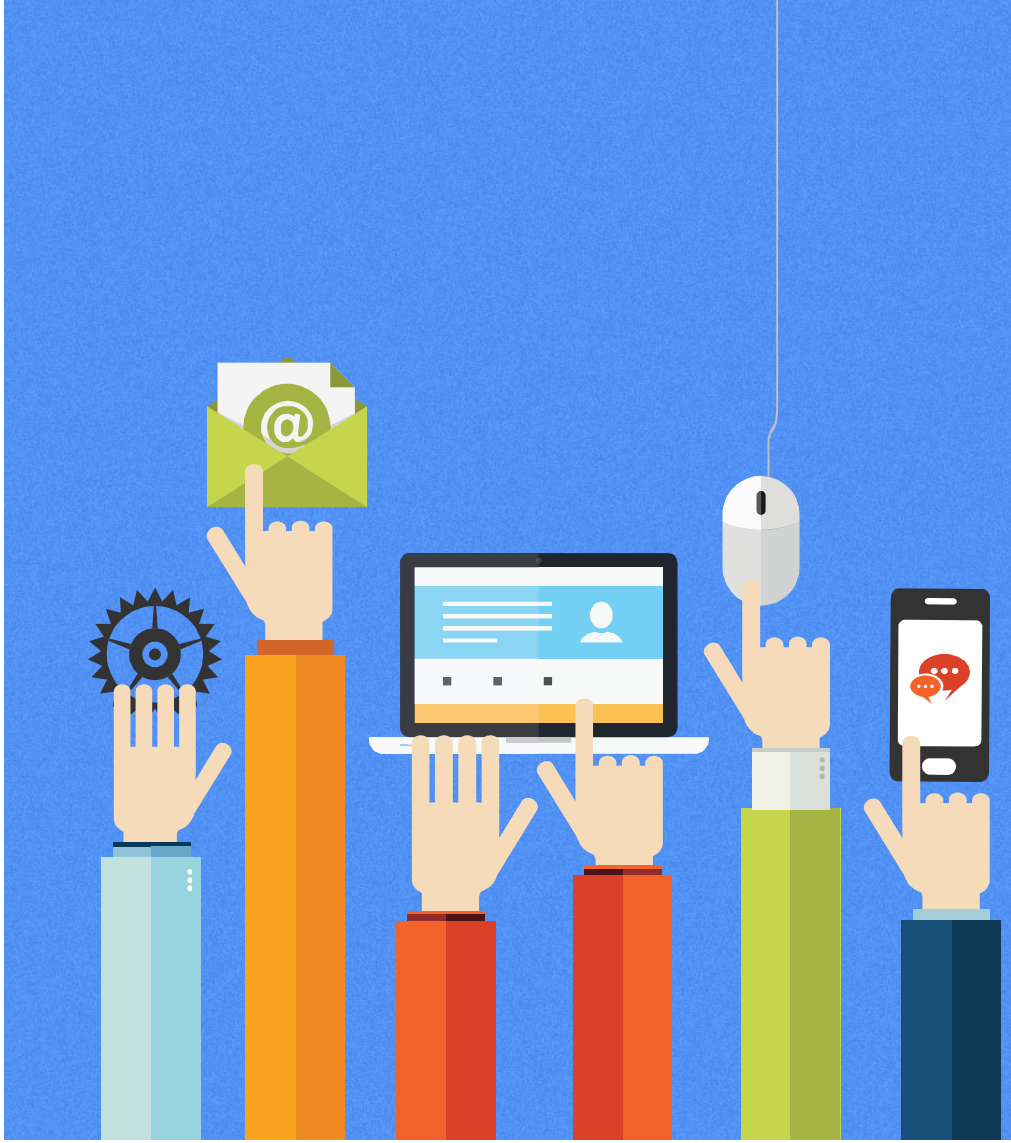


제12장

변수 유효범위

Perfect C

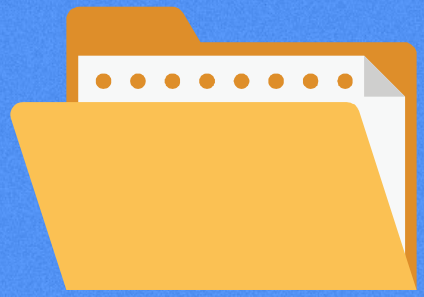
C언어로 배우는 프로그래밍 기초

단원 목표

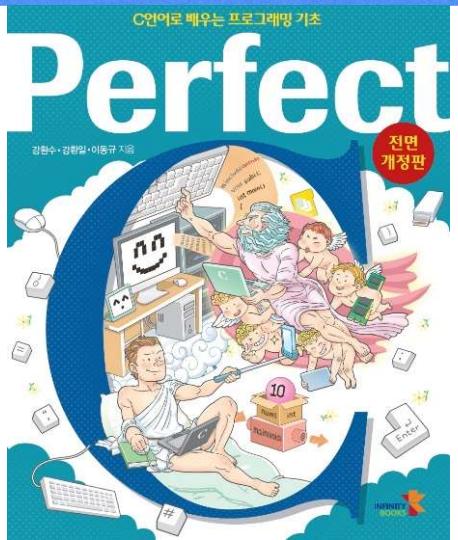
학습목표

- **변수 유효범위를 이해하고 설명할 수 있다.**
  - 지역변수, 자동변수를 선언하고 사용
  - 전역변수를 선언하고 사용
  - 키워드 extern의 필요성과 사용 방법
- **정적 변수와 레지스터 변수를 이해하고 설명할 수 있다.**
  - 기억 부류 auto, static, register, extern
  - 지역 정적 변수와 전역 정적 변수의 필요성과 사용

학습목차



01. 전역변수와 지역변수



변수 scope(1)

- **변수의 유효 범위(scope)**
  - 변수의 참조가 유효한 범위
  - 변수의 유효 범위 구분
    - 지역 유효 범위(local scope)와 전역 유효 범위(global scope)로 나뉨
  - 지역 유효 범위
    - 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위
  - 전역 유효 범위
    - 파일에서만 변수의 참조가 가능한 범위
    - 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위

그림 12-1 변수의 유효범위

변수 scope(2)

- **하나의 프로젝트는 여러 파일로 구성 가능**
  - 다음은 파일 main.c, 그리고 파일 sub1.c와 sub2.c로 구성되는 프로그램
  - 파일 main.c의 상단에 선언된 global
    - 프로젝트 전체 파일에서 사용될 수 있는 전역변수
  - 파일 main.c의 함수 asub()에서 선언된 local
    - 함수 내부에서만 사용될 수 있는 지역변수
  - 파일 sub2.c의 상단에 선언된 staticvar
    - 파일 sub2.c에서 사용될 수 있는 전역변수
  - 파일sub1.c에서 파일 main.c의 상단에 선언된 global을 사용하려면
    - extern int global로 선언이 필요

