



## Chapter 04. 함수와 기억클래스

# 목차

1. 함수의 이해
2. 함수의 정의와 호출
3. 함수의 선언
4. 기억클래스와 변수

# 학습목표

- C++로 함수의 개념에 대해 알아본다.
- 함수를 정의하기 위한 리턴형, 함수명, 매개변수에 대해 학습한다.
- 함수의 선언(원형 정의)에 대해 학습한다.
- 기억클래스에 대해서 학습한다.
- 지역변수와 전역변수의 차이점을 이해한다.

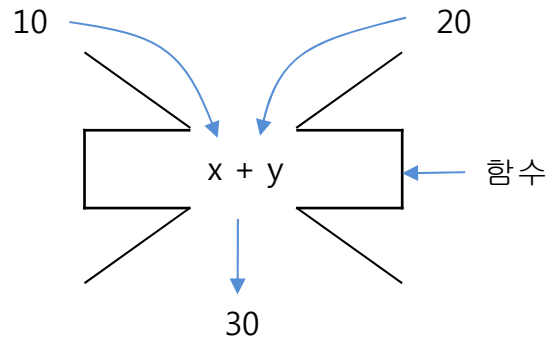
# 01 함수의 이해

## ■ 함수의 필요성과 기본 개념

- 함수는 특정 기능(function)을 처리하기 위한 명령을 묶어 놓은 작은 프로그램 단위로서 프로그램에서 자주 사용되는 코드 블록을 기능 단위로 따로 한 번만 만들어 두고 필요할 때마다 호출해서 그 기능을 처리한다.

자연수의 합을 구하는 함수가 없는 경우	자연수의 합을 구하는 함수를 정의한 경우
<pre>void main() {     int total;     int a=10;     int i;      total=0; // 반드시 초기화해야 한다.     for(i=1; i&lt;=a; i++) {         total+=i;     }     cout&lt;&lt;"total ="&lt;&lt; total&lt;&lt;endl;      total=0; // 반드시 초기화해야 한다.     for(i=1; i&lt;=100; i++) {         total+=i;     }     cout&lt;&lt;"total ="&lt;&lt; total&lt;&lt;endl;      total=0; // 반드시 초기화해야 한다.     for(i=1; i&lt;=5; i++) {         total+=i;     }     cout&lt;&lt;"total ="&lt;&lt; total&lt;&lt;endl; }</pre>	<pre>int sum(int n) {     int total=0; // 반드시 초기화해야 한다.     int i;     for(i=1; i&lt;=n; i++) {         total+=i;     }     return total; }  void main() {     int a=10;     cout&lt;&lt;"total ="&lt;&lt;sum(a)&lt;&lt;endl;     cout&lt;&lt;"total ="&lt;&lt;sum(100)&lt;&lt;endl;     cout&lt;&lt;"total ="&lt;&lt;sum(5)&lt;&lt;endl; }</pre>

# 01 함수의 이해



[그림 4-1] 함수의 동작 원리

## ■ 라이브러리 함수와 사용자 정의 함수

- ① **라이브러리 함수** : 컴파일러를 제작한 곳에서 미리 만들어 제공하는 함수다. 이미 정의되어 있으므로 가져다 사용만 하면 된다. 주의할 점은 각 함수마다 특정 헤더파일을 포함(include)한 후에 사용해야 한다는 것이다.
- ② **사용자 정의 함수** : 프로그램을 작성하는 사용자가 필요에 따라 스스로 만들어 사용하는 함수다.

## ■ 함수를 사용할 때의 장점

- ① 반복적으로 실행해야 할 내용을 함수로 만들어 두고 필요할 때마다 호출해서 사용할 수 있다.
- ② 프로그램이 모듈화(블록화)되므로 읽기 쉽고, 디버그와 편집이 쉽다.
- ③ 프로그램의 기능과 구조를 한 눈에 알아보기 쉽다.
- ④ 다른 프로그램에서 다시 사용할 수 있다.

## 02 함수의 정의와 호출

### ■ 함수를 사용할 때 반드시 존재해야 하는 3가지

- ① 함수의 정의 : 함수에 어떤 내용을 담고 어떻게 동작할 것인지를 기술하는 함수의 정의가 있어야 한다.
- ② 함수의 호출 : 정의한 함수는 호출하지 않으면 실행되지 않으므로 필요한 곳에서 함수를 호출해야 한다.
- ③ 함수의 선언 : 함수의 정의가 나타나기 이전에 함수가 호출되었다면 함수를 반드시 선언(원형 정의)한 후에 호출할 수 있다.

