SIEC: BASIC C PROGRAMMING

L #06: C STORAGE CLASSES

Seung Beop Lee

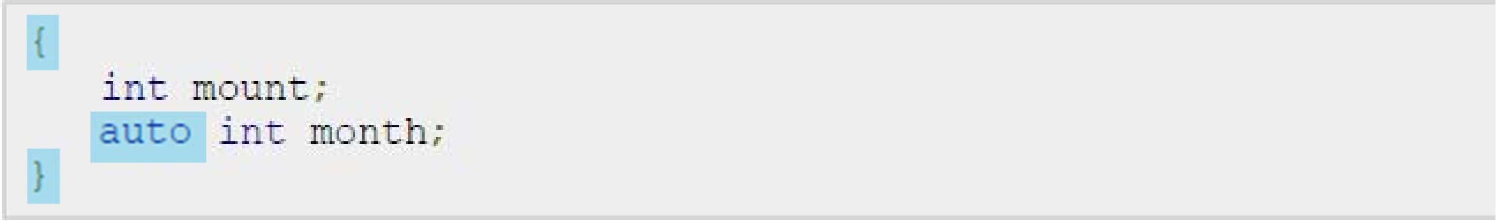
School of International Engineering and Science

CHONBUK NATIONAL UNIVERSITY

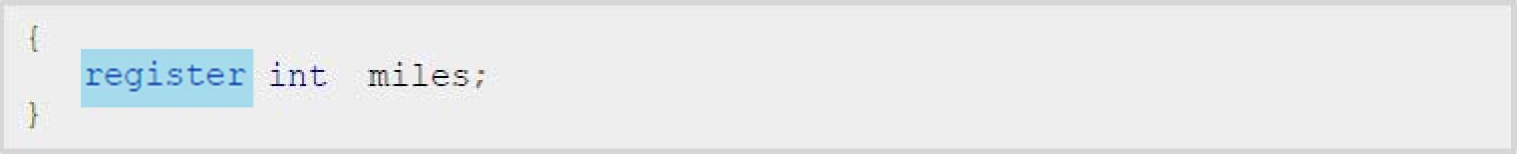
Storage classes

Storage classes

- Storage classes
  - A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program.
  - There are the four storage classes, which can be used in a C Program.
    - auto
    - register
    - extern
    - static
  - These specifiers precede the type that they modify.



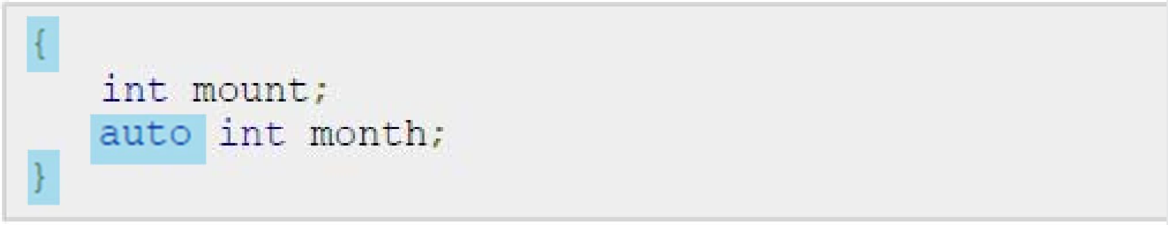
```
{
    int mount;
    auto int month;
}
```



```
{
   register int  miles;
}
```

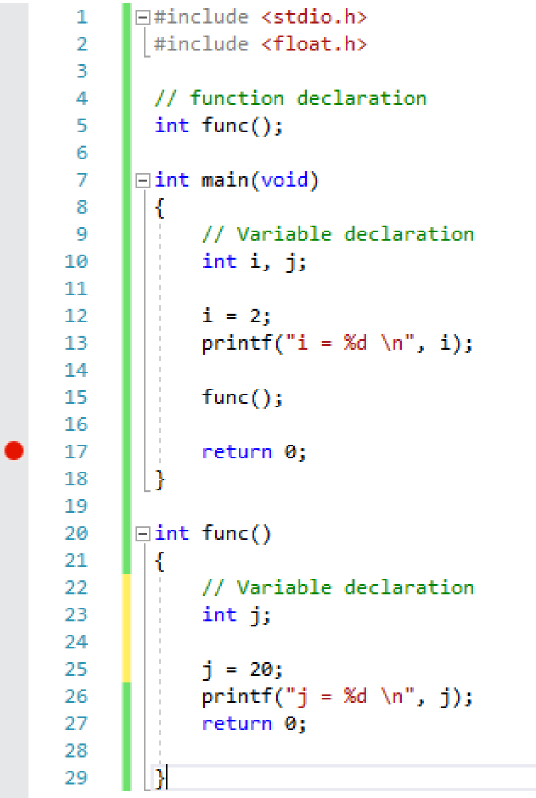
Storage classes

- The auto storage classes
  - The auto storage class is the default storage class for all local variables.
  - The following example defines two variables with the same storage class, auto can only be used within functions, i.e., inside their braces { }.
  - Variables of this kind are called local variables. Another name for them is auto(or automatic) variables.



```
{
    int mount;
    auto int month;
}
```

- The scope of auto (i.e. local) variables is only within functions, i.e., inside their braces { }.
- The life-time of auto (i.e. local) variables is only useful within functions, i.e., inside their braces { }.

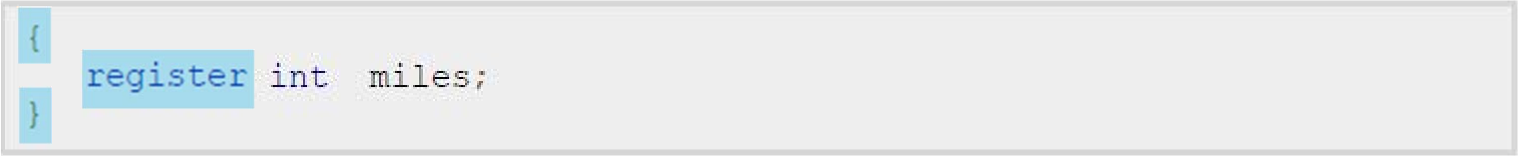


```
#include <stdio.h>
#include <float.h>

// function declaration
int func();

int main(void)
{
    // Variable declaration
    int i, j;

    i = 2;
    printf("i = %d \n", i);

    func();

    return 0;
}

int func()
{
    // Variable declaration
    int j;

    j = 20;
    printf("j = %d \n", j);
    return 0;

}
```

Storage classes

- The register storage classes
  - The register is a storage in the CPU (Central Processing Unit).
  - The register is faster in computation than RAM (Random Access Memory).
  - The register is very important and expensive.
  - So, the register should only be used for variables that require quick access and have a short life-time.

