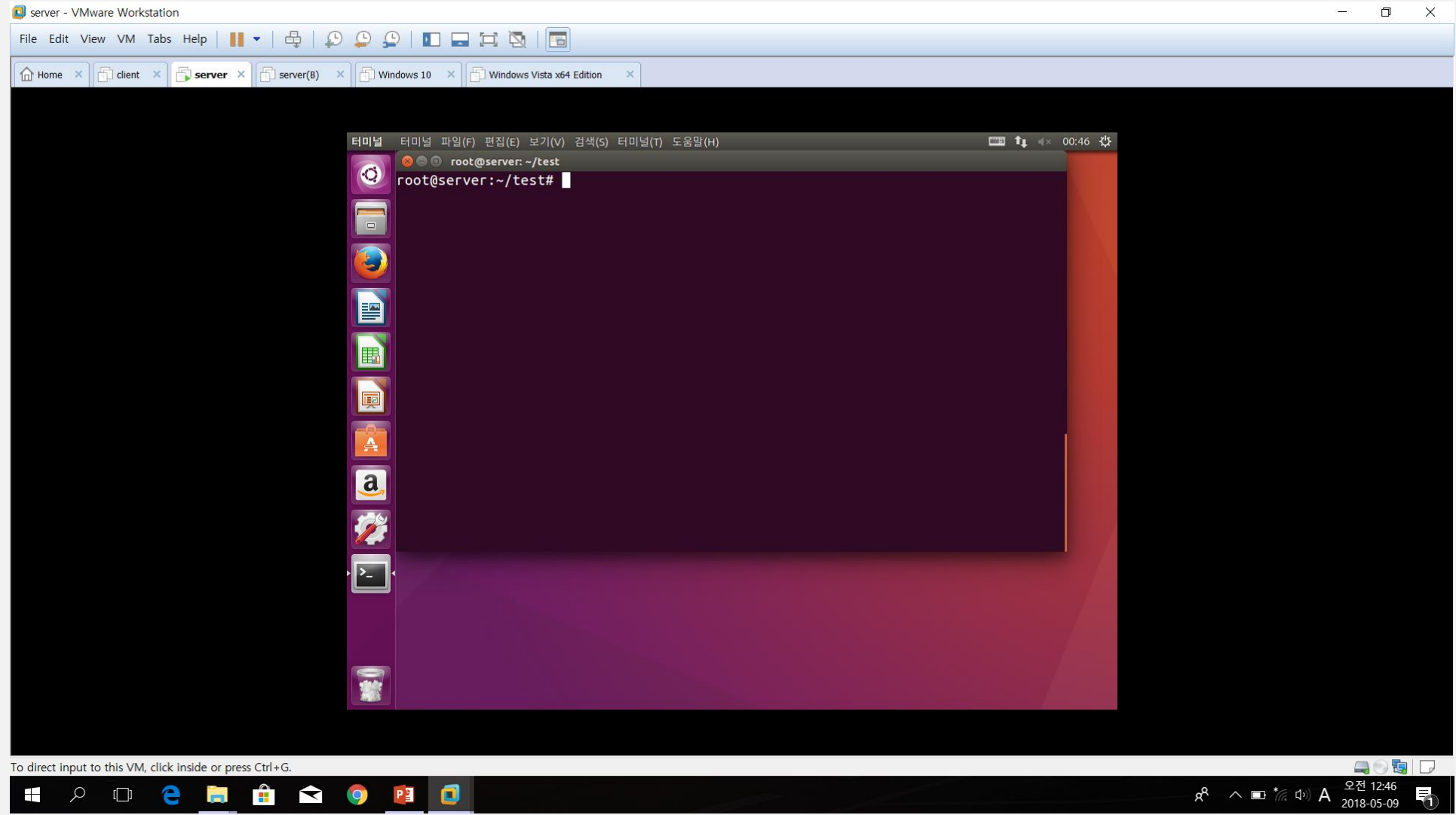
## 2 Windows vs Linux



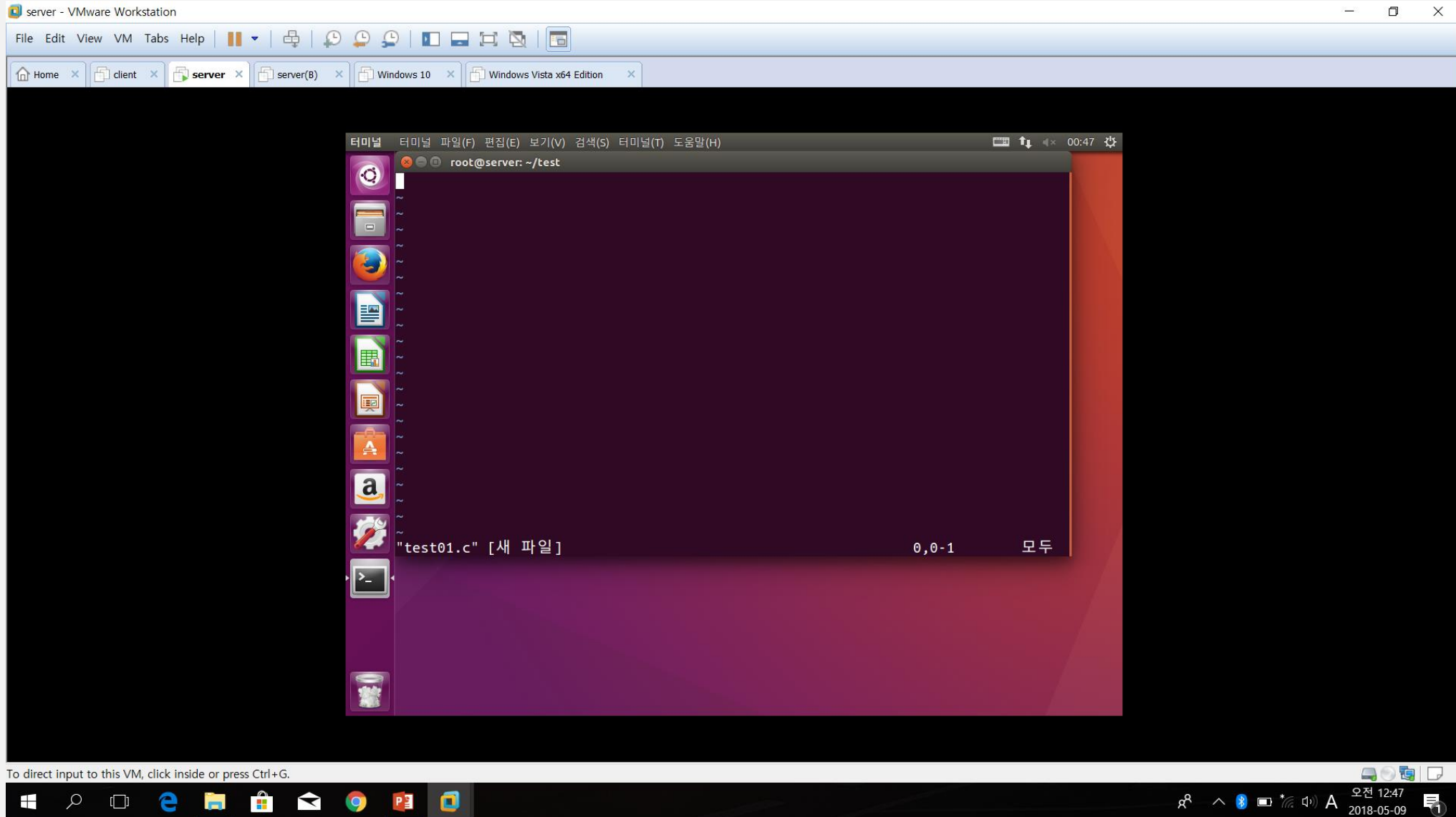
## 2 Windows vs Linux