### ■ 함수의 기본 형식

함수 기본 형식	함수 사용 예
자료형 함수명(매개변수 리스트) { 변수 선언; 문장; return 결과값; }	<pre>int sum(int x, int z) {     int add;     add=x+y;     return add; }</pre>

## 02 함수의 정의와 호출

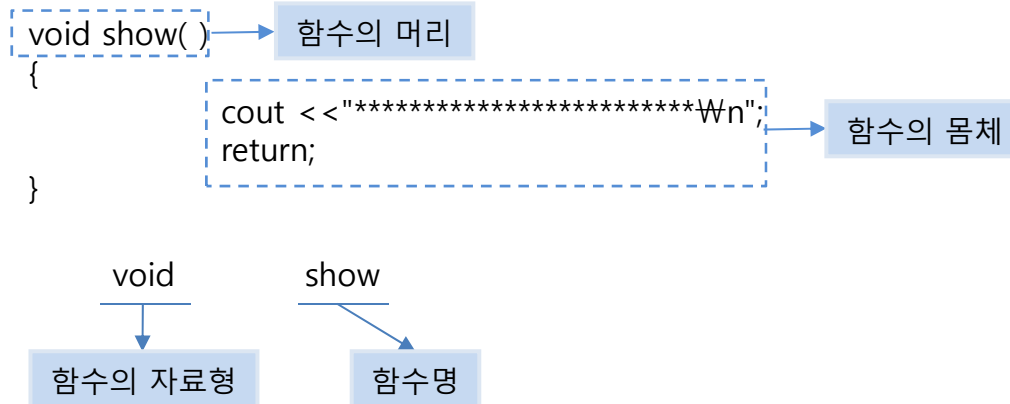
- ① 자료형 : 다음과 같은 규칙에 따라 함수가 호출한 곳에 지정된 **반환값의 자료형**을 알려준다.
  - 반환값이 없으면 void형으로 지정한다.
  - 반환값이 있을 경우 return 다음에 적은 결과값과 동일한 자료형으로 선언해 준다.
- ② 함수명 : 식별자 작성 규칙과 동일하게 작성하고, 가급적 함수명만 보고도 이것이 무슨 기능을 수행하는 함수인지 알 수 있도록 의미있게 명명하는 것이 좋다.
- ③ 매개변수 리스트 : 함수의 입력으로 어떤 매개변수(인자)가 들어오는지를 나타낸다. 일반적으로 매개변수의 자료형이 무엇인지와 어떤 형태(인자의 수와 순서)인지를 나타낸다.

### ■ 함수의 기본 유형

- ① 매개변수와 반환값이 모두 없는 함수
- ② 매개변수만 있고 반환값이 없는 함수
- ③ 매개변수와 반환값이 모두 있는 함수

## 02 함수의 정의와 호출

### ■ 매개변수와 반환값이 모두 없는 함수



- ( )안에 void를 사용해서 매개변수가 없음을 명시할 수 있다.  
`void show(void)`
- 다음과 같은 형태로 호출해야 함수 몸체에 기술된 로직이 수행된다.  
`show( );`

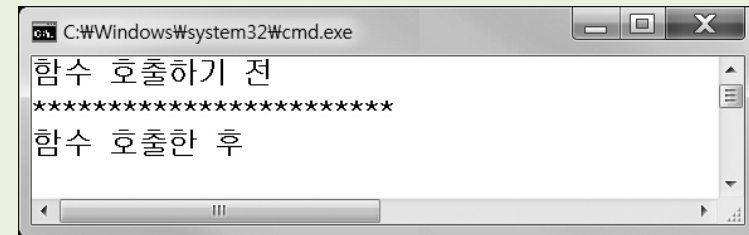
### ■ 사용자 정의 함수가 main 함수와 다른 점

- ① main 함수명은 반드시 main이어야 하지만 사용자 정의 함수에서 함수명은 변수명을 만드는 규칙에 준수해서 만들어 준다.
- ② main 함수는 프로그램이 실행되면서 자동으로 수행되지만 사용자 정의 함수는 프로그래머가 명시적으로 함수를 호출해야 한다.