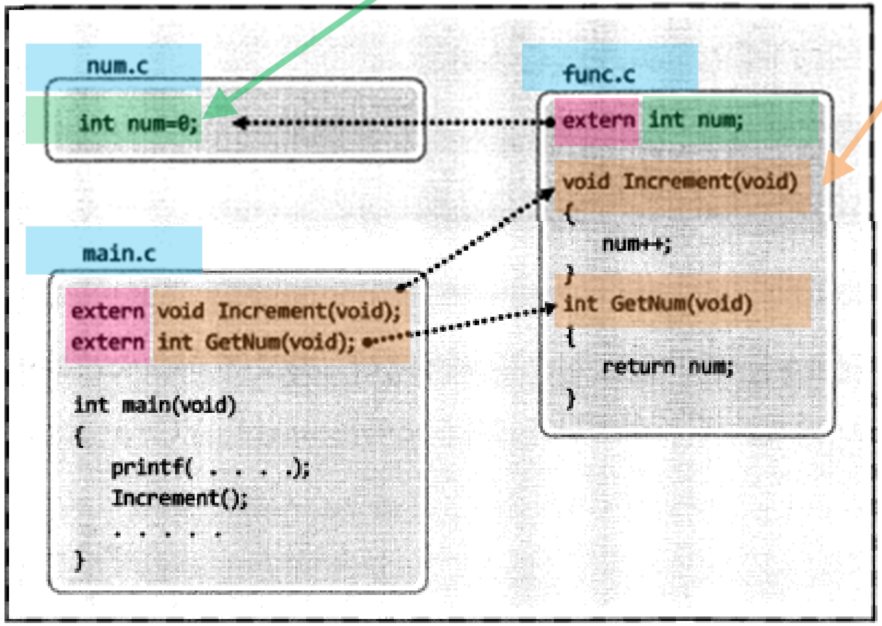
  - The register storage class is used to define local variables that should be stored in a register instead of RAM.



```
{
    register int  miles;
}
```

Storage classes

- The extern storage classes
  - The extern storage class is used to give a reference of a global variable that is visible to ALL the program files.
  - When you have multiple files and you define a global variable or function, which will be used in other files also, then extern will be used in another file to give reference of defined variable or function.
  - The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions.



num.c

```
int num=0;
```

func.c

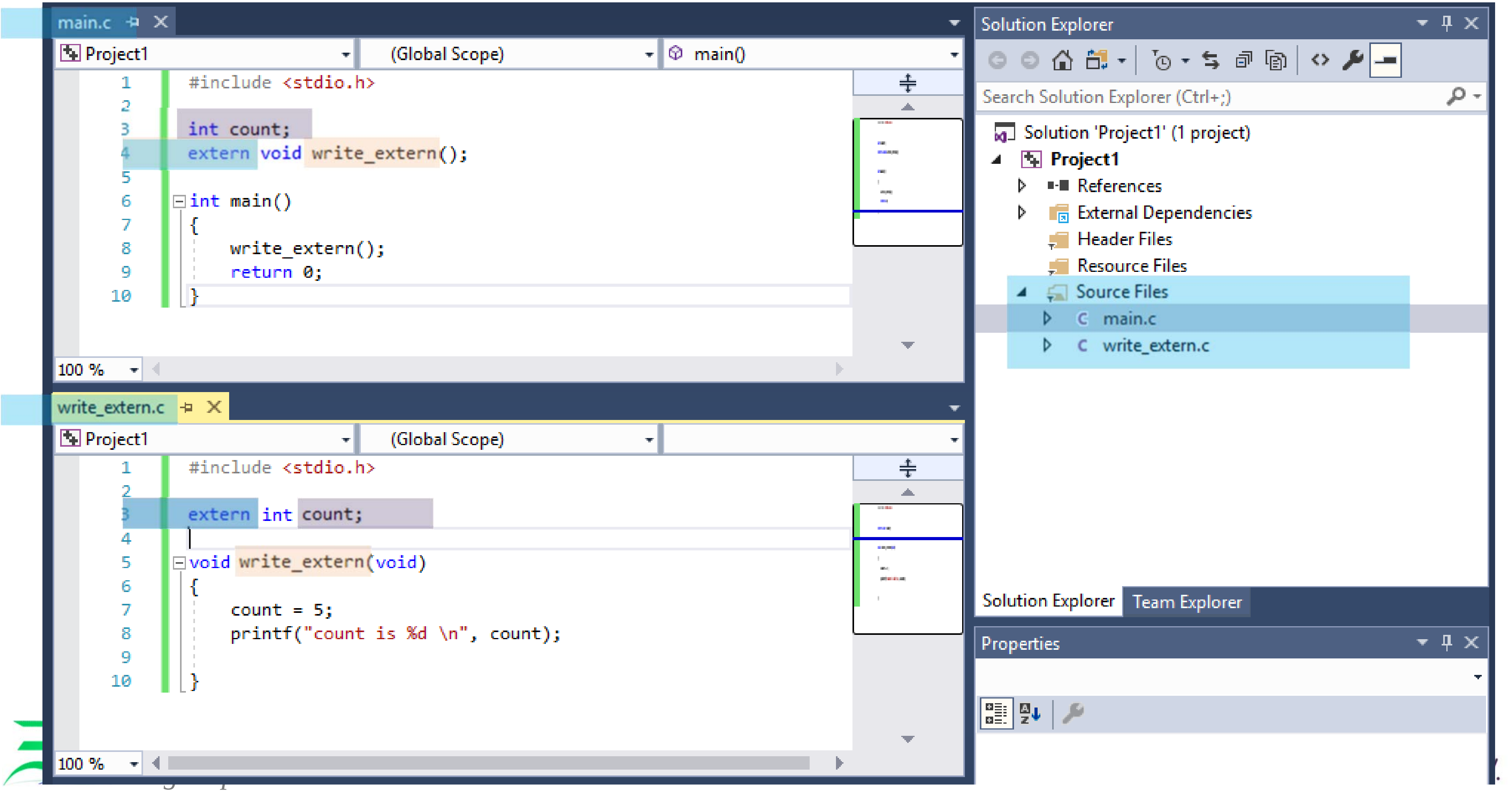
```
extern int num;

void Increment(void)
{
    num++;
}
int GetNum(void)
{
    return num;
}
```

main.c

```
extern void Increment(void);
extern int GetNum(void);

int main(void)
{
    printf( . . . .);
    Increment();
    . . . . .
}
```

Storage classes

- The extern storage classes
  - Just for understanding, extern is used to declare a global variable or function in another file as explained below.