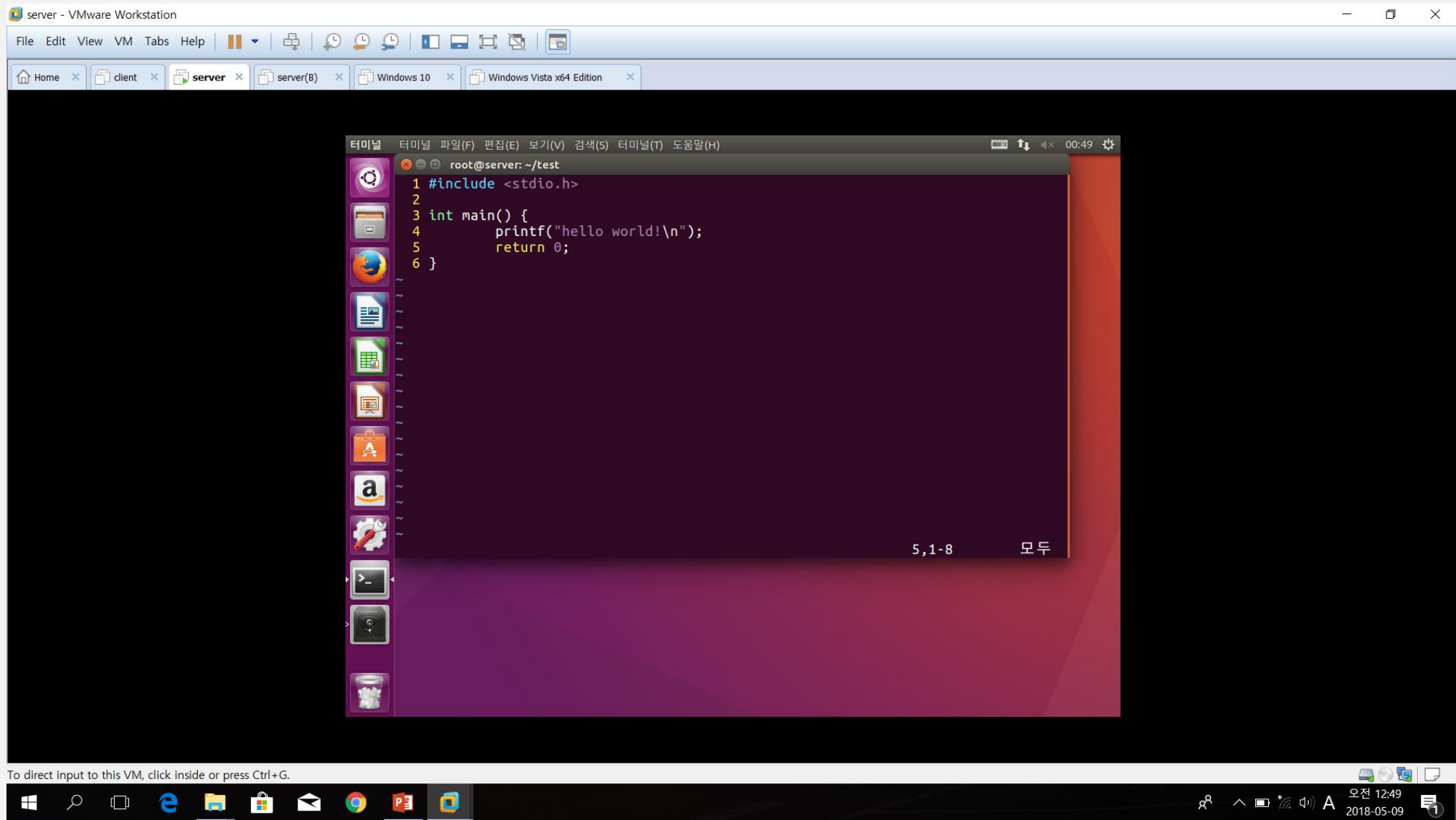
## 2 Windows vs Linux



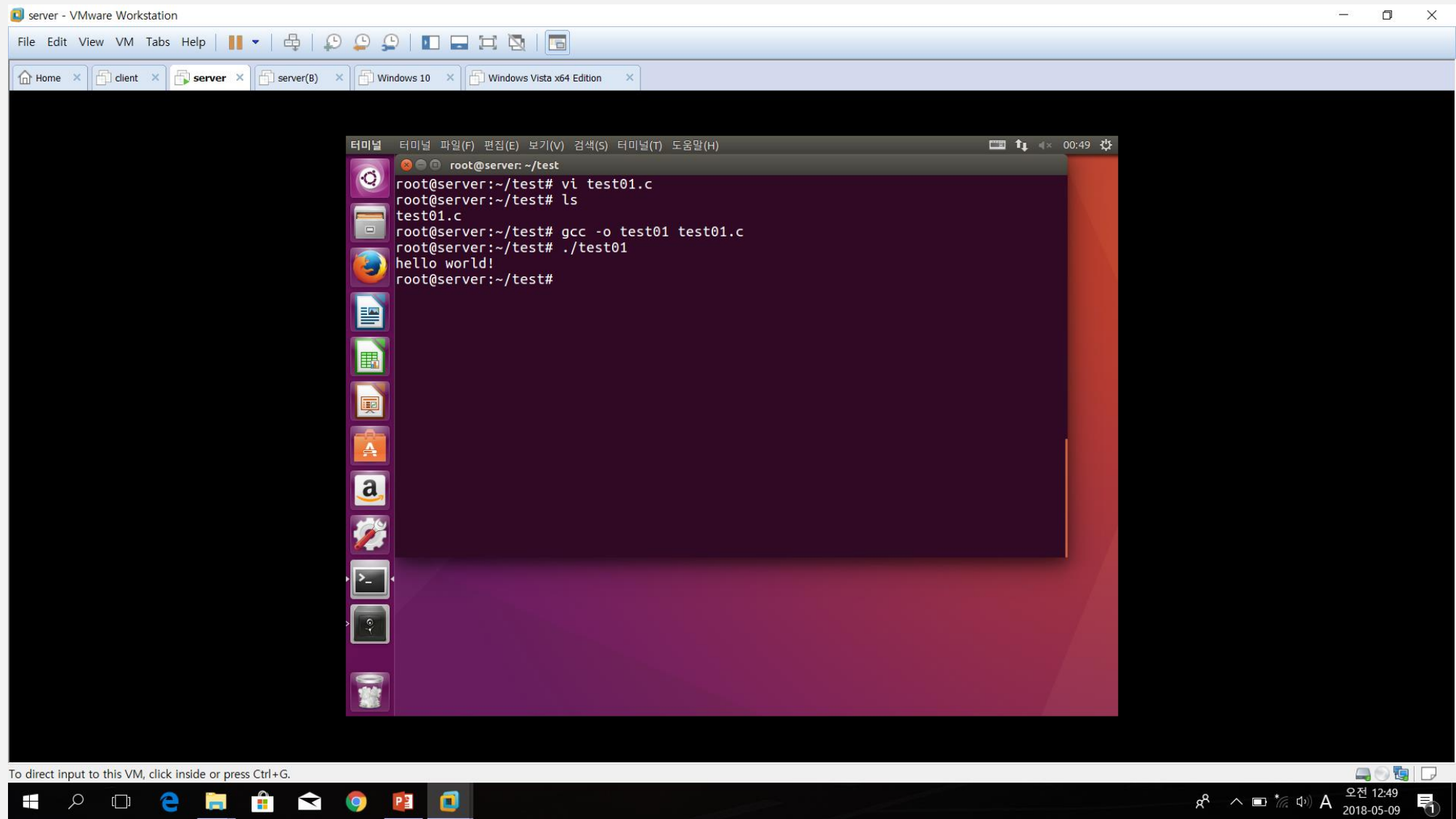
## 2 Windows vs Linux



## 2 Windows vs