## 예제 4-1. 매개변수도 반환값도 없는 함수 작성하기(04\_01.cpp)

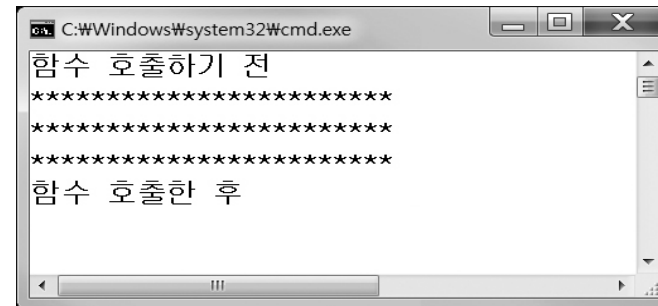
```
01 #include <iostream>
02 using namespace std;
03 void show()
04 {
05     cout << "*****\n";
06     return;
07 }
08 void main()
09 {
10     cout << "함수 호출하기 전\n";
11     show(); // 함수의 호출
12     cout << "함수 호출한 후\n";
13 }
```



## 02 함수의 정의와 호출

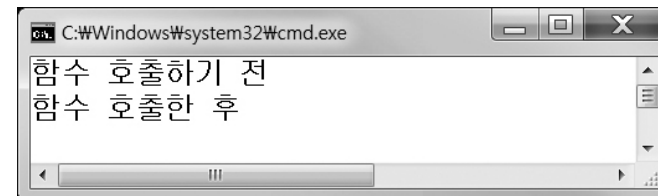
- 함수는 중복되는 코드를 줄여줄 수 있다. 실행 결과는 다음과 같이 함수를 호출한 3곳에서 선이 출력된다.

```
10 cout << "함수 호출하기 전\n";
11 show( );
12 show( );
13 show( );
14 cout << "함수 호출한 후\n";
```



- 함수는 한 번만 정의하고 이 함수가 필요할 때마다 여러 번 호출할 수 있으므로 함수를 유용하게 사용할 수 있다. 반면에 함수를 호출하지 않고 정의만 해두면 사용자 정의 함수에 기술한 내용은 절대 수행되지 않는다.

```
08 void main()
09 {
10     cout << "함수 호출하기 전\n";
11     // show( ); // 함수의 호출
12     cout << "함수 호출한 후\n";
13 }
```



## 02 함수의 정의와 호출

### ■ 매개변수만 있고 반환값이 없는 함수

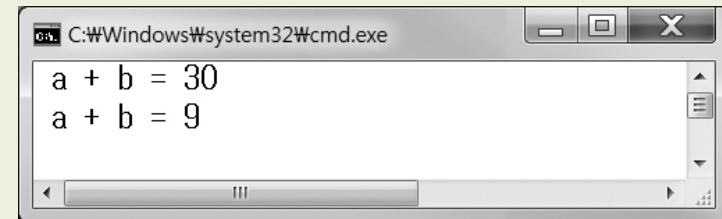
- 실 매개변수 : 함수를 호출할 때 기술되는 매개변수
- 형식 매개변수 : 함수를 정의할 때 기술되는 매개변수

```
void sum(int a, int b)           // a, b는 형식 매개변수
{
    cout << "a + b = " << a + b << "\n" ;
}
void main()
{
    int a=10, b=20;
    sum(a, b);                   // 변수 a, b는 실 매개변수
}
```

- 위 경우, 실 매개변수로 기술한 변수는 기억공간이 전달되는 것이 아니고 변수에 저장되어 있던 값만 전달된다 (call by value).

## 예제 4-2. 두 수의 합을 구하는 함수 작성하기(04\_02.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void sum(int a, int b) // a, b는 형식 매개변수
04 {
05     cout << " a + b = "<< a + b<<"\n" ;
06 }
07 void main()
08 {
09     int a=10, b=20;
10     sum(a, b); // 변수 a, b는 실 매개변수
11     sum(4, 5); // 상수(값) 4, 5 역시 실 매개변수
12 }
```



## 02 함수의 정의와 호출

### ■ 매개변수와 반환값이 모두 있는 함수

- 함수의 실행 결과를 반환하고자 할 때는 return문을 이용한다.