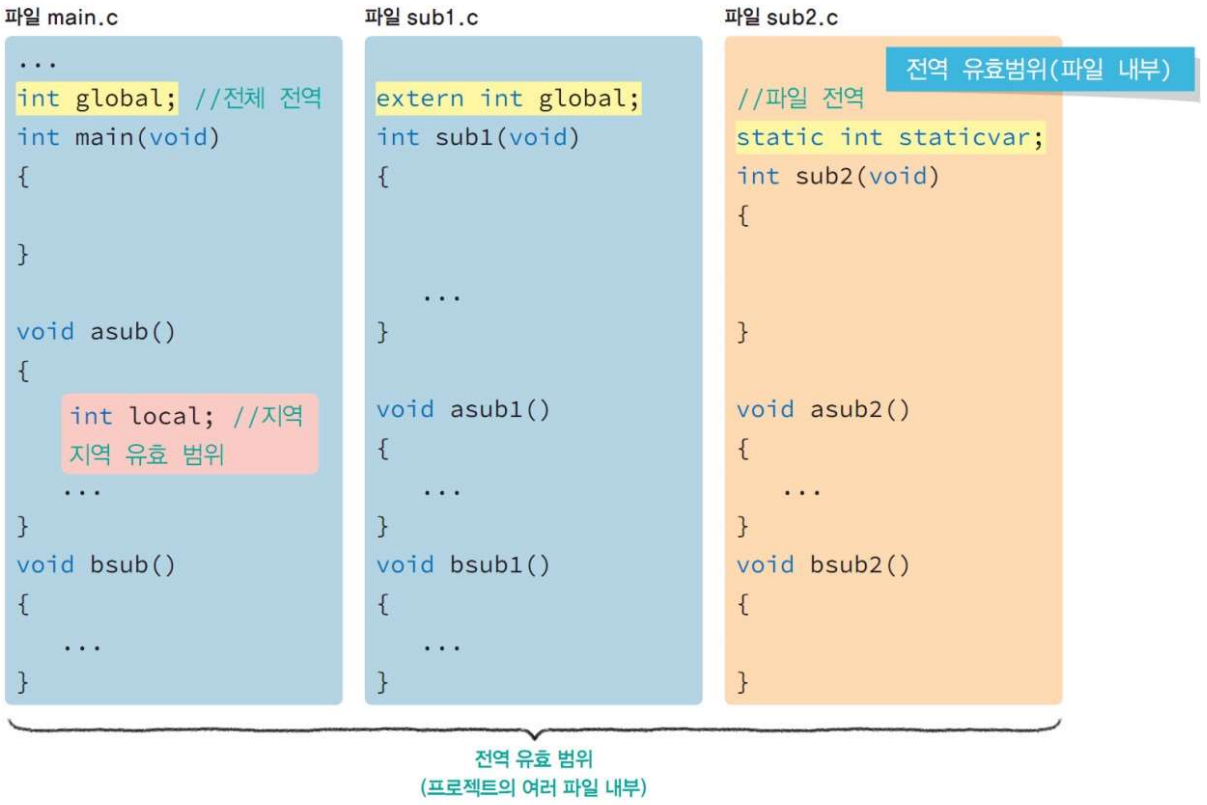


그림 12-2 변수의 유효 범위(지역, 전체 파일 전역, 현재 파일 전역)

지역변수(1)

- **국내 전용 카드가 지역 변수**
  - 국내외 사용 카드는 전역 변수
- **지역변수**
  - 함수 또는 블록에서 선언된 변수
    - 내부변수 또는 자동변수라고도 부름
  - 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능
    - 다른 함수나 블록에서는 사용 불가능
  - 함수의 매개변수도 함수 전체에서 사용 가능한 지역변수
  - 선언 후 초기화하지 않으면 쓰레기값이 저장되므로 주의
  - 변수가 선언된 함수 또는 블록에서 선언 문장이 실행되는 시점에서 메모리에 할당

그림 12-3 지역 변수와 전역 변수

지역변수(2)

- **스택(stack)**
  - 지역변수가 할당되는 메모리 영역
- **선언된 부분에서 자동으로 생성되고**
  - 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거
  - 이러한 이유에서 지역변수는 자동변수(automatic variable)라 부름
  - 지역변수 선언에서 자료형 앞에 키워드 auto가 사용될 수 있음
  - 키워드 auto는 생략 가능하여 일반적으로 auto가 없는 경우가 대부분

```
int main(void)
{
    //지역변수 선언
    int n = 10;

    printf("%d\n", n);

    //m, sum은 for 문 내부의 블록 지역변수
    for (int m = 0, sum = 0; m < 3; m++)
    {
        sum += m;
        printf("%d %d\n", m, sum);
    }

    printf("%d %d\n", n, sum);

    return 0;
}
```

지역변수 n의 영역 유효 범위

지역변수 sum 영역 유효 범위

오류발생: error C2065: 'sum' : 선언되지 않은 식별자입니다.

그림 12-4 지역변수 선언과 유효 범위(scope)

지역변수 예제

예제 localvar.c

- 함수와 블록의 지역변수 사용

제어변수

- for문 블록에서 선언된 지역변수 sum
  - for 문 블록에서만 사용이 가능
  - sum은 블록 외부에서 참조가 불가능
- 함수 sub(int param)에서와 같이 매개변수 param
  - 지역변수와 같이 사용

실습예제 12-1 localvar.c

지역변수 선언과 사용

```
01  // file: localvar.c
02  #include <stdio.h>
03
04  void sub(int param);
05
06  int main(void)
07  {
08     //지역변수 선언
09     auto int n = 10;
10     printf("%d\n", n);
11
12     //m, sum은 for 문 내부의 블록 지역변수
13     for (int m = 0, sum = 0; m<3; m++)
14     {
15        sum += m;
16        printf("\t%d %d\n", m, sum);
17     }
18
19     printf("%d\n", n); //n 참조 가능
20     //printf("%d %d\n", m, sum); //m, sum 참조 불가능
21
22     //함수호출
23     sub(20);
24
25     return 0;
26  }
27
28  //매개변수인 param도 지역 변수와 같이 사용
29  void sub(int param)
30  {
31     //지역변수 local
32     auto int local = 100;
33     printf("\t%d %d\n", param, local); //param과 local 참조 가능
34     //printf("%d\n", n); //n 참조 불가능
35  }
```