Linux

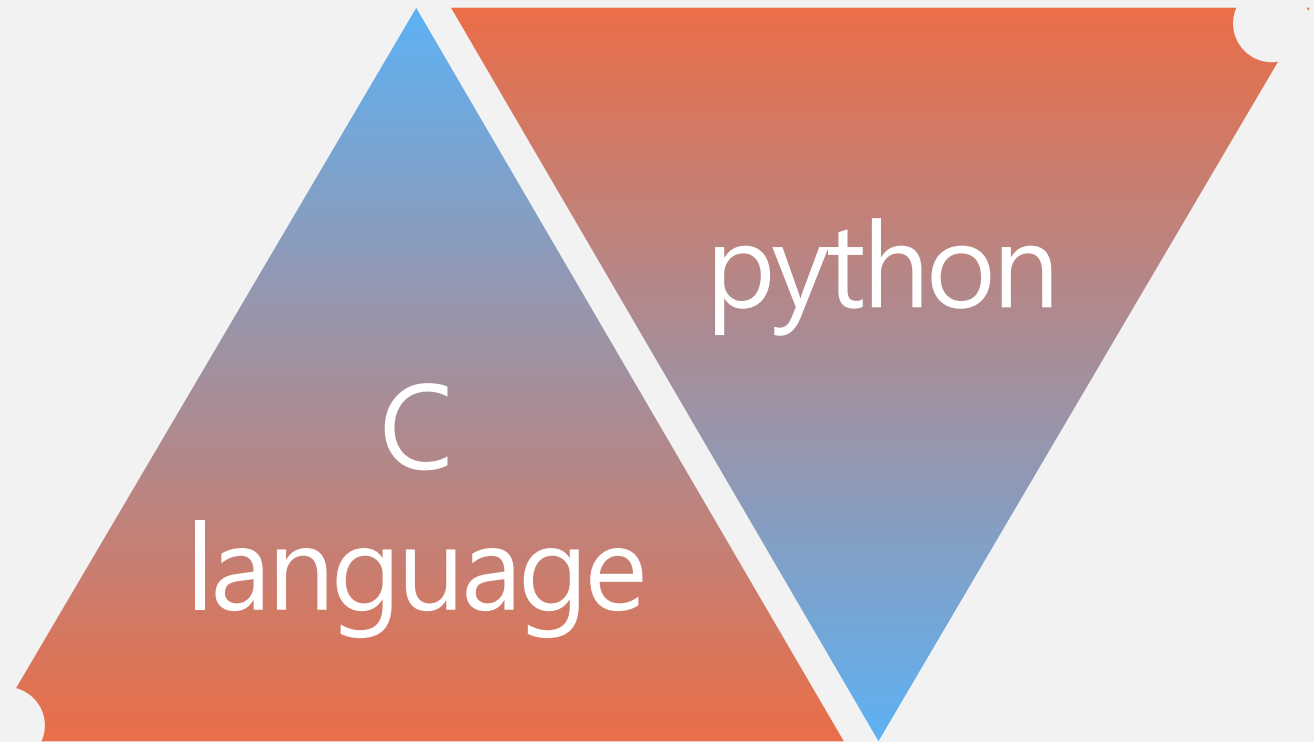


## 2 Windows vs Linux





### 3. Pointer and Array



Memory!!!

---

1 Byte = 8 bits  
00000000  
~  
11111111

<b>0xF014</b>	CC
<b>0xF015</b>	CC
<b>0xF016</b>	CC
<b>0xF017</b>	CC
<b>0xF018</b>	CC
<b>0xF019</b>	CC
<b>0xF020</b>	CC
<b>0xF021</b>	CC
<b>0xF022</b>	CC
<b>0xF023</b>	CC
<b>0xF024</b>	CC
<b>0xF025</b>	CC
<b>0xF026</b>	CC
<b>0xF027</b>	CC
<b>0xF028</b>	CC