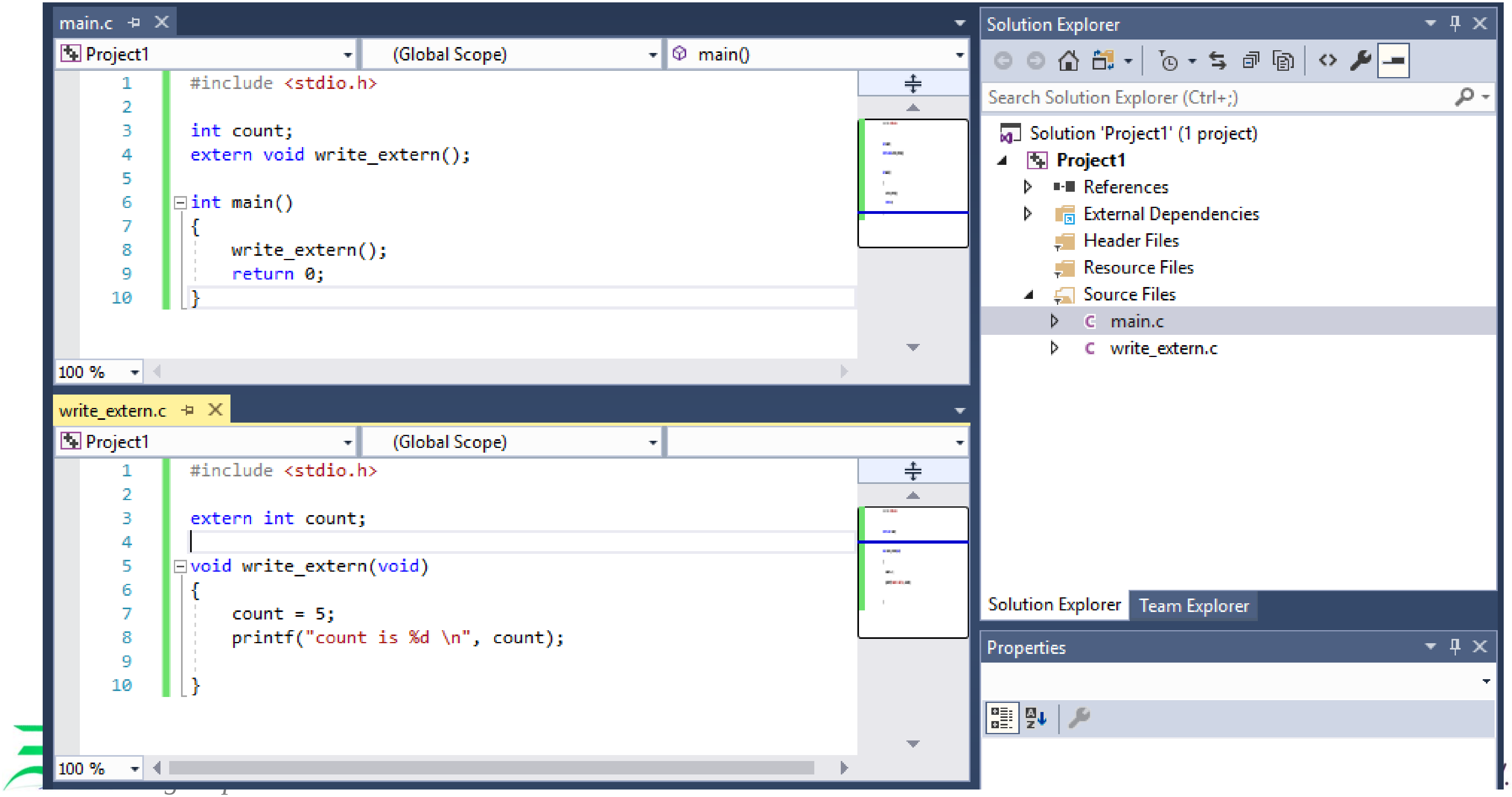


```
#include <stdio.h>

int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Storage classes

- The extern storage classes
  - Just for understanding, extern is used to declare a global variable or function in another file as explained below.

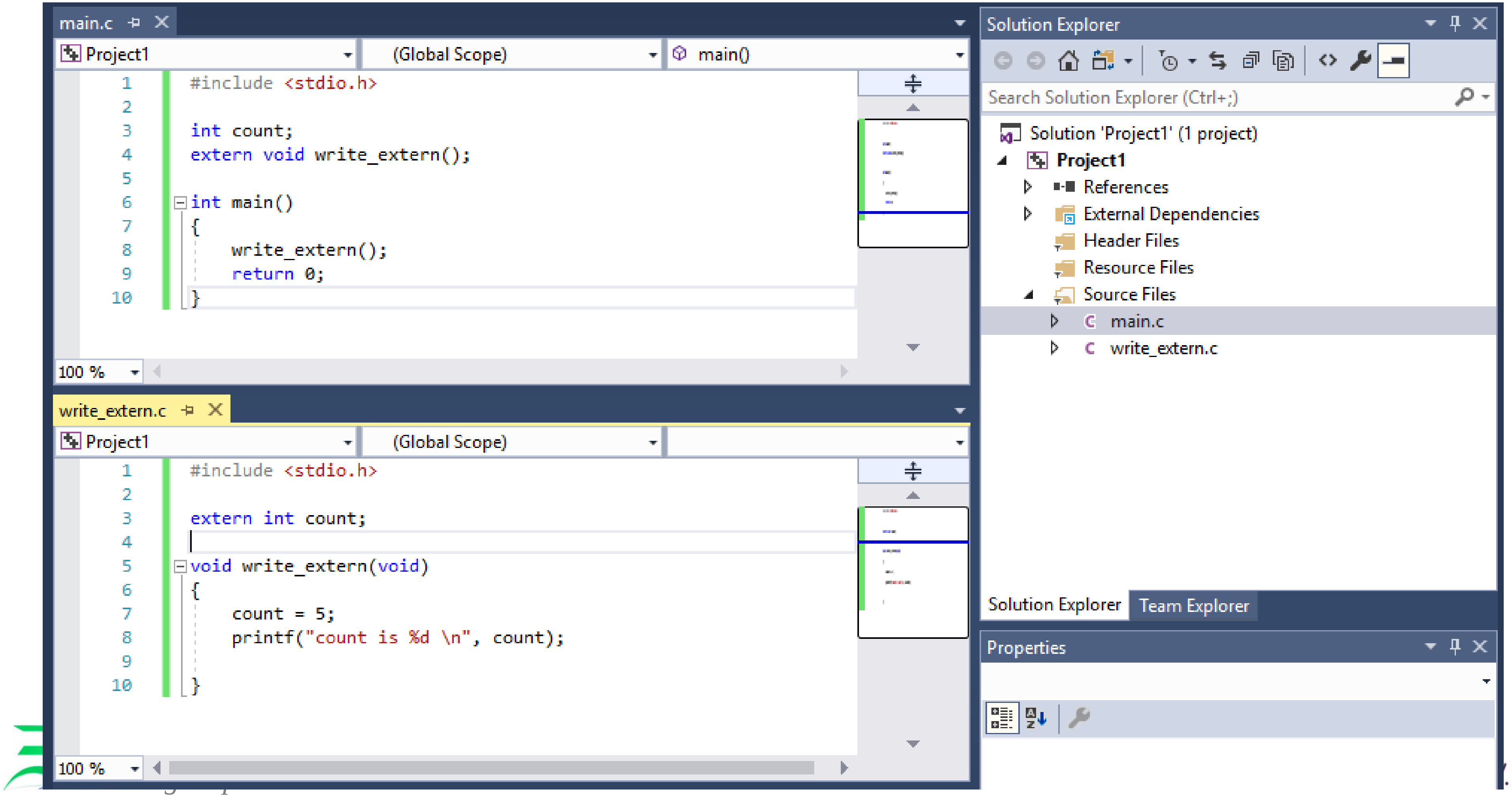


```c
#include <stdio.h>

int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

```c
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);

}
```

Storage classes

- The extern storage classes
  - Just for understanding, extern is used to declare a global variable or function in another file as explained below.