```
return 식;
```

return문 기본 형식

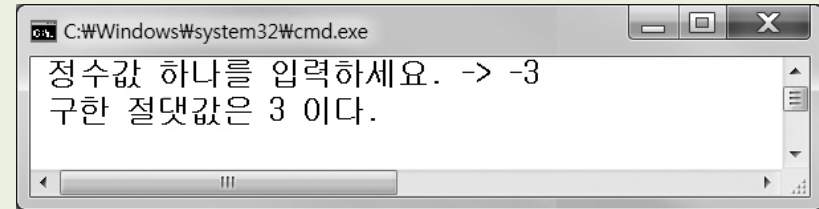
- 위 형식에서 식 부분에는 다음과 같이 상수, 변수, 수식을 사용할 수 있다.

```
return 1;  
return a;  
return (a);  
return a+b;
```

- return문은 2가지 용도로 사용한다.
  - ① 함수를 실행한 결과값을 함수로 호출한 쪽으로 되돌려줄 때 사용한다.
  - ② 현재 실행 시점에서 더 이상 문장을 진행하지 않고 그 함수를 빠져나올 때 사용한다.

## 예제 4-3. 절댓값을 구하는 함수 작성하기(04\_03.cpp)

```
01 #include <iostream>
02 using namespace std;
03 int myAbs(int x)
04 {
05     int y; // 절댓값을 저장할 변수
06     if(x < 0) // 절댓값을 구하는 공식
07         y = -x;
08     else
09         y = x;
10     return y; // 구해진 결과값을 return문으로 반환한다.
11 }
12 void main()
13 {
14     int a, result;
15     cout << "₩n 정수값 하나를 입력하세요. -> ";
16     cin >> a;
17     result = myAbs(a); // 함수의 결과값을 변수 result에 대입한다.
18     cout << " 구한 절댓값은 " << result << " 이다. ₩n" ;
19 }
```



## 03 함수의 선언

### ■ 함수의 선언 (함수의 원형)

- 아직 정의되지 않은(아래에 존재하지만) 함수를 미리 호출하려면 호출하기 전에 함수의 머리 부분을 기술해 두어야 한다. 함수의 선언은 함수를 호출할 때 컴파일러가 매개변수의 자료형과 개수를 확인하려고 사용한다.

함수 선언 형식	함수 선언 사용 예
자료형 함수명(매개변수 리스트);	int myAbs(int x);

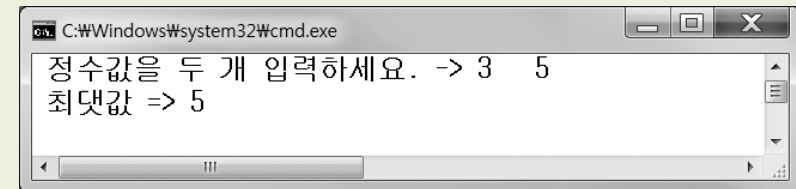
```
❶ #include<iostream>
using namespace std;
int max(int x, int y)
{
    return ( x > y ) ? x : y ;
}
void main( )
{
    int a, b;
    cin >> a >> b;
    cout << max(a, b);
}
```

```
❷ #include<iostream>
using namespace std;
int max(int, int);
void main( )
{
    int a, b;
    cin >> a >> b;
    cout << max(a, b);
}
int max(int x, int y)
{
    return ( x > y ) ? x : y ;
}
```

- ❶은 main 함수 위에 함수를 정의했고 ❷는 main 함수 아래에 함수를 정의했다. ❷처럼 함수를 먼저 호출하고 나중에 정의할 경우에는 반드시 함수 호출 전에 함수를 선언해야 한다.

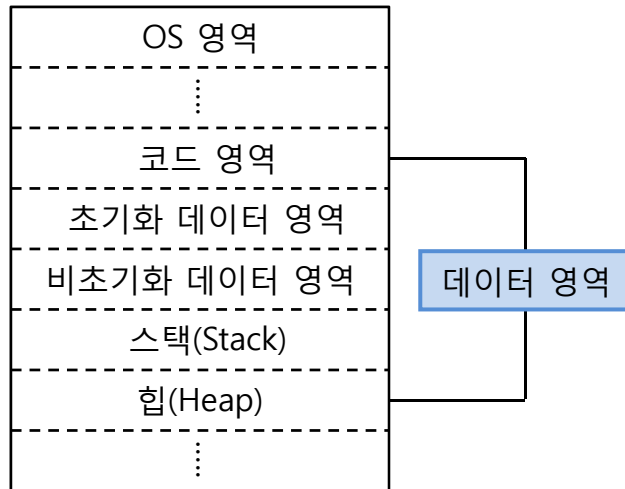
## 예제 4-4. 최댓값을 구하는 함수 작성하기(04\_04.cpp)