---

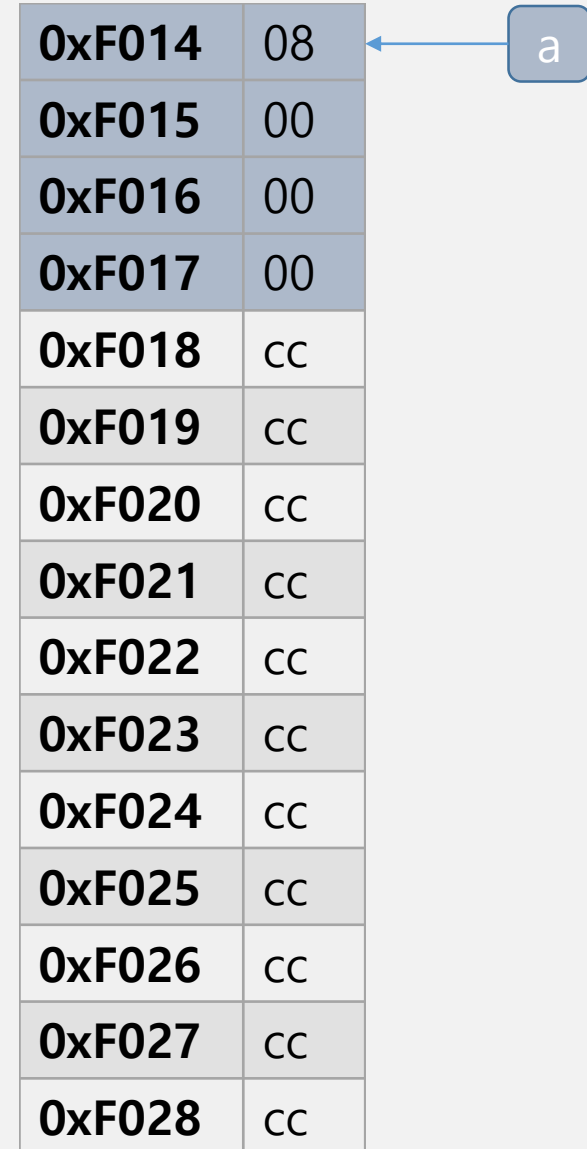
`int a;`

int: 4Bytes(32bits)

<b>0xF014</b>	00
<b>0xF015</b>	00
<b>0xF016</b>	00
<b>0xF017</b>	00
<b>0xF018</b>	CC
<b>0xF019</b>	CC
<b>0xF020</b>	CC
<b>0xF021</b>	CC
<b>0xF022</b>	CC
<b>0xF023</b>	CC
<b>0xF024</b>	CC
<b>0xF025</b>	CC
<b>0xF026</b>	CC
<b>0xF027</b>	CC
<b>0xF028</b>	CC

a


```
int a;  
  
a=8;
```



A diagram illustrating the memory layout of a variable 'a'. A blue box labeled 'a' has a blue arrow pointing to the first row of a table. The table has two columns: the first column contains hexadecimal addresses from 0xF014 to 0xF028, and the second column contains the corresponding memory values. The first four rows (0xF014 to 0xF017) are highlighted in blue, indicating they are part of the variable 'a'. The remaining rows (0xF018 to 0xF028) are in white with gray borders.

0xF014	08
0xF015	00
0xF016	00
0xF017	00
0xF018	CC
0xF019	CC
0xF020	CC
0xF021	CC
0xF022	CC
0xF023	CC
0xF024	CC
0xF025	CC
0xF026	CC
0xF027	CC
0xF028	CC

```
int a;  
  
a=8;  
  
a=0x12345678;
```



A blue box containing the letter 'a' has a blue arrow pointing to the first row of the memory table (0xF014).

<b>0xF014</b>	78
<b>0xF015</b>	56
<b>0xF016</b>	34
<b>0xF017</b>	12
<b>0xF018</b>	CC
<b>0xF019</b>	CC
<b>0xF020</b>	CC
<b>0xF021</b>	CC
<b>0xF022</b>	CC
<b>0xF023</b>	CC
<b>0xF024</b>	CC
<b>0xF025</b>	CC
<b>0xF026</b>	CC
<b>0xF027</b>	CC
<b>0xF028</b>	CC

```
int a;  
a=0x12345678;  
int* pa;
```

0xF014	78	← a
0xF015	56	
0xF016	34	
0xF017	12	
0xF018	cc	← pa
0xF019	cc	
0xF020	cc	
0xF021	00	
0xF022	00	
0xF023	00	
0xF024	00	
0xF025	00	
0xF026	00	
0xF027	00	
0xF028	00	

```
int a;  
a=0x12345678;  
int* pa;  
pa=&a;
```

0xF014	78	← a
0xF015	56	
0xF016	34	
0xF017	12	
0xF018	cc	← pa
0xF019	cc	
0xF020	cc	
0xF021	14	
0xF022	F0	
0xF023	00	
0xF024	00	
0xF025	00	
0xF026	00	
0xF027	00	
0xF028	00	