main.c

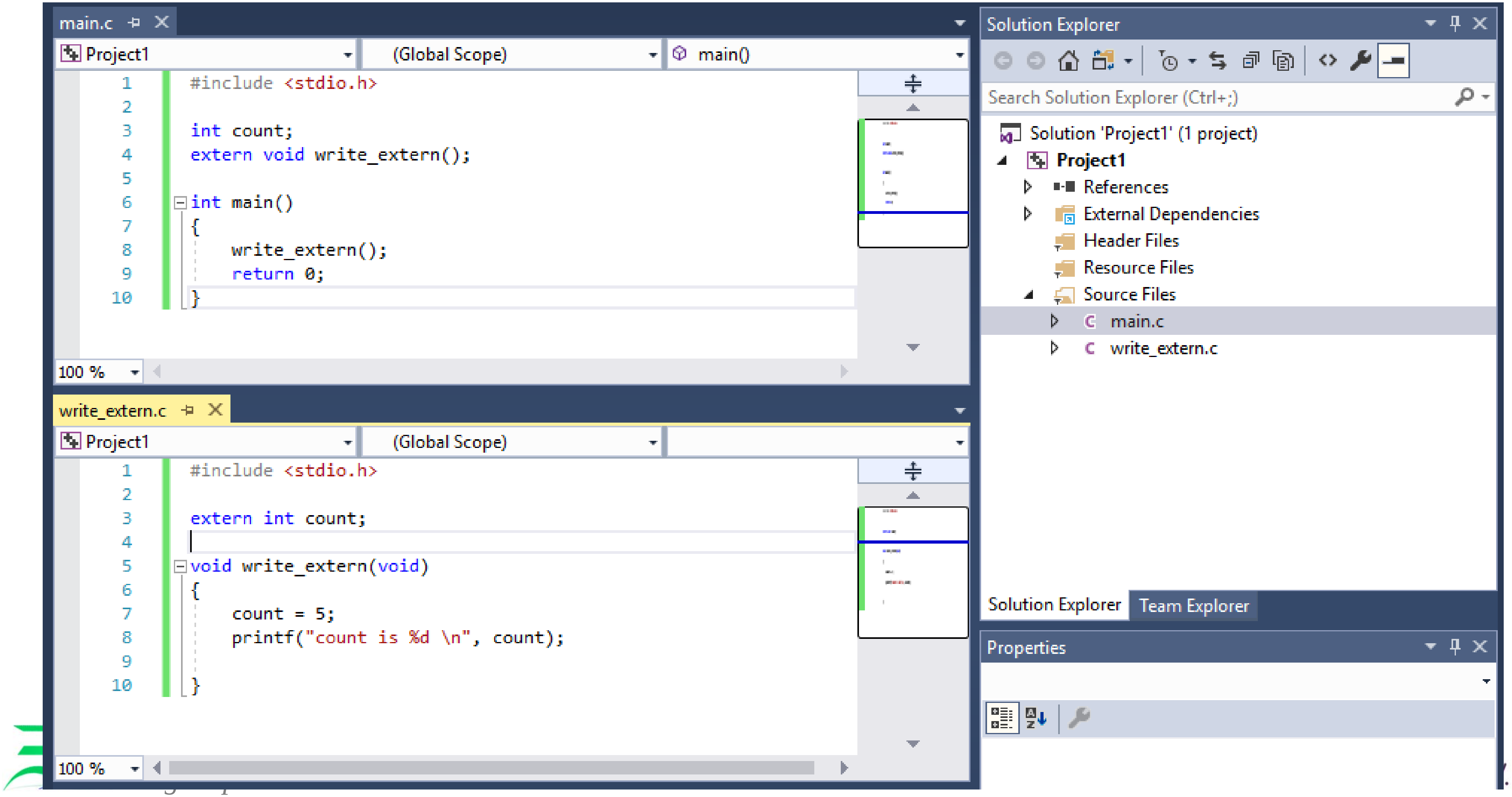
```c
#include <stdio.h>

int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

write_extern.c

```c
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Storage classes

- The extern storage classes
  - Just for understanding, extern is used to declare a global variable or function in another file as explained below.



main.c

```c
#include <stdio.h>

int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

write_extern.c

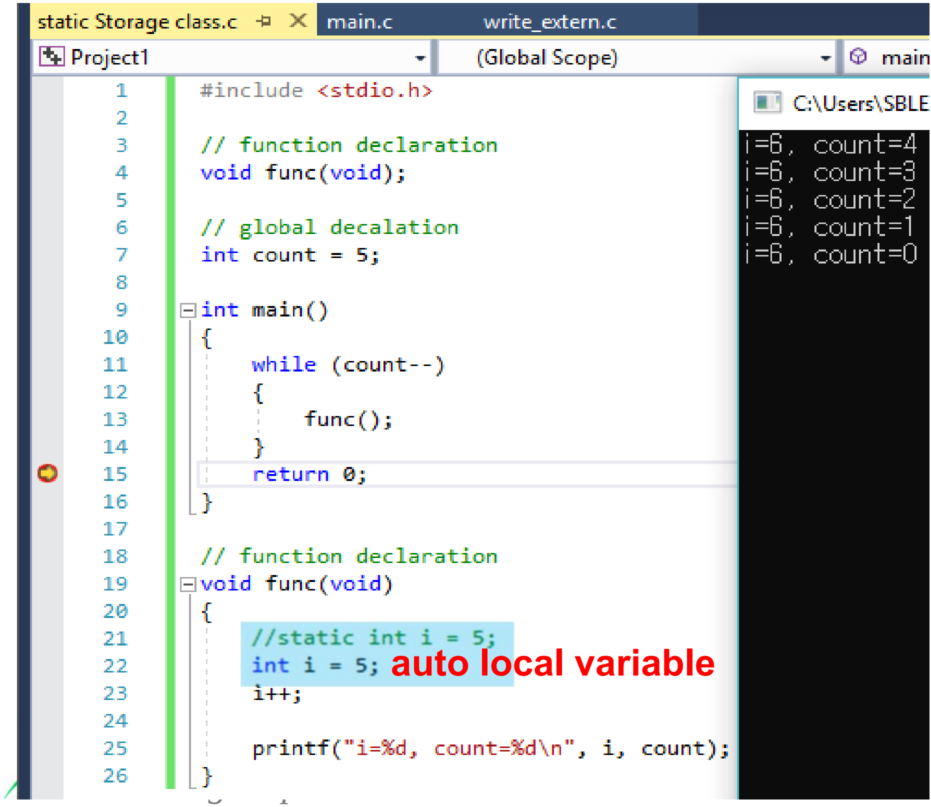
```c
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);

}
```

Storage classes

- The static storage classes
  - **Local variable**
    - The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying the local variable each time it comes into and goes out of scope, i.e., their braces { }.
    - Therefore, making local variables static allows them to maintain their values between function calls.
  - **Global variable**
    - When the static modifier is applied to global variables, it causes that variable's scope to be restricted to the file (with extension **.c**) in which it is declared.
    - It means that another file doesn't call the global variables with the static modifier through the extern modifier.

Storage classes

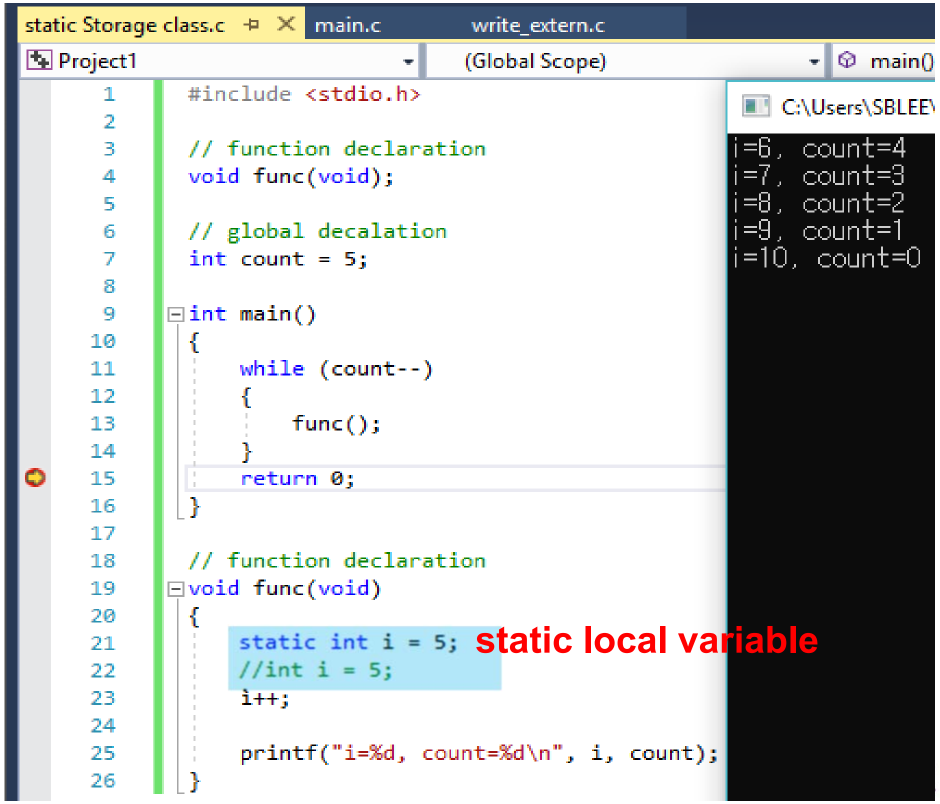
- The static storage classes
  - **Local variable**
    - ➢ The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying the local variable <u>each time it comes into and goes out of scope, i.e., their braces { }.</u>