```
01 #include <iostream>
02 using namespace std;
03 int max(int, int) ; // 함수의 원형
04 void main()
05 {
06     int a, b; // 정수형으로 변수 2개를 선언한다.
07     cout << "정수값을 두 개 입력하세요. => ";
08     cin >> a >> b;
09     // 함수의 선언
10     cout << " 최댓값 => " << max(a, b) << "\n";
11 }
12 int max(int x, int y) // 함수의 정의
13 {
14     return ( (x > y) ? x : y ); // return문에 조건 연산자
15 }
```





## 04 기억클래스와 변수



[그림 4-2] 램(RAM) 구조

- 세분화된 데이터 영역에서 변수가 어디에 어떻게 확보될지를 결정하는 것이 기억클래스다.

[표 4-1] 기억클래스의 종류

구분	유효 범위	생존 기간	메모리	초기화 여부
auto(자동변수)	블록 내	일시적	스택	쓰레기값
extern(외부변수)	프로그램 내	영구적	메모리	숫자 0
static(정적변수)	내부 : 블록 내 외부 : 모듈 내	영구적	메모리	숫자 0
register(레지스터 변수)	블록 내	일시적	CPU 내의 레지스터	쓰레기값

## 04 기억클래스와 변수

- 생존 기간

- ❶ 일시적 : 변수가 선언된 시점에서 생성되었다가 블록 종료 시 자동 소멸된다. 프로그램 내에서 생성과 소멸이 여러 번 반복된다.
- ❷ 영구적 : 프로그램이 시작할 때 생성되어 프로그램 실행 중에 항상 메모리상의 동일한 기억 공간에 존재한다. 프로그램이 완전히 끝날 때 비로소 변수가 소멸된다.

- 일반적으로 선언하는 변수는 기억클래스가 생략된다.

변수 선언 기본 형식	변수 선언 사용 예
[기억클래스] 자료형 변수명;	<code>auto int a = 0;</code>

- 변수 선언은 자료형과 기억클래스에 의해 다음 2가지로 구분될 수 있다.

- ❶ 자료형에 의한 구분 : 생략할 수 없으며 변수에 저장되는 값의 형태와 기억공간의 크기를 결정한다 (int, char, float 등).
- ❷ 기억클래스에 의한 구분 : 생략할 수 있으며 생략할 경우 auto로 인식한다. 유효 범위와 생존 기간 등 변수의 성격을 결정한다(auto, extern, static 등).

## 04 기억클래스와 변수

### ■ 자동변수 (auto variable)

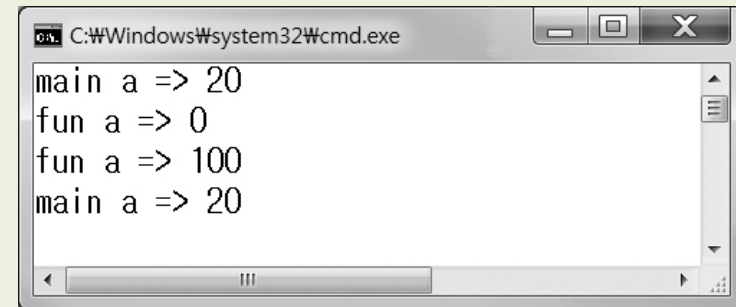
- 자동변수(auto)는 함수 정의부 안에 선언된 변수로서 함수의 매개변수도 자동변수에 포함되며, 프로그램이 실행되는 동안 생성과 소멸을 반복한다.
- 변수가 선언되면 메모리 할당이 일어나고, 변수 선언이 된 블록 밖으로 벗어나면 메모리가 해제되어 변수를 사용할 수 없게 된다.
- 가장 일반적으로 사용하는 형태가 자동변수이며, 자동변수는 auto라는 기억클래스를 지정자를 생략한다.

### ■ 지역변수 (local variable)와 전역변수 (global variable)

- 지역변수 : 블록(함수) 안에 선언된 변수를 뜻하며, 변수를 사용할 수 있는 범위는 그 변수가 선언된 블록 내부로 한정된다.
- 전역변수 : 변수 선언을 함수 외부에서 하는 것이다. 전역변수는 프로그램 내의 모든 함수에서 사용할 수 있는 변수다. 또한 외부변수는 프로그램이 실행되는 동안 그 값이 유지된다. 전역변수는 자동으로 초기화 (수치 데이터라면 0으로)가 이루어진다.

## 예제 4-6. 전역변수의 성격 알아보기(04\_06.cpp)