설명

```
09  지역 변수 n 선언하여 10 저장, 변수 n은 유효 범위는 9행에서 26행까지
13  for 문 내부 블록 지역함수 m과 sum을 선언, 변수 m, sum은 유효 범위는 13행에서 17행까지
29  매개변수 param은 지역변수로 유효 범위는 30행에서 35행까지
```

```
32  지역 변수 local의 유효 범위는 32행에서 35행까지
```

실행결과

```
10          0  0
            1  1
            2  3
10          20 100
```

전역변수

- **전역변수(global variable)**
  - 함수 외부에서 선언되는 변수
  - 외부변수라고도 부름
  - 일반적으로 프로젝트의 모든 함수나 블록에서 참조 가능
  - 선언되면 자동으로 초기값이 자료형에 맞는 0으로 지정
    - 즉 정수형은 0, 문자형은 null 문자인 '₩0', 실수형은 0.0, 포인터 형은 NULL 값이 저장
- **함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언 가능**
  - 함수내부나 블록에서 그 이름을 참조하면 지역변수로 인식
    - 그러므로 지역변수와 동일한 이름의 전역변수는 참조 불가능
    - 가능한 이러한 변수는 사용하지 않도록
- **전역변수는 동일 프로젝트의 다른 파일에서도 참조가 가능**
  - 다른 파일에서 선언된 전역변수를 참조하려면
    - 키워드 extern을 사용하여 이미 다른 파일에서 선언된 전역변수임을 선언
  - extern을 사용한 참조선언 구문
    - 변수선언 문장 맨 앞에 extern을 넣는 구조
      - extern 참조선언 구문에서 자료형은 생략 가능
    - 키워드 extern을 사용한 변수 선언은 새로운 변수를 선언하는 것이 아니며,
      - 단지 이미 존재하는 전역변수의 유효 범위를 확장

한 프로젝트에 여러 소스 파일로 구성

- **다음 프로젝트는 파일 global.c와 circumference.c로 구성된 예제**
  - 함수 getArea()와 getCircum()에서 파일 global.c에서 선언된 전역변수 PI를 사용
  - 함수 getCircum()이 구현된 파일은 circumference.c

- 전역변수 PI가 선언된 파일과 다름
- 파일 circumference.c에서 전역변수 PI를 사용하려면 문장 extern double PI;
- 외부에서 이미 선언된 변수임을 알리는 선언을 다시 해주어야 함

파일 global.c

```
// file: globalvar.c
...
//전역변수 선언
double PI = 3.14;

int main(void)
{
   //지역변수 선언
   double r = 5.87;
   //전역변수 PI와 같은 이름의 지역변수 선언
   const double PI = 3.141592;

   ...
   printf("PI: %f\n", PI);    //지역변수 PI 참조
   return 0;
}

double getArea(double r)
{
   return r*r*PI;       //전역변수 PI 참조
}
```

파일 circumference.c

```
// circumference.c

//이미 선언된 전역변수 선언
extern double PI;

double getCircum(double r)
{
   //전역변수 PI 참조
   return 2*r*PI;
}
```

다른 파일에서 선언된 전역변수 PI임을 알리는 변수 선언이 필요하다

전역변수 PI 변수범위

지역변수 PI 변수범위

전역변수 PI와 같은 지역변수는 선언이 가능하고 사용할 수 있다.

그림 12-5 전역변수 선언과 유효 범위

전역변수 사용(1)

예제1 globalvar.c

- 전역변수 PI, gi 선언

제어변수

- 함수 main()에서 전역변수와 같은 이름의 지역변수 PI를 선언
  - 전역변수와 동일한 이름의 지역변수 선언은 더 이상 함수 main()에서 전역변수 PI를 다시 참조할 수 없게 만듬
- 전역변수 gi는 선언 후 초기값을 저장하지 않아도 자동으로 기본값(default value)인 0이 저장

실행결과

```
면적: 108.19
둘레1: 36.88
둘레2: 36.86
PI: 3.141592
gi: 0
```

전역 변수gi는 초기값이 저장되지 않았으나 자동으로 0이 저장됨

실습예제 12-2 globalvar.c

전역변수 PI의 선언과 사용

```c
// file: globalvar.c
#include <stdio.h>

double getArea(double);
double getCircum(double);

//전역변수 선언
double PI = 3.14;
int gi;

int main(void)
{
   //지역변수 선언
   double r = 5.87;
   //전역변수 PI와 같은 이름의 지역변수 선언
   const double PI = 3.141592;

   printf("면적: %.2f\n", getArea(r));
   printf("둘레1: %.2f\n", 2 * PI*r);
   printf("둘레2: %.2f\n", getCircum(r));
   printf("PI: %f\n", PI);    //지역변수 PI 참조
   printf("gi: %d\n", gi);    //전역변수 gi 기본값

   return 0;
}

double getArea(double r)
{
   return r*r*PI;       //전역변수 PI 참조
}
```

설명

```
전역 변수 PI 선언하여 3.14 저장, 변수 PI의 유효 범위는 이 파일 및 프로젝트 전체
전역 변수 gi 선언하여 초기값을 대입하지 않았으나 기본값인 0이 저장되고 변수 gi의 유효 범위는 이 파일 및 프로젝트 전체
지역 변수 r
지역 변수 PI 선언, 그러나 이 변수는 전역 변수 PI와 이름이 충돌, 함수 내부에서 지역 변수가 우선하므로 전역 변수는 참조가 불가능
변수 PI는 지역 변수인 PI
지역 변수인 PI 출력
전역 변수인 gi의 기본값 출력
r은 매개변수이며, PI는 8줄에서 선언된 전역 변수
```

전역변수 사용(2)

예제2 circumference.c

- 외부 변수의 사용

외부변수 PI

- 다른 파일의 전역변수 PI 사용
  - extern으로 선언 후 사용

실습예제 12-3 circumference.c

외부 전역변수 PI를 참조

```
01  // circumference.c
02  //이미 외부에서 선언된 전역변수임을 알리는 선언
03  extern double PI;
04
05  double getCircum(double r)
06  {
07     //extern double PI;     //함수 내부에서만 참조 가능
08     return 2 * r * PI;    //전역변수 PI 참조
09  }
```