1 Byte = 8 bits  
00000000  
~  
11111111

<b>0xF014</b>	CC
<b>0xF015</b>	CC
<b>0xF016</b>	CC
<b>0xF017</b>	CC
<b>0xF018</b>	CC
<b>0xF019</b>	CC
<b>0xF020</b>	CC
<b>0xF021</b>	CC
<b>0xF022</b>	CC
<b>0xF023</b>	CC
<b>0xF024</b>	CC
<b>0xF025</b>	CC
<b>0xF026</b>	CC
<b>0xF027</b>	CC
<b>0xF028</b>	CC

---

`char ch=0x30;`

0xF014	CC	
0xF015	CC	
0xF016	CC	
0xF017	CC	
0xF018	CC	
0xF019	CC	
0xF020	CC	
0xF021	CC	
0xF022	CC	
0xF023	CC	
0xF024	CC	
0xF025	CC	
0xF026	30	← ch
0xF027	CC	
0xF028	CC	

```
char ch=0x30;  
char* pch=&ch;
```

0xF014	cc	
0xF015	26	← pch
0xF016	F0	
0xF017	00	
0xF018	00	
0xF019	00	
0xF020	00	
0xF021	00	
0xF022	00	
0xF023	cc	
0xF024	cc	
0xF025	cc	
0xF026	30	← ch
0xF027	cc	
0xF028	cc	

1 Byte = 8 bits  
00000000  
~  
11111111

<b>0xF014</b>	CC
<b>0xF015</b>	CC
<b>0xF016</b>	CC
<b>0xF017</b>	CC
<b>0xF018</b>	CC
<b>0xF019</b>	CC
<b>0xF020</b>	CC
<b>0xF021</b>	CC
<b>0xF022</b>	CC
<b>0xF023</b>	CC
<b>0xF024</b>	CC
<b>0xF025</b>	CC
<b>0xF026</b>	CC
<b>0xF027</b>	CC
<b>0xF028</b>	CC

---

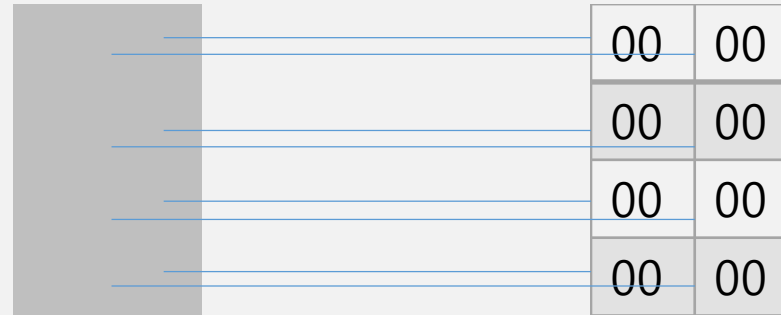
Double \*da,  
\*db;

0xF014	00	
0xF015	00	← da
0xF016	00	
0xF017	00	
0xF018	00	
0xF019	00	
0xF020	00	
0xF021	00	
0xF022	00	
0xF023	cc	
0xF024	cc	
0xF025	cc	
0xF026	00	← db
0xF027	00	
0xF028	00	↓

why?

CPU

RAM



$2^3$  Bytes

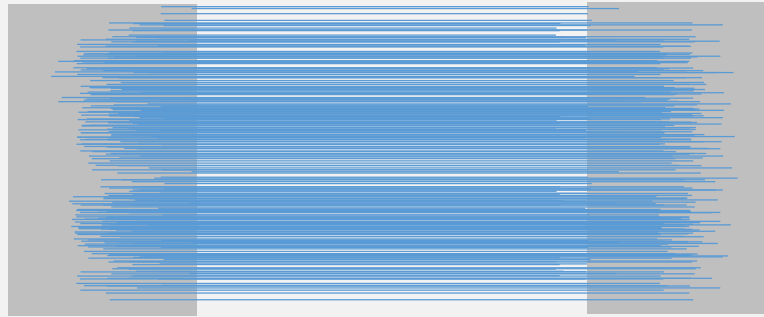
연결통로  $2^3$  개 필요

---

why?