```
01 #include<iostream>
02 using namespace std;
03 void fun();
04 int a; // 전역변수 a
05 void main()
06 {
07     int a = 20; // 지역변수 a
08     cout<<"\n main a => "<<a;
09     fun();
10     cout<<"\n main a => "<<a<<"\n";
11 }
12 void fun()
13 {
14     cout<<"\n fun a => "<<a;
15     a=a+100;
16     cout<<"\n fun a => "<<a;
17 }
```



```
C:\Windows\system32\cmd.exe
main a => 20
fun a => 0
fun a => 100
main a => 20
```

## 04 기억클래스와 변수

### ■ 정적변수 (static variable)

- 변수를 선언할 때 변수명 앞에 static을 붙이면 정적변수로 선언된다.
  - 지역변수로서의 정적변수 : 블록 내부에 변수가 선언된다.
  - 전역변수로서의 정적변수 : 블록 외부에 변수가 선언된다.

### ■ 지역변수로서의 정적변수

[표 4-2] 자동변수와 지역변수로서의 정적변수 비교

변수명	구분	유효 범위	생존 기간	초기화 여부	변숫값 여부
auto	자동변수	블록 내	일시적	쓰레기 값	유지 안됨
static	지역변수로 정적변수	블록 내	영구적	숫자 0	유지됨

### ■ 전역변수로서의 정적변수

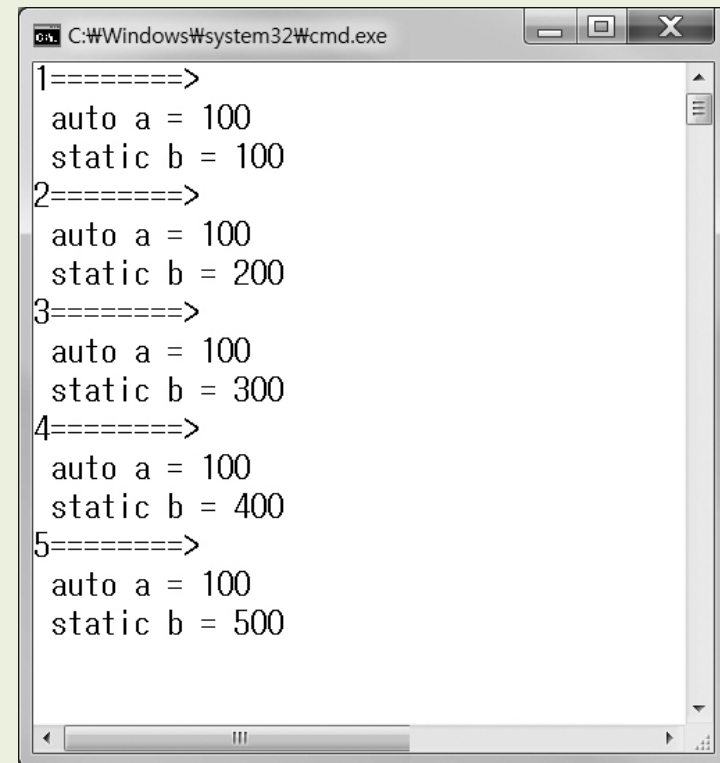
- 프로그램을 작성할 때 파일 여러 개를 모아서 만든 프로그램 하나를 프로젝트라고 한다. 프로젝트로 묶여있는 모든 파일 내에서 공유해서 사용할 수 있는 변수를 외부변수라 하며, 다른 파일에 선언된 외부변수에 접근하려면 사용하기 전에 extern을 붙여 변수를 선언한 후 사용한다.

외부변수 선언 기본 형식	외부변수 선언 사용 예
extern 자료형 변수명;	extern int a;

- 전역변수로서의 정적변수는 그 변수가 선언되어 있는 파일에서만 사용할 수 있다. 즉, 파일 하나에서만 사용할 수 있는 전역변수가 필요하다면 static을 붙인다.

## 예제 4-7. 자동변수와 지역변수로서의 정적변수의 차이점 (04\_07.cpp)