설명

03 파일 globalvar.c 에서 선언된 전역 변수 PI를 사용하려는 문장

08 r은 매개변수이며, PI는 globalvar.c 파일 8줄에서 선언된 전역 변수

전역변수 장단점

- **전역변수의 선언 위치가 변수를 참조하려는 위치보다 뒤에 있는 경우**
  - 전역변수를 사용하기 위해서는 extern을 사용한 참조선언이 필요
    - 동일한 파일에서도 extern을 사용해야 하는 경우가 발생 가능
    - 소스 파일 중간이나 하단에 전역변수를 배치하는 방법은 바람직하지 않음
- **장 단점**
  - 전역변수는 어디에서든지 수정할 수 있으므로 사용이 편한 장점
  - 전역변수에 예상하지 못한 값이 저장
    - 프로그램 어느 부분에서 수정되었는지 알기 어려운 단점
    - 이러한 문제로 전역변수는 가능한 제한적으로 사용하는 것이 바람직

```
...
int main(void)
{
    //하부에 선언한 전역변수를 위한 참조 선언
    extern int gi;
    //extern gi;    //자료형이 없는 문장도 가능
    ...
    printf("gi: %d\n", gi);        //전역변수 gi 기본값
    return 0;
}

//전역변수 선언
int gi;
```

그림 12-6 동일한 파일에서의 extern의 사용

정적 변수의 초기화

- **반드시 상수로만 가능**
  - 왼쪽 소스에서 정적 변수의 초기값에 변수를 대입
    - 초기화 문법 오류가 발생
  - 오른쪽과 같이 상수를 대입

```c
#include <stdio.h>

int a = 1;
static s = a; //오류

int main(void)
{
   int data = 10;
   static value = data; //오류

   return 0;
}
```

```c
#include <stdio.h>

int a = 1;
static s = 1;

int main(void)
{
   int data = 10;
   static value = 10;

   return 0;
}
```

그림 12-11 정적 변수의 초기화 주의

LAB 피보나츠 수의 출력(1)

- **피보나츠의 수는 1, 1로 시작하여 이전 두 수를 더한 수**
  - 즉 5개의 피보나츠 수는 1, 1, 2, 3, 5
  - 표준입력으로 받은 3 이상의 정수를 전역변수 count에 저장한 후
  - 재귀함수인 fibonacci()에서 count-1 개의 피보나츠의 수를 출력
  - 함수 fibonacci()의 매개변수는 (int prev_number, int number)으로 이전 두 정수가 인자, 자기자신을 호출하는 재귀함수
  - 함수 fibonacci()에서 자기 자신이 호출된 수를 저장하는 정적 지역변수 i를 사용하며 초기값은 1로 지정
- **결과**
  - 피보나츠를 몇 개 구할까요?(3 이상) >> 20
  - 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

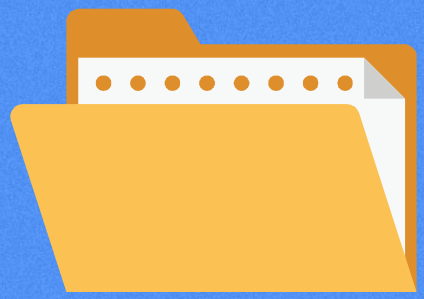
LAB 피보나츠 수의 출력(2)

Lab 12-1 fibonacci.c

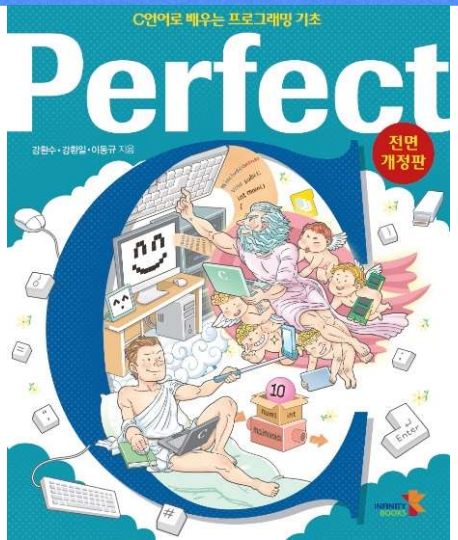
```c
//file fibonacci.c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>

//전역변수
int count;
//함수원형
void fibonacci(int prev_number, int number);

void main() {
   //자동 지역변수
   auto prev_number = 0, number = 1;

   printf("피보나츠를 몇 개 구할까요?(3 이상) >> ");
   //전역변수를 표준입력으로 저장
   scanf("%d", &count);
   if (count <= 2)
      return 0;

   printf("1 ");
   fibonacci(prev_number, number);
   printf("\n");
}

void fibonacci(int prev_number, int number)
{
   //정적 지역변수 i
   static _____________________;

   //전역변수 count와 함수의 정적 지역변수를 비교
   while (i++ < count)
   {
      //지역변수
      int next_num = _________________________________;
      prev_number = number;
      number = next_num;
      printf("%d ", next_num);
      fibonacci(_____________________________);
   }
}
```

정답

```c
static int i = 1;
   int next_num = prev_number + number;
   fibonacci(prev_number, number);
```



02. 정적 변수와 레지스터 변수



기억부류(1)

- **변수 선언의 위치에 따라 변수: 전역과 지역**
- **변수 4가지 기억부류(storage class)**
  - auto, register, static, extern
    - 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정
  - 기억 부류는 키워드 auto, register, static, extern에 의해 구분
    - 자동변수인 auto는 일반 지역변수로 생략 가능
  - 전역 변수 또는 지역 변수
  - auto와 register
    - 지역변수에만 이용이 가능
  - static
    - 지역과 전역 모든 변수에 이용 가능
  - extern
    - 전역변수에만 사용이 가능
    - 컴파일러에게 변수가 이미 어딘가 (주로 다른 파일)에 존재하고 이제 사용하겠다는 것을 알리는 구문에 사용되는 키워드
    - extern이 선언되는 위치에 따라 이 변수의 사용의 범위는 전역 또는 지역으로 한정
  - 기억부류 auto, register, static
    - 새로운 변수의 선언에 사용되는 키워드

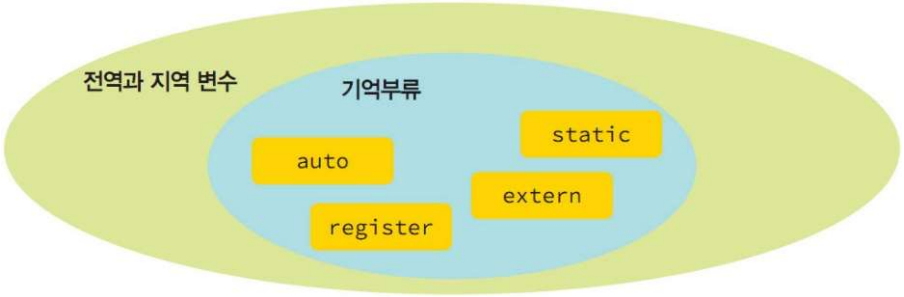


그림 12-7 전역과 지역, 다양한 기억 부류

기억부류(2)