```
#include <stdio.h>

// function declaration
void func(void);

// global decalation
int count = 5;

int main()
{
    while (count--)
    {
        func();
    }
    return 0;
}

// function declaration
void func(void)
{
    //static int i = 5;
    int i = 5;
    i++;

    printf("i=%d, count=%d\n", i, count);
}
```

```
i=6, count=4
i=6, count=3
i=6, count=2
i=6, count=1
i=6, count=0
```

auto local variable



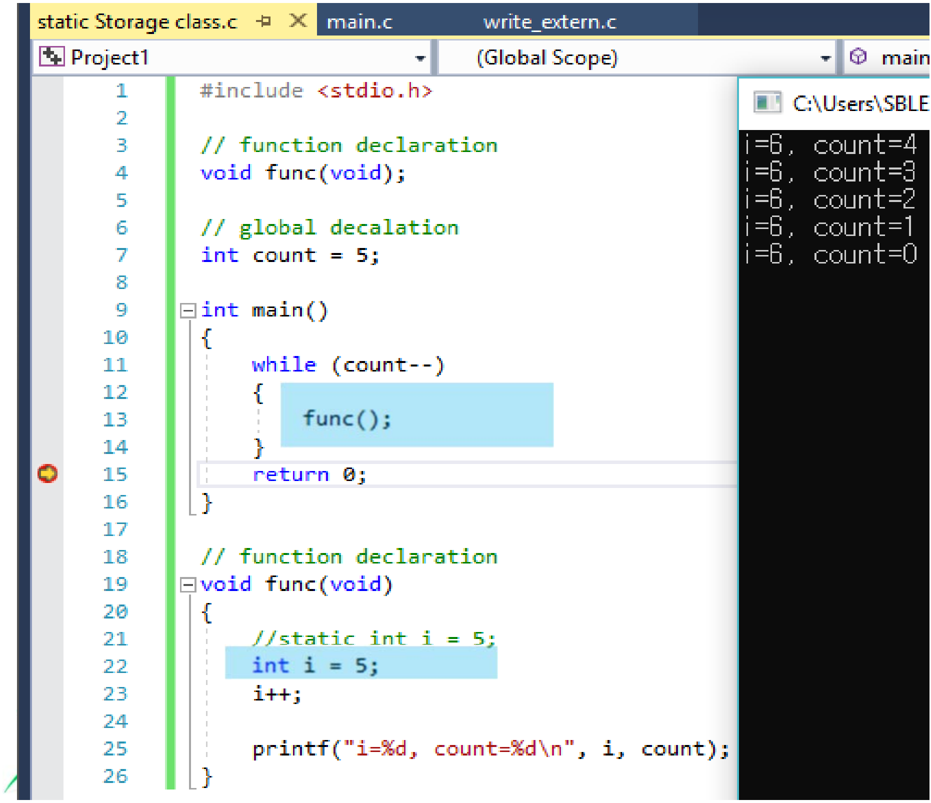
```
#include <stdio.h>

// function declaration
void func(void);

// global decalation
int count = 5;

int main()
{
    while (count--)
    {
        func();
    }
    return 0;
}

// function declaration
void func(void)
{
    static int i = 5;
    //int i = 5;
    i++;

    printf("i=%d, count=%d\n", i, count);
}
```

```
i=6, count=4
i=7, count=3
i=8, count=2
i=9, count=1
i=10, count=0
```

static local variable

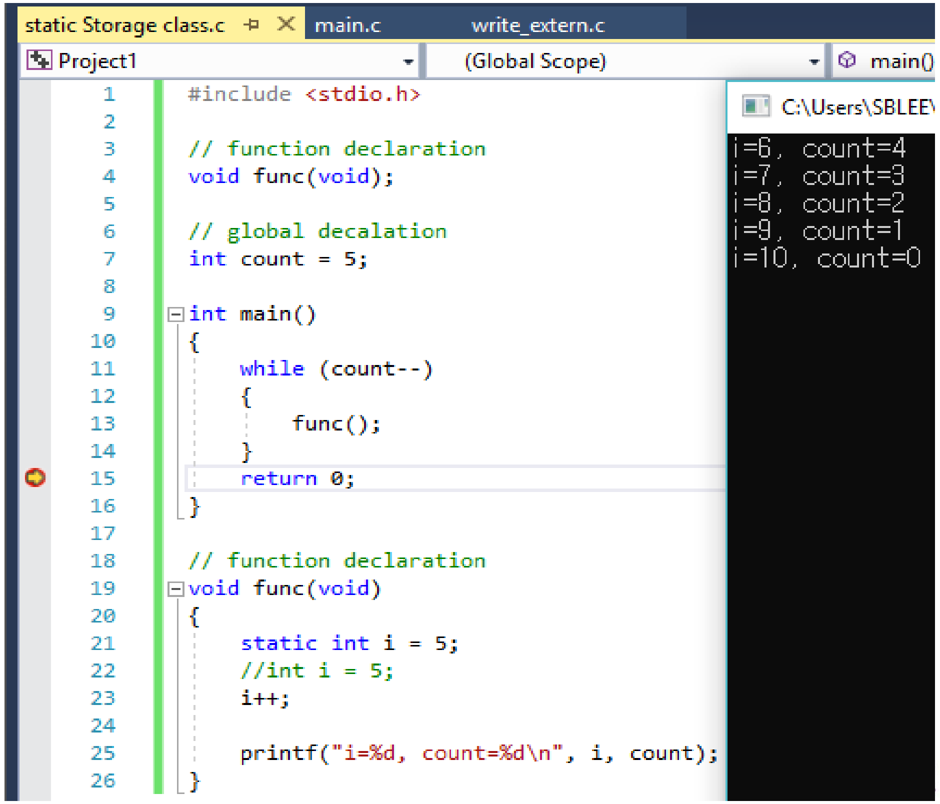
Storage classes

- The static storage classes
  - **Local variable**
    - ➢ The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying the local variable <u>each time it comes into and goes out of scope, i.e., their braces { }.</u>



```c
#include <stdio.h>

// function declaration
void func(void);