```
01 #include <iostream>
02 using namespace std;
03 void sub();
04 void main()
05 {
06     for(int i=1; i<=5; i++){
07         cout<< i << "=====>\\n";
08         sub();
09     }
10 }
11 void sub()
12 {
13     int a = 0;
14     static int b = 0;
15     a+=100;
16     b+=100;
17     cout<<" auto a = "<<a<<endl;
18     cout<<" static b = "<<b<<endl;
19 }
```

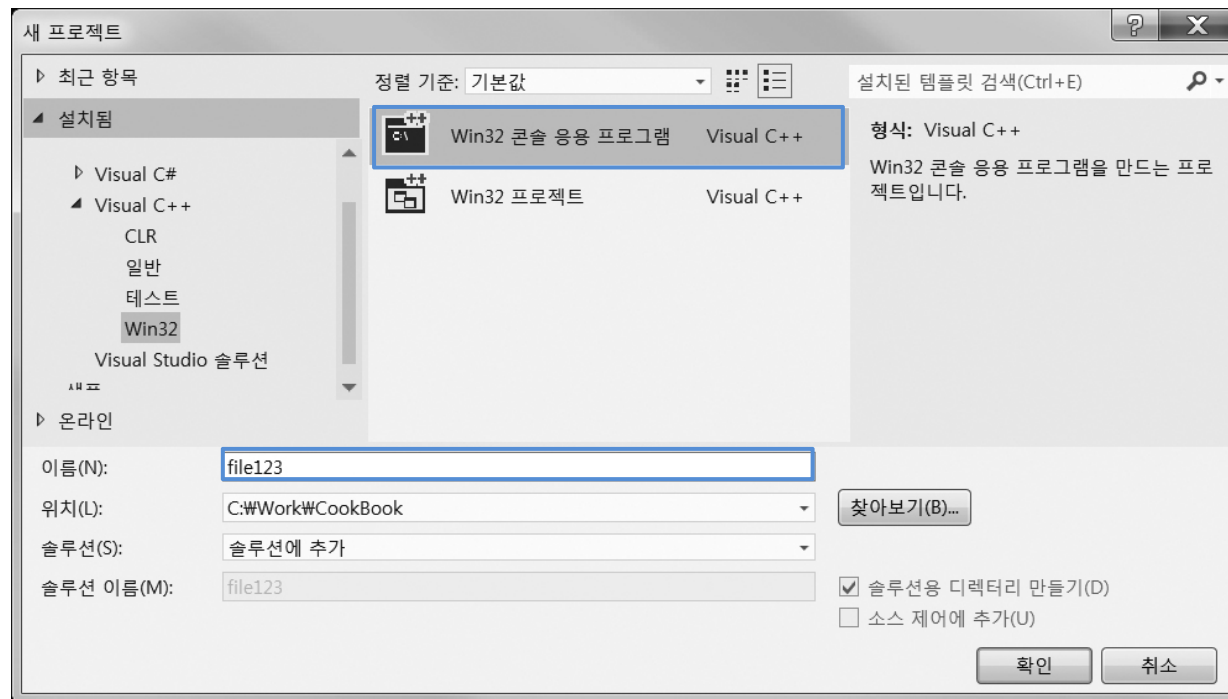