- **기억부류 사용 구문**
  - 변수 선언 문장에서 자료형 앞에 하나의 키워드를 넣는 방식
  - 키워드 extern을 제외하고 나머지 3개의 기억부류의 변수선언에서 초기값을 저장 가능
  - 이미 지역 변수에서 다룬 것처럼 키워드 auto는 지역변수 선언에 사용되며 생략 가능.
  - 함수에 선언된 모든 변수가 auto가 생략된 자동변수

표 12-1 기억부류 종류와 유효 범위

기억부류 종류	전역	지역
auto	×	○
register	×	○
static	○	○
extern	○	×

기억부류 변수 선언

```
기억부류 자료형 변수이름;
기억부류 자료형 변수이름 = 초기값;
```

```
auto int n;
register double yield;
static double data = 5.85;
int age;
extern int input;
```

그림 12-8 기억부류를 사용한 변수선언 구문

키워드 register

- **변수를 연산에 참여 시키려면 다시 CPU 내부에 레지스터에 불러들여 연산을 수행**
- **레지스터 변수**
  - 변수의 저장공간이 일반 메모리가 아니라 CPU 내부의 레지스터(register)에 할당되는 변수
  - 키워드 register를 자료형 앞에 넣어 선언
  - 지역변수에만 이용이 가능
  - 지역변수로서 함수나 블록이 시작되면서 CPU의 내부 레지스터에 값이 저장
    - 함수나 블록을 빠져나오면서 소멸되는 특성
  - CPU내부에 있는 기억장소이므로 일반 메모리보다 빠르게 참조 가능
  - 일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용 불가능
    - 주소연산자 &를 사용하면 문법오류가 발생.

레지스터 변수

예제 registervar.c

- 지역변수인 레지스터 변수의 선언과 사용

연산자 &와 *

- 시스템의 레지스터는 그 수가 한정
  - 레지스터 변수로 선언하더라도 레지스터가 모자라면 일반 지역변수로 할당
- 주로 레지스터 변수는 처리 속도를 증가시키려는 변수에 이용
  - 특히 반복문의 횟수를 제어하는 제어변수에 이용하면 효과적
- 레지스터 변수도 일반 지역변수와 같이 초기값이 저장되지 않으면
  - 쓰레기값이 저장

실습예제 12-4 registervar.c

레지스터 변수의 선언과 사용

```
01  // file: registervar.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  int main(void)
06  {
07     //레지스터 지역변수 선언
08     register int sum = 0;
09
10     //메모리에 저장되는 일반 지역변수 선언
11     int max;
12     printf("양의 정수 입력 >> ");
13     scanf("%d", &max);
14
15     //레지스터 블록 지역변수 선언
16     for (register int count = 1; count <= max; count++)
17        sum += count;
18
19     printf("합: %d\n", sum);
20
21     return 0;
22  }
```

설명

```
08  레지스터 변수 sum 선언하면서 초기값으로 0 저장, 초기값이 없으면 쓰레기값 저장
11  일반 지역변수 max 저장, 초기값이 없으면 쓰레기값 저장
13  max에 표준입력으로 정수 저장
16  레지스터 블록 지역변수 count 선언하면서 초기값으로 1 저장, count는 max까지 반복하면서
    함수 몸체인 sum += count를 실행
17  함수 몸체 sum += count에서 count는 1에서 max까지 변하므로 1에서 max까지의 합이 저장
19  합이 저장된 sum 출력
```

실행결과

```
양의 정수 입력 >> 8
합: 36
```

키워드 static

- **정적변수(static variable)를 선언**
  - 변수 선언에서 자료형 앞에 키워드 static 기술
- **종류**
  - 정적 지역변수(static global variable)와 정적 전역변수(static local variable)로 구분
- **특성**
  - 초기 생성된 이후 메모리에서 제거되지 않으므로
    - 지속적으로 저장값을 유지하거나수정 가능한 특성
    - 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거
  - 초기값
    - 초기값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '₩0' 또는 NULL 값이 저장.
    - 초기화는 단 한번만 수행
      - 한번 초기화된 정적변수는 프로그램 실행 중간에 더 이상 초기화되지 않는 특성
      - 주의할 점은 초기화는 상수로만 가능

```
static int svar = 1;    //정적변수
```

- 전역변수 선언 시 키워드 static을 가장 앞에 붙이면 정적 전역변수가 된다.
  - 정적 전역변수는 참조범위는 선언된 파일에만 한정되며 변수의 할당과 제거는 전역변수 특징을 갖는다.
- 지역변수 선언 시 키워드 static을 가장 앞에 붙이면 정적 지역변수가 된다.
  - 정적 지역변수는 참조범위는 지역변수이면서 변수의 할당과 제거는 전역변수 특징을 갖는다.

그림 12-9 정적변수의 특징

정적 지역변수

예제 neighborvar.c

- 정적 지역변수인 sindex
  - 함수가 종료되어도 메모리에 계속 변수값이 남아 있어 함수 increment()가 호출될 때마다 그 값이 1씩 증가
- 지역변수 aindex
  - 함수 increment()가 호출될 때마다 다시 새롭게 메모리에 할당되고, 함수가 종료되면 메모리에서 제거되어, aindex는 항상 1이 출력

연산자 &와 *

- 정적 지역변수
  - 함수나 블록에서 정적으로 선언되는 변수, 유효 범위는 선언된 블록 내부에서만 참조 가능: 지역변수 특성
  - 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성: 전역변수 특성
- 함수에서 이전에 호출되어 저장된 값을 유지하여 이번 호출에 사용 가능

실습예제 12-5 staticl0cal.c

정적 지역변수와 자동 지역변수의 차이

```
01  // file: staticlocal.c
02  #include <stdio.h>
03
04  void increment(void); //함수원형
05
06  int main(void)
07  {
08     //자동 지역변수
09     for (int count = 0; count < 3; count++)
10        increment();   //3번 함수호출
11  }
12
13  void increment(void)
14  {
15     static int sindex = 1;   //정적 지역변수
16     auto int aindex = 1;   //자동 지역변수
17
18     printf("정적 지역변수 sindex: %2d,\t", sindex++);
19     printf("자동 지역변수 aindex: %2d\n", aindex++);
20  }
```