CPU

RAM

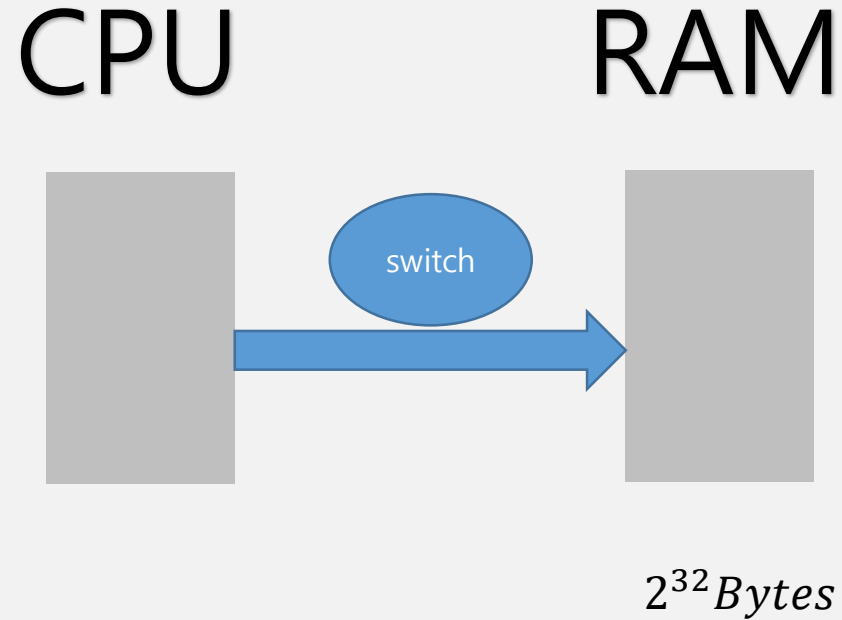


$2^{32}$  Bytes

연결통로  $2^{32}$  개 필요

---

why?



➔ 연결통로 32 개 필요

---



# CPU와 메모리의 원활한 소통

---

# 사용 예)

# 포인터 변수 선언

int\* pa;    ok!

int \*pa;    ok!

# point

\*address → 메모리에서  
          특정 주소 칸에  
          담긴 정보

#address

&variable → 메모리에서  
          특정 변수의 주소값

```
#include <stdio.h>
```

```
int function(int*,int*);
```

```
int main(void) {
```

```
    int num1=3, num2=5;
```

```
    function(&num1,&num2);
```

```
    printf("%d, %d\n",num1,num2);
```

```
    return 0;
```

```
}
```

```
void function(int* pa, int* pb){
```

```
    int temp=*pa;
```

```
    *pa=*pb;
```


```
    *pb=temp;
```

```
}
```

Array?

---

`char ch=0x30;`

<b>0xF014</b>	30	
<b>0xF015</b>	CC	
<b>0xF016</b>	CC	
<b>0xF017</b>	CC	
<b>0xF018</b>	CC	
<b>0xF019</b>	CC	
<b>0xF020</b>	CC	
<b>0xF021</b>	CC	
<b>0xF022</b>	CC	
<b>0xF023</b>	CC	
<b>0xF024</b>	CC	
<b>0xF025</b>	CC	
<b>0xF026</b>	CC	
<b>0xF027</b>	CC	
<b>0xF028</b>	CC	

```
char ch=0x30;
```

```
char cArr[10];
```

0xF014	30	← ch
0xF015	cc	
0xF016	00	← cArr
0xF017	00	
0xF018	00	
0xF019	00	
0xF020	00	
0xF021	00	
0xF022	00	
0xF023	00	
0xF024	00	
0xF025	00	
0xF026	cc	
0xF027	cc	
0xF028	cc	

```
printf("%x",ch);  
ch=*(0xF014);
```

```
printf("%p",cArr);  
cArr=0xF016;  
cArr[0]=*(0XF016);
```

0xF014	30	← ch
0xF015	cc	
0xF016	00	← cArr
0xF017	00	
0xF018	00	
0xF019	00	
0xF020	00	
0xF021	00	
0xF022	00	
0xF023	00	
0xF024	00	
0xF025	00	
0xF026	cc	
0xF027	cc	
0xF028	cc	

---

복습

---