```
C:\\Windows\\system32\\cmd.exe
1=====>
  auto a = 100
  static b = 100
2=====>
  auto a = 100
  static b = 200
3=====>
  auto a = 100
  static b = 300
4=====>
  auto a = 100
  static b = 400
5=====>
  auto a = 100
  static b = 500
```

## 04 기억클래스와 변수

### ■ 소스파일 3개로 구성된 프로젝트 작성하기

#### ① 새 프로젝트 생성



## 04 기억클래스와 변수

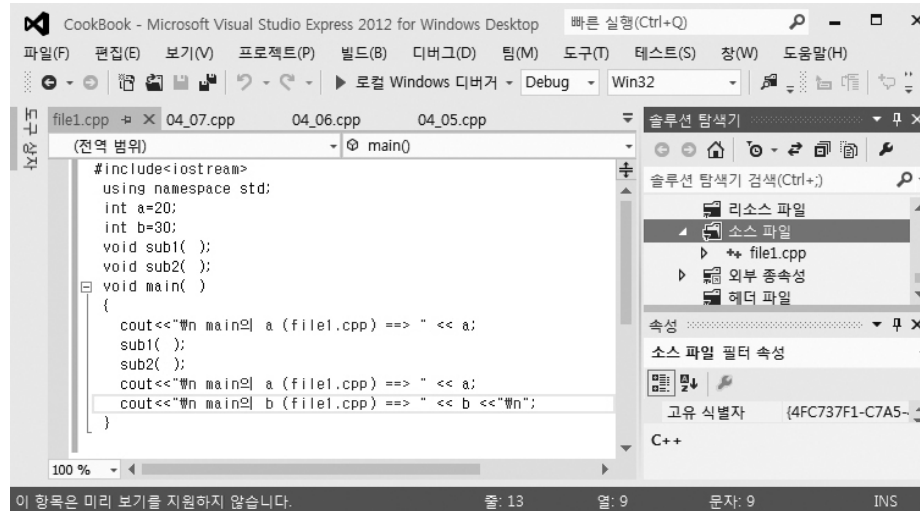
② 소스입력 : 'file1.cpp'

```
01  #include<iostream>
02  using namespace std;
03  int a=20;
04  int b=30;
05  void sub1( );
06  void sub2( );
07  void main( )
08  {
09      cout<<"\n main의 a (file1.cpp) ==> " << a;
10      sub1( );
11      sub2( );
12      cout<<"\n main의 a (file1.cpp) ==> " << a;
13      cout<<"\n main의 b (file1.cpp) ==> " << b <<"\n";
14  }
```



## 04 기억클래스와 변수

### ③ 새 파일 생성



### ④ 다른 파일에 선언된 외부변수를 가져다 사용 : 'file2.cpp'

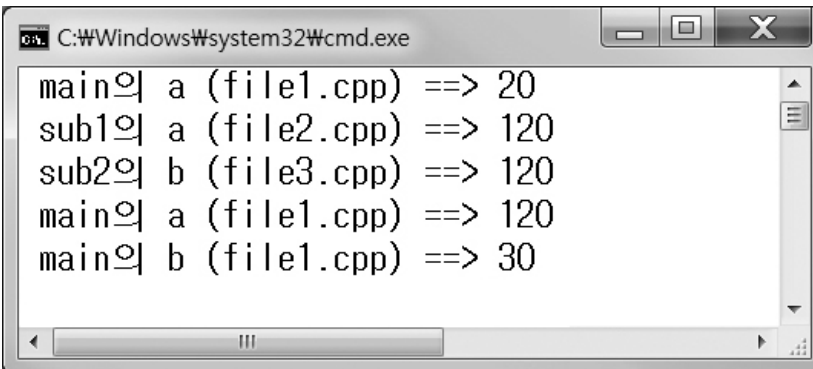
```
01 #include<iostream>
02 using namespace std;
03 extern int a;
04 void sub1( )
05 {
06     a+=100;
07     cout<<"\n sub1의 a (file2.cpp) ==> " << a;
08 }
```

## 04 기억클래스와 변수

⑤ 파일 하나에서만 사용하는 전역변수 선언 : 'file3.cpp'

```
01 #include<iostream>
02 using namespace std;
03 static int b=20;
04 void sub2( )
05 {
06     b+=100;
07     cout<<"\n sub2의 b (file3.cpp) ==> " << b;
08 }
```

⑥ 실행 결과



```
C:\Windows\system32\cmd.exe
main의 a (file1.cpp) ==> 20
sub1의 a (file2.cpp) ==> 120
sub2의 b (file3.cpp) ==> 120
main의 a (file1.cpp) ==> 120
main의 b (file1.cpp) ==> 30
```

# Homework

---

- Chapter 4 Exercise: 7, 9, 10, 11, 13, 15