설명

```
09  for 문에서 사용될 수 있는 자동 지역변수인 count 선언하면서 초기값으로 0 저장,
    count 값이 0, 1, 2로 변하면서 함수 increment()를 실행
10  함수호출 increment()이 세 번 발생
13  함수 increment() 헤더, 매개변수도 없고, 반환값도 없음
15  정적 지역변수 sindex 선언하면서 초기값으로 1 저장, 첫 번째 호출되어 실행될 때 초기값으로
    sindex가 1이 저장되고 함수가 종료되어도 그 변수는 계속 유지되는 특성
16  자동 지역변수 aindex 선언하면서 초기값으로 1 저장, 함수가 호출되어 실행될 때마다 다시
    선언되고 초기값으로 aindex가 1이 저장되며, 함수가 종료되면 이 변수는 메모리에서 사라지므로,
    호출될 때마다 1만 사용됨
18  정적 지역변수 sindex를 출력하고 1 증가시킴, 증가된 sindex는 다음 호출에서 계속 사용되므로,
    2회 호출에서는 2가 출력되고, 3회 호출에서는 3이 출력됨
19  자동 지역변수 aindex를 출력하고 1 증가시키나, 증가된 aindex를 사용할 일이 없음
```

실행결과

```
정적 지역변수 sindex:  1,      자동 지역변수 aindex:  1
정적 지역변수 sindex:  2,      자동 지역변수 aindex:  1
정적 지역변수 sindex:  3,      자동 지역변수 aindex:  1
```

정적 전역변수(1)

예제 staticgvar.c

- 정적 전역변수 svar의 사용

정적 전역변수

- 함수 외부에서 정적으로 선언되는 변수
  - 정적 전역변수는 선언된 파일 내부에서만 참조가 가능한 변수
  - 정적 전역변수는 extern에 의해 다른 파일에서 참조가 불가능
- 일반 전역변수
  - 파일 소스가 다르더라도 extern을 사용하여 참조가 가능

변수 svar는 정적 전역변수로서 초기화는 한 번만 실행되고, 이 파일 내부에서만 이용이 가능한 변수이다. 즉 다른 파일에서는 참조가 불가능하다.

파일 static.c

```
static int svar;

...
   svar++;
```

파일 other.c

```
extern int svar;

...
   svar = 10;
```

링크 시 오류 발생: fatal error LNK1120: 1개의 확인할 수 없는 외부 참조입니다.

그림 12-10 정적 전역변수의 참조 범위

실습예제 12-6 staticgvar.c

정적 전역변수 선언과 사용

```
01  // file: staticgvar.c
02  #include <stdio.h>
03
04  //정적 전역변수 선언
05  static int svar;
06  //전역변수 선언
07  int gvar;
08
09  //함수 원형
10  void increment();
11  void testglobal();
12  //void teststatic();
13
14  int main(void)
15  {
16     for (int count = 1; count <= 5; count++)
17        increment();
18     printf("함수 increment()가 총 %d번 호출되었습니다.\n", svar);
19
20     testglobal();
21     printf("전역 변수: %d\n", gvar);
22     //teststatic();
23  }
24
25  //함수 구현
26  void increment()
27  {
28     svar++;
29  }
```

설명

```
05    키워드 static을 사용하여 정적 전역변수 svar 선언하므로 이 파일에서만 사용 가능하며,
      초기값이 없어도 기본값인 0으로 저장
07    키워드 static이 없으므로 전역변수 gvar 선언하므로 이 프로젝트의 모든 파일에서 사용 가능하며,
      초기값이 없어도 기본값인 0으로 저장

10~11 함수원형
16    변수 count는 자동 지역변수로 for 문 내부에서만 사용이 가능하며, 1에서 5까지 변하면서
      몸체인 함수 increment() 호출
18    정적 전역변수인 savr 호출
20    함수 testglobal() 호출
21    전역변수인 gvar 출력
26    함수 increment() 함수 헤더
28    정적 전역변수인 svar 호출
```

실행결과

```
함수 increment()가 총 5번 호출되었습니다.
전역 변수: 10
```

정적 전역변수(2)

예제 gfunc.c

- 정적 전역변수인 svar 사용 불가능
  - 다른 파일
- 전역변수인 gvar 사용 가능

부작용(side effect)

- 전역변수의 사용은 모든 함수에서 공유할 수 있는 저장공간을 이용할 수 있는 장점
- 어느 한 함수에서 잘못 다루면 모든 함수에 영향을 미치는 단점
- 특히 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용(side effect)의 위험성이 항상 존재
  - 그러므로 가급적이면 전역변수의 사용을 자제하는 것이 좋으며,
  - 부득이 전역변수를 이용하는 경우에는 파일에서만 전역변수로 이용할 수 있는 정적 전역변수를 이용하는 것이 바람직

실습예제 12-7 gfunc.c

정적 전역변수 선언과 사용

```
01  // file: gfunc.c
02
03  void teststatic()
04  {
05     //정적 전역변수는 선언 및 사용 불가능
06     //extern svar;
07     //svar = 5;
08  }
09
10  void testglobal()
11  {
12     //전역변수는 선언 및 사용 가능
13     extern gvar;
14     gvar = 10;
15  }
```

설명

```
03  함수 teststatic() 구현 헤드
06  정적 전역변수 svar는 다른 파일에서 사용 불가하여 실행 시 오류발생
07  정적 전역변수 svar 사용 불가
10  함수 testglobal() 구현 헤드
13  다른 파일에 정의된 전역변수 gvar는 이 파일에서 extern을 사용해 선언 한 후 사용 가능
14  전역변수 gvar 사용 가능하므로 10으로 저장
```

LAB 지역변수와 정적변수의 사용

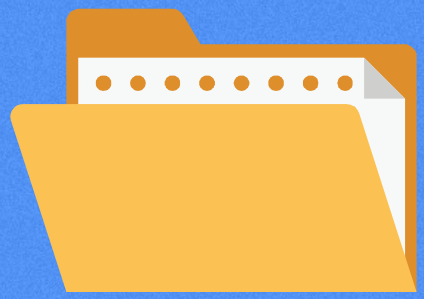
- **함수 main()에서 함수 process()를 세 번 호출**
  - 함수 process()에는 지역변수와 정적 지역변수가 선언
  - 간단한 연산과 함께 출력
- **다음 소스를 보고 출력결과를 예상**
  - 함수 process()의 지역변수 x, 정적 지역변수 sx