// global decalation
int count = 5;

int main()
{
    while (count--)
    {
        func();
    }
    return 0;
}

// function declaration
void func(void)
{
    //static int i = 5;
    int i = 5;
    i++;

    printf("i=%d, count=%d\n", i, count);
}
```

```
i=6, count=4
i=6, count=3
i=6, count=2
i=6, count=1
i=6, count=0
```

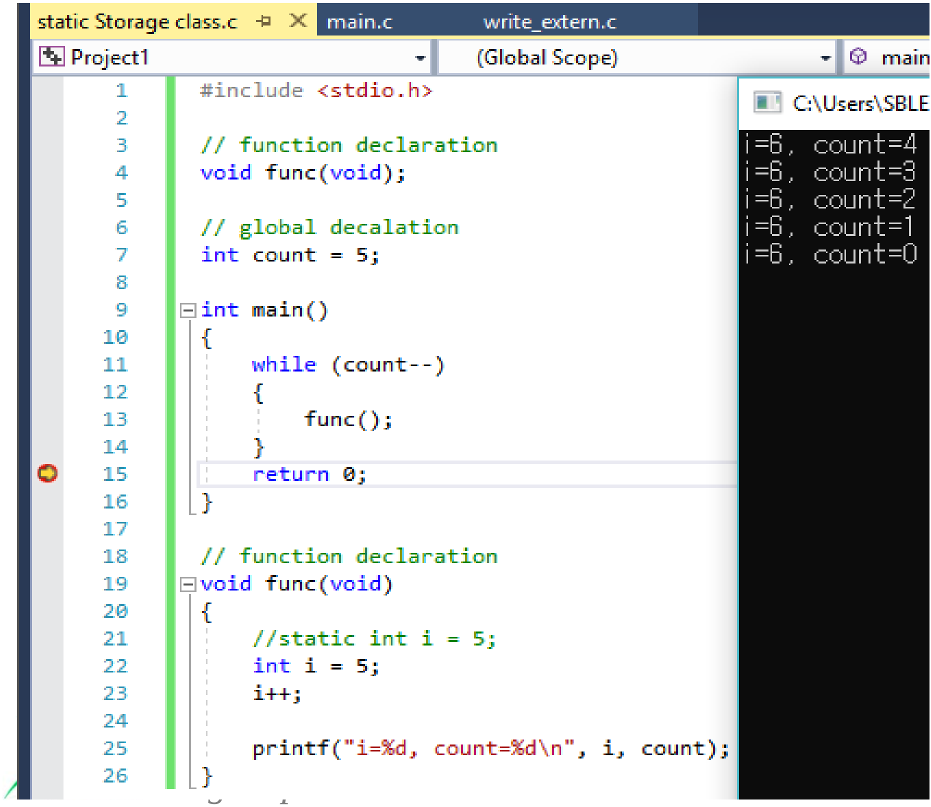


```c
#include <stdio.h>

// function declaration
void func(void);

// global decalation
int count = 5;

int main()
{
    while (count--)
    {
        func();
    }
    return 0;
}

// function declaration
void func(void)
{
    static int i = 5;
    //int i = 5;
    i++;

    printf("i=%d, count=%d\n", i, count);
}
```

```
i=6, count=4
i=7, count=3
i=8, count=2
i=9, count=1
i=10, count=0
```

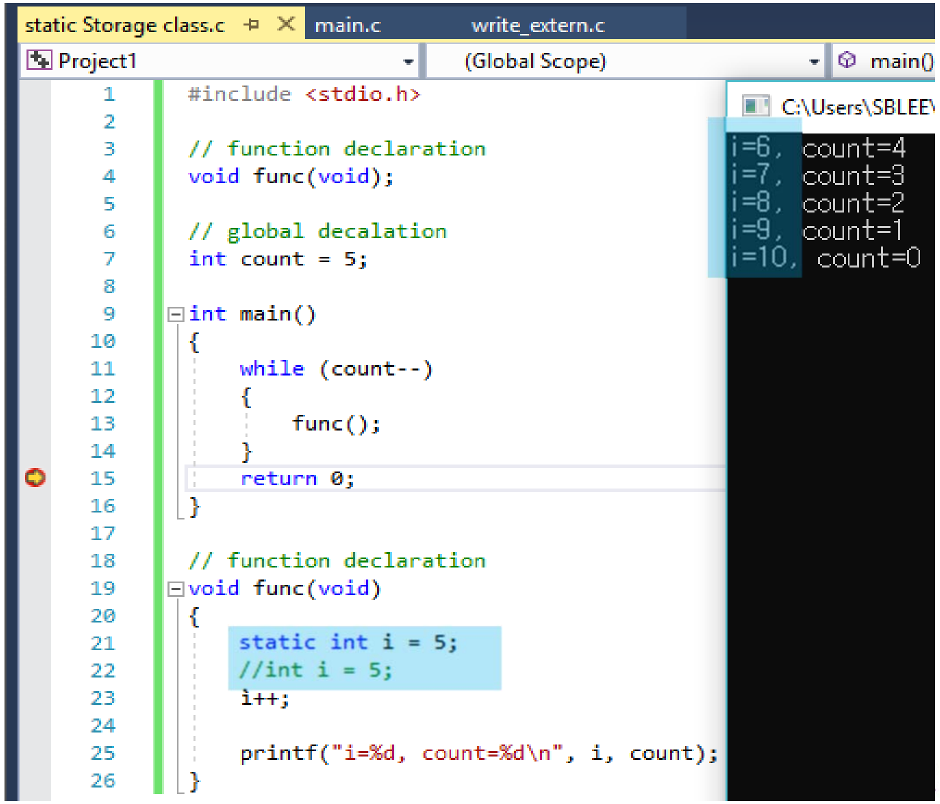
Storage classes

- The static storage classes
  - **Local variable**
    - ➢ The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying the local variable each time it comes into and goes out of scope, i.e., their braces { }.



```c
#include <stdio.h>

// function declaration
void func(void);

// global decalation
int count = 5;

int main()
{
    while (count--)
    {
        func();
    }
    return 0;
}

// function declaration
void func(void)
{
    //static int i = 5;
    int i = 5;
    i++;

    printf("i=%d, count=%d\n", i, count);
}
```

```
i=6, count=4
i=6, count=3
i=6, count=2
i=6, count=1
i=6, count=0
```

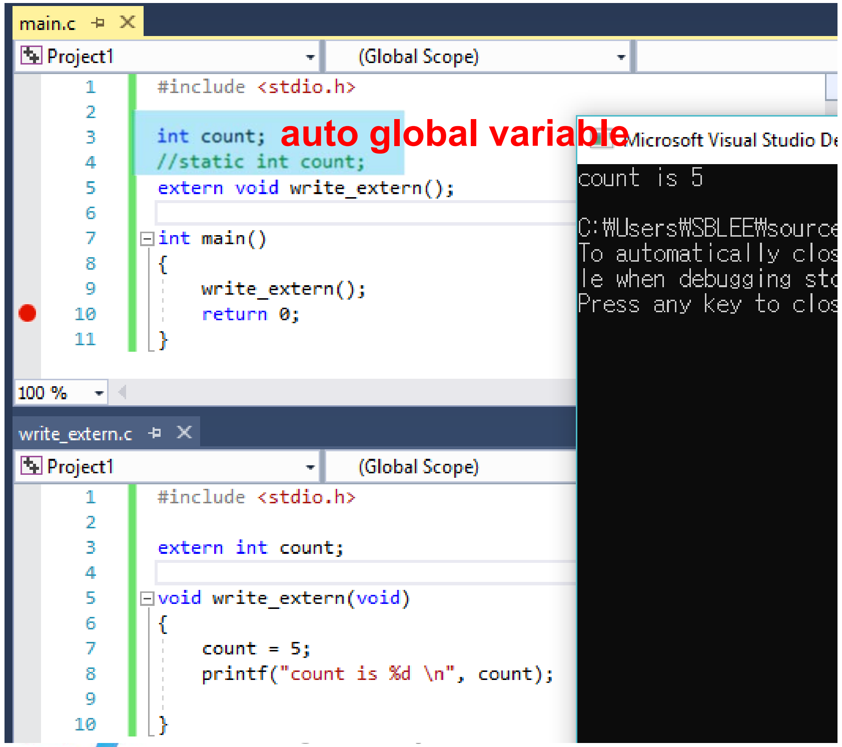