Lab 12-2 static.c

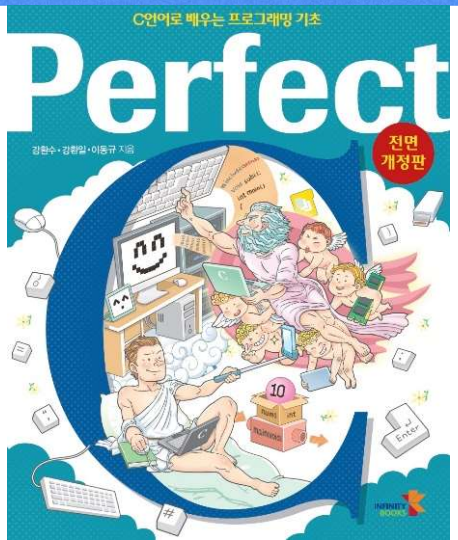
```c
//file: static.c
#include <stdio.h>

void process();

int main()
{
   process();
   process();
   process();

   return 0;
}

void process()
{
   //정적 변수
   static int sx;
   //지역 변수
   int x = 1;

   printf("%d %d\n", x, sx);

   x += 3;
   sx += x + 3;
}
```

정답

```
1 0
1 7
1 14
```



03. 메모리 영역과 변수 이용



메모리 영역(1)

- **데이터, 스택, 힙 영역**
  - 메모리 영역은 변수의 유효범위(scope)와 생존기간(life time)에 결정적 역할
- **변수는 기억부류(storage class)에 따라 할당되는 메모리 공간이 달라짐**
  - 기억부류는 변수의 유효범위(scope)와 생존기간(life time)을 결정
  - 기억부류는 변수의 저장공간의 위치가 데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역인지도 결정하며, 초기값도 결정

그림 12-12 기억부류

메모리 영역(2)

- **데이터 영역**
  - 전역변수와 정적변수가 할당되는 저장공간
  - 메모리 주소가 낮은 값에서 높은 값으로 저장 장소가 할당
  - 프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리 영역이 확보
- **힙 영역**
  - 동적 할당(dynamic allocation)되는 변수가 할당되는 저장공간
  - 데이터 영역과 스택 영역 사이에 위치
- **스택 영역**
  - 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간
  - 힙 영역과 스택영역은 프로그램이 실행되면서 영역 크기가 계속적으로 변함
  - 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당
    - 함수 호출과 종료에 따라 메모리가 할당되었다가 다시 제거되는 작업이 반복

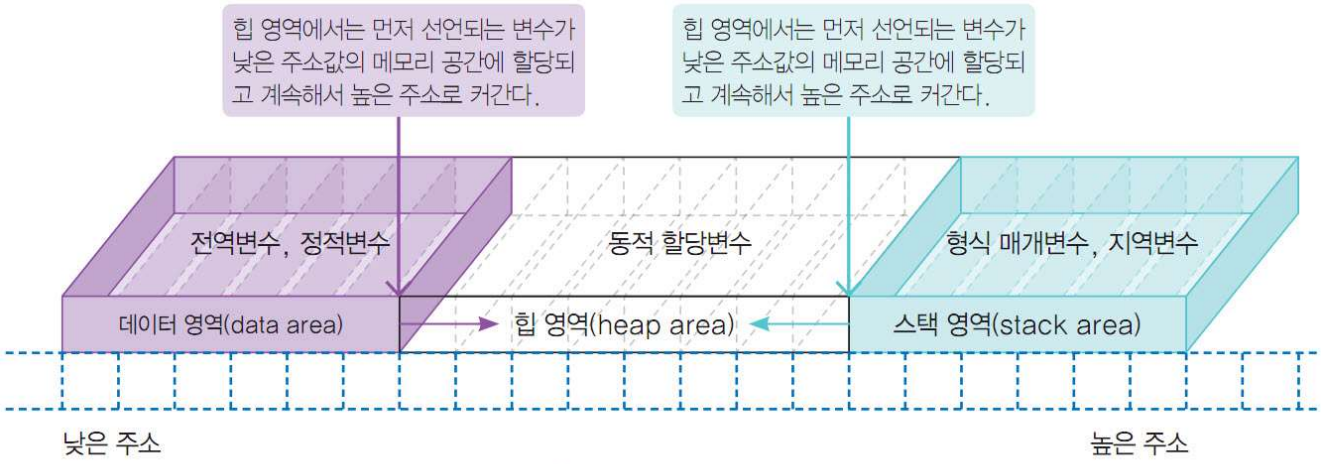


그림 12-13 메모리 영역

변수 이용 기준

- **전역변수의 사용을 자제하고 지역변수를 주로 이용**
  - ❶ 레지스터 변수
    - 실행 속도를 개선하고자 하는 경우
  - ❷ 정적 지역변수
    - 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장
  - ❸ 정적 전역변수
    - 해당 파일 내부에서만 변수를 공유하고자 하는 경우
  - ❹ 전역변수
    - 프로그램의 모든 영역에서 값을 공유하고자 하는 경우
    - 가능하면 전역 변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수 있으며 발생할 수 있는 프로그램 문제를 줄일 수 있음
- **전역변수와 지역변수를 정리**
  - 변수 할당 메모리 영역에 따라 변수의 할당과 제거의 시기가 결정
  - 데이터 영역의 전역변수와 정적 변수
    - 프로그램 시작 시 메모리가 할당되고, 프로그램 종료 시 메모리에서 제거
  - 스택 영역과 레지스터에 할당되는 자동 지역변수와 레지스터 변수
    - 함수 또는 블록 시작 시 메모리가 할당되고, 함수 또는 블록 종료 시 메모리에서 제거

표 12-2 변수의 종류

선언위치	상세 종류	키워드		유효범위	기억장소	생존기간
전역	전역 변수	참조선언	extern	프로그램 전역	메모리 (데이터 영역)	프로그램 실행 시간
	정적 전역변수	static		파일 내부		
지역	정적 지역변수	static		함수나 블록 내부		
	레지스터 변수	register			레지스터	함수 또는 블록 실행 시간
	자동 지역변수	auto (생략가능)			메모리 (스택 영역)	

변수의 유효범위와 초기값

- **변수의 유효 범위는 O, X로 구분**
  - 그 지역에서 참조 가능하면 O, 아니면 X로 구분

표 12-3 변수의 유효 범위

구분	종류	메모리할당 시기	동일 파일 외부 함수에서의 이용	다른 파일 외부 함수에서의 이용	메모리제거 시기
전역	전역변수	프로그램시작	○	○	프로그램종료
	정적 전역변수	프로그램시작	○	×	프로그램종료
지역	정적 지역변수	프로그램시작	×	×	프로그램종료
	레지스터 변수	함수(블록) 시작	×	×	함수(블록) 종료
	자동 지역변수	함수(블록) 시작	×	×	함수(블록) 종료

- **변수의 종류에 따라 초기값**
  - 초기 값 저장 문장의 실행 시점
  - 초기값이 명시적으로 지정되지 않을 경우 자동으로 지정되는 기본 초기값

표 12-4 변수의 초기값

지역, 전역	종류	자동 저장되는 기본 초기값	초기값 저장
전역	전역변수	자료형에 따라 0이나 '\0' 또는 NULL 값이 저장됨.	프로그램 시작 시
	정적 전역변수		
지역	정적 지역변수		
	레지스터 변수	쓰레기값이 저장됨.	함수나 블록이 실행될 때마다
	자동 지역변수		

전역변수와 지역변수(1)

예제 storageclass.c

- 소스 storageclass.c와 out.c로 구성되는 프로그램

전역/지역, 정적 변수

- 전역변수 global, sglobal
- 지역변수 fa, fs의 선언과 사용을 알아보는 프로그램
- sglobal은 정적 전역변수이므로 외부 파일에서는 참조할 수 없으며
- 정적 지역변수인 fs는 함수가 종료되더라도 그전에 저장된 값이 계속 유지

실습예제 12-8 storageclass.c

전역변수와 지역변수의 선언과 참조

```
01  // file: storageclass.c
02  #include <stdio.h>
03
04  void infunction(void);
05  void outfunction(void);
06
07  /* 전역변수*/
08  int global = 10;
09  /* 정적 전역변수*/
10  static int sglobal = 20;
11
12  int main(void)
13  {
14     auto int x = 100;   /* main() 함수의 자동 지역변수*/
15
16     printf("%d, %d, %d\n", global, sglobal, x);
17     infunction(); outfunction();
18     infunction(); outfunction();
19     infunction(); outfunction();
20     printf("%d, %d, %d\n", global, sglobal, x);
21
22     return 0;
23  }
24
25  void infunction(void)
26  {
27     /* infunction() 함수의 자동 지역변수*/
28     auto int fa = 1;
29     /* infunction() 함수의 정적 지역변수*/
30     static int fs;
31
32     printf("\t%d, %d, %d, %d\n", ++global, ++sglobal, fa, ++fs);
33  }
```

설명

```
08  전역변수 global 선언하면서 10으로 초기화, 함수 외부이므로 전역변수이고 모든 프로젝트에서
    사용 가능하나 다른 파일이나 이 위치 위에서 사용하려면 extern int global; 선언이 필요
10  정적 전역변수 sglobal 선언하면서 20으로 초기화, 함수 외부이므로 전역변수이나 이 파일의
    이 위치 이하에서 사용 가능
14  변수 x는 자동 지역변수로, 초기화로 100 저장, 함수 main()에서만 사용 가능
16  변수 전역변수 global, 정적 전역변수 sglobal, 지역변수 x를 출력
17  함수 infunction()과 outfunction()을 호출
20  변수 전역변수 global, 정적 전역변수 sglobal, 지역변수 x를 출력
25  함수 infunction()의 헤더로 26 행에서 33행까지 구현
28  변수 fa는 자동 지역변수로, 초기화로 1 저장, 함수 infunction()에서만 사용 가능
30  변수 fs는 정적 지역변수로, 첫 번째 호출에서 초기화가 없으므로 기본값인 00l 저장되며,
    이후에는 함수가 종료돼도 제거되지 않고 값이 계속 유지되며, 함수 infunction()에서만 사용 가능
32  탭을 출력한 후, 전역변수 ++global, 정적 전역변수 ++sglobal, 자동 지역변수 fa,
    정적 지역변수 ++fs 출력
```

실행결과

```
10, 20, 100
        11, 21, 1, 1
                    12
        13, 22, 1, 2
                    14
        15, 23, 1, 3
                    16
16, 23, 100
```

탭 이후에 출력되는 행은 함수 infunction()에서 출력되는 부분

탭이 2번 출력되고 이후에 출력되는 행은 함수 outfunction()에서 출력되는 부분

전역변수와 지역변수(2)

예제 storageclass.c

- 소스 storageclass.c와 out.c로 구성되는 프로그램

전역/지역, 정적 변수

- 정적 전역변수 sglobal은 참조 불가능

실습예제 12-9 out.c

외부 파일의 전역변수 참조

```
01  // file:out.c
02  #include <stdio.h>
03
04  void outfunction()
05  {
06     extern int global, sglobal;
07
08     printf("\t\t%d\n", ++global);
09
10     //외부 파일에 선언된 정적 전역변수이므로 실행 시 오류
11     //printf("%d\n", ++sglobal);
12  }
```

설명

06 외부 파일 storageclass.c 에 선언된 전역 변수 global과 sglobal을 사용하려는 선언

08 탭을 두 번 출력한 후, 전역 변수 global을 하나 증가시켜 출력

11 외부 파일에 선언된 정적 전역변수 sglobal 은 참조할 수 없으므로 실행 시 오류

LAB 은행계좌의 입출금 구현

- **은행 계좌의 입출금을 구현**
  - 전역변수 total
  - 두 함수 save()와 withdraw() 구현
  - 정적 지역 변수 amount를 사용
  - 몇 개의 입출금에 대해 출력
    - 전역변수 total에는 초기 금액과 계좌 잔고가 저장
  - 함수 save()와 withdraw()는 각각 매개변수 금액의 입출금을 구현
    - 정적 지역변수 amount를 사용하여 총입금액과 총출금액을 관리하여 출력

Lab 12-3 bank.c

전역변수와 지역변수의 선언과 참조

```c
// file: bank.c
#include <stdio.h>

//전역변수
_____________________________;

//입금 함수원형
void save(int);
//출금 함수원형
void withdraw(int);

int main(void)
{
   printf(" 입금액   출금액    총입금액    총출금액      잔고\n");
   printf("=============================================\n");
   printf("%46d\n", total);
   _____________________________;
   withdraw(30000);
   save(60000);
   withdraw(20000);
   printf("=============================================\n");

   return 0;
}

//입금액을 매개변수로 사용
void save(int money)
{
   //총입금액이 저장되는 정적 지역변수
   _____________________________;
   total += money;
   amount += money;
   printf("%7d %17d %20d\n", money, amount, total);
}

//출금액을 매개변수로 사용
void withdraw(int money)
{
   //총출금액이 저장되는 정적 지역변수
   static int amount;
   _____________________________;
   amount += money;
   printf("%15d %20d %9d\n", money, amount, total);
}
```

정답

```c
int total = 10000;
 save(50000);
 static int amount;
 total -= money;
```

Thank you