```c
#include <stdio.h>

// function declaration
void func(void);

// global decalation
int count = 5;

int main()
{
    while (count--)
    {
        func();
    }
    return 0;
}

// function declaration
void func(void)
{
    static int i = 5;
    //int i = 5;
    i++;

    printf("i=%d, count=%d\n", i, count);
}
```

```
i=6, count=4
i=7, count=3
i=8, count=2
i=9, count=1
i=10, count=0
```

Storage classes

- The static storage classes
  - **Global variable**
    - When the static modifier is applied to global variables, it causes that variable's scope to be restricted to the file (with extension **.c**) in which it is declared. It means that another file doesn't call the global variables with the static modifier through the extern modifier.



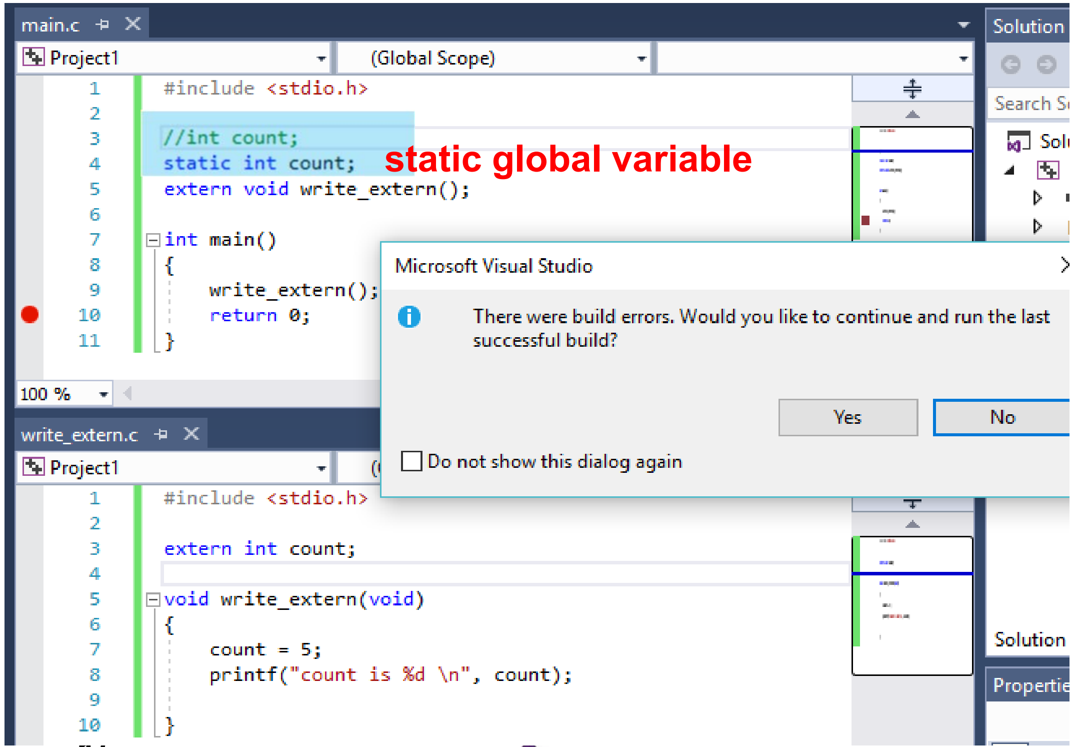
auto global variable

```
#include <stdio.h>

int count;
//static int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Microsoft Visual Studio De

```
count is 5

C:₩Users₩SBLEE₩source
To automatically clos
le when debugging sto
Press any key to clos
```



static global variable

```
#include <stdio.h>

//int count;
static int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Microsoft Visual Studio

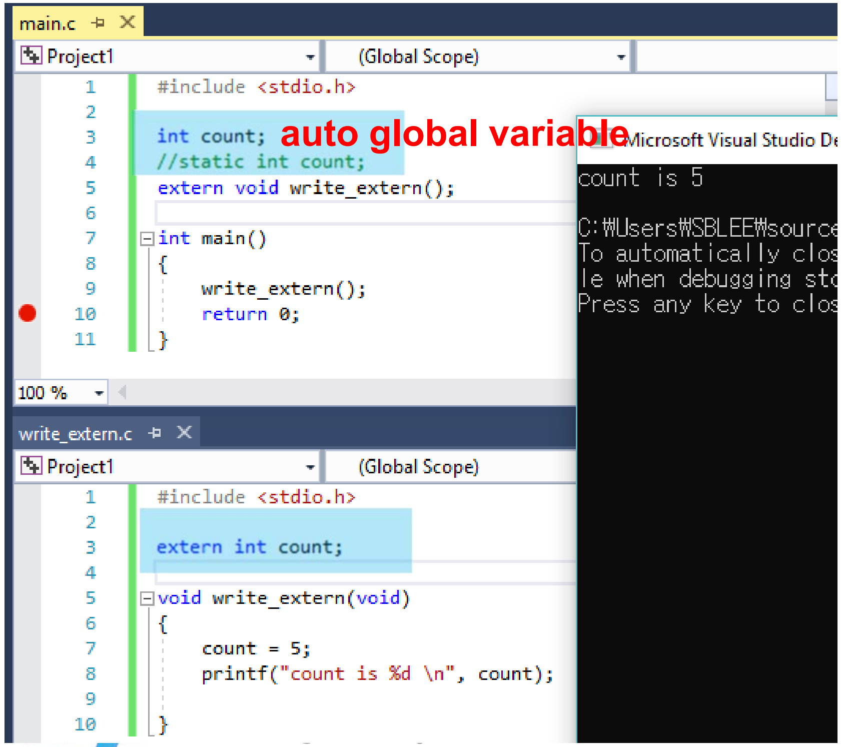
There were build errors. Would you like to continue and run the last successful build?

Yes | No

□ Do not show this dialog again

Storage classes

- The static storage classes
  - **Global variable**
    - When the static modifier is applied to global variables, it causes that variable's scope to be restricted to the file (with extension **.c**) in which it is declared. It means that another file don't call the global variables with the static modifier through the extern modifier.



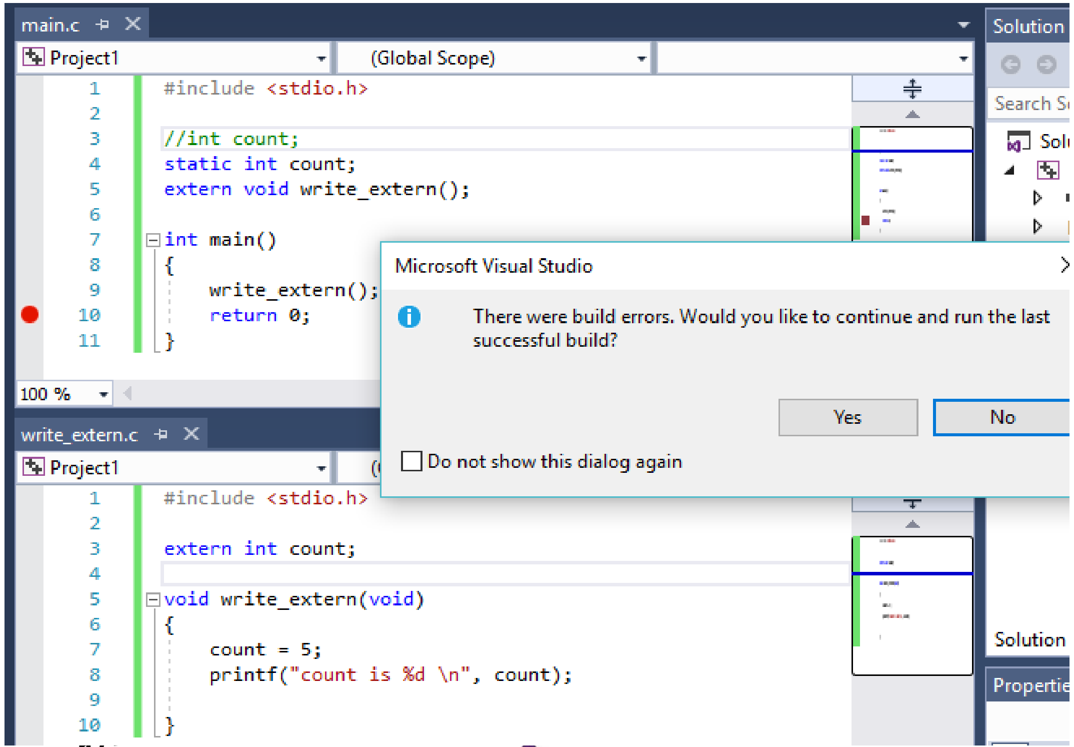
main.c:

```
#include <stdio.h>

int count; auto global variable
//static int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

```
count is 5

C:₩Users₩SBLEE₩source
To automatically clos
le when debugging sto
Press any key to clos
```

write_extern.c:

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```



main.c:

```
#include <stdio.h>

//int count;
static int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

Microsoft Visual Studio

There were build errors. Would you like to continue and run the last successful build?

Yes | No

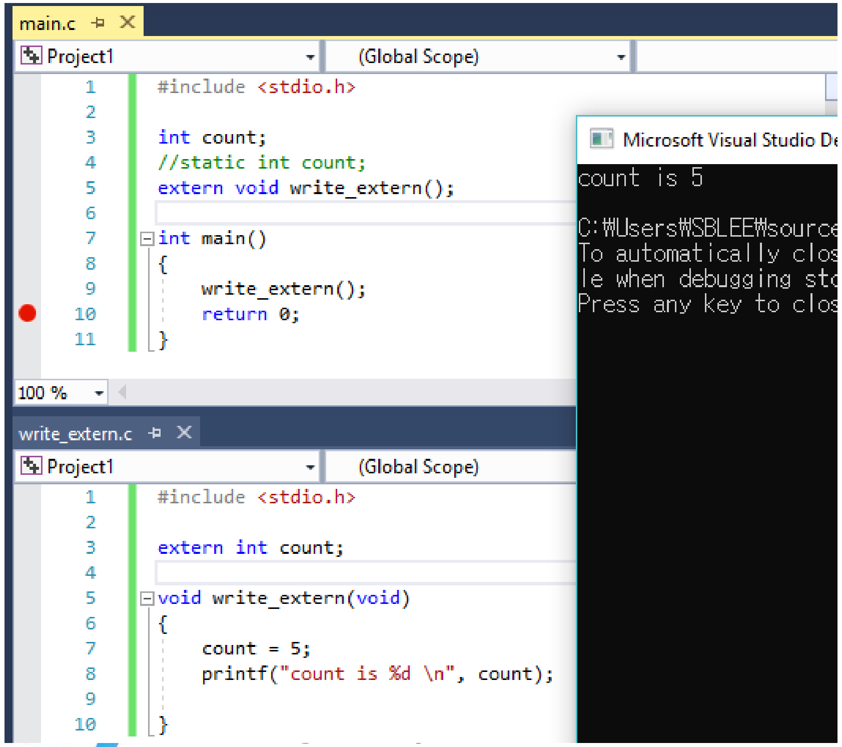
□ Do not show this dialog again

write_extern.c:

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Storage classes

- The static storage classes
  - **Global variable**
    - When the static modifier is applied to global variables, it causes that variable's scope to be restricted to the file (with extension **.c**) in which it is declared. It means that another file don't call the global variables with the static modifier through the extern modifier.

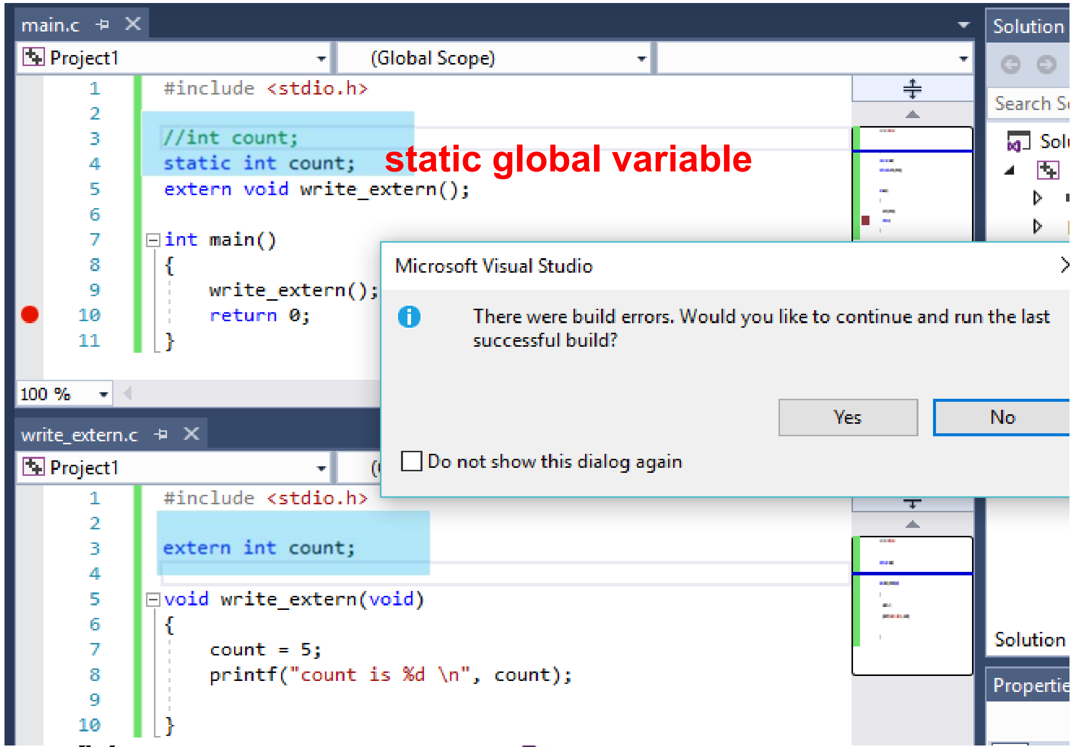


```
#include <stdio.h>

int count;
//static int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Microsoft Visual Studio De

```
count is 5

C:₩Users₩SBLEE₩source
To automatically clos
le when debugging sto
Press any key to clos
```



```
#include <stdio.h>

//int count;
static int count;
extern void write_extern();

int main()
{
    write_extern();
    return 0;
}
```

static global variable

Microsoft Visual Studio

There were build errors. Would you like to continue and run the last successful build?

Yes No

□ Do not show this dialog again

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
    count = 5;
    printf("count is %d \n", count);
}
```

Self-coding class

Self-coding class for the lecture 5 and 6

- Self-coding class
  - After the self-coding class, please submit your codes for all the examples we covered in the **lecture 5 and 6** by e-mail (seungbeop.lee@gmail.com).
  - If you don’t submit your codes for all the examples of the **lecture 3 and 4**, please submit them by e-mail (seungbeop.lee@gmail.com).

Thank You