제 10 장 변수 유효범위

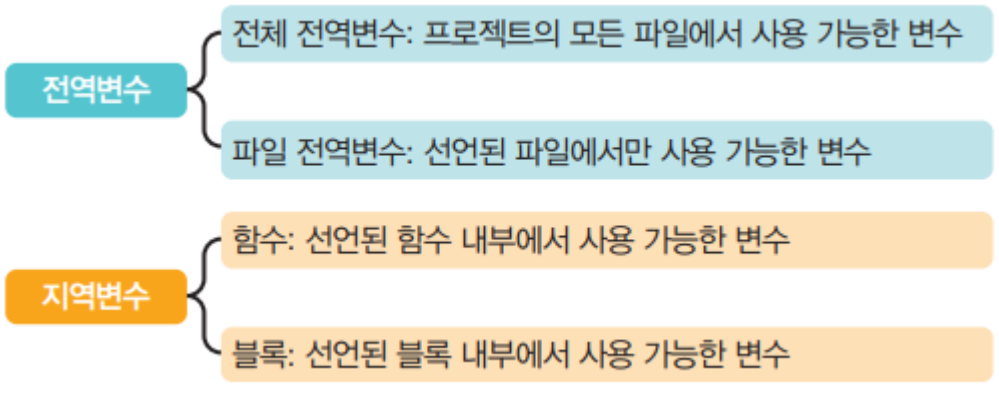
01 전역변수와 지역변수
02 정적 변수와 레지스터 변수
03 메모리 영역과 변수 이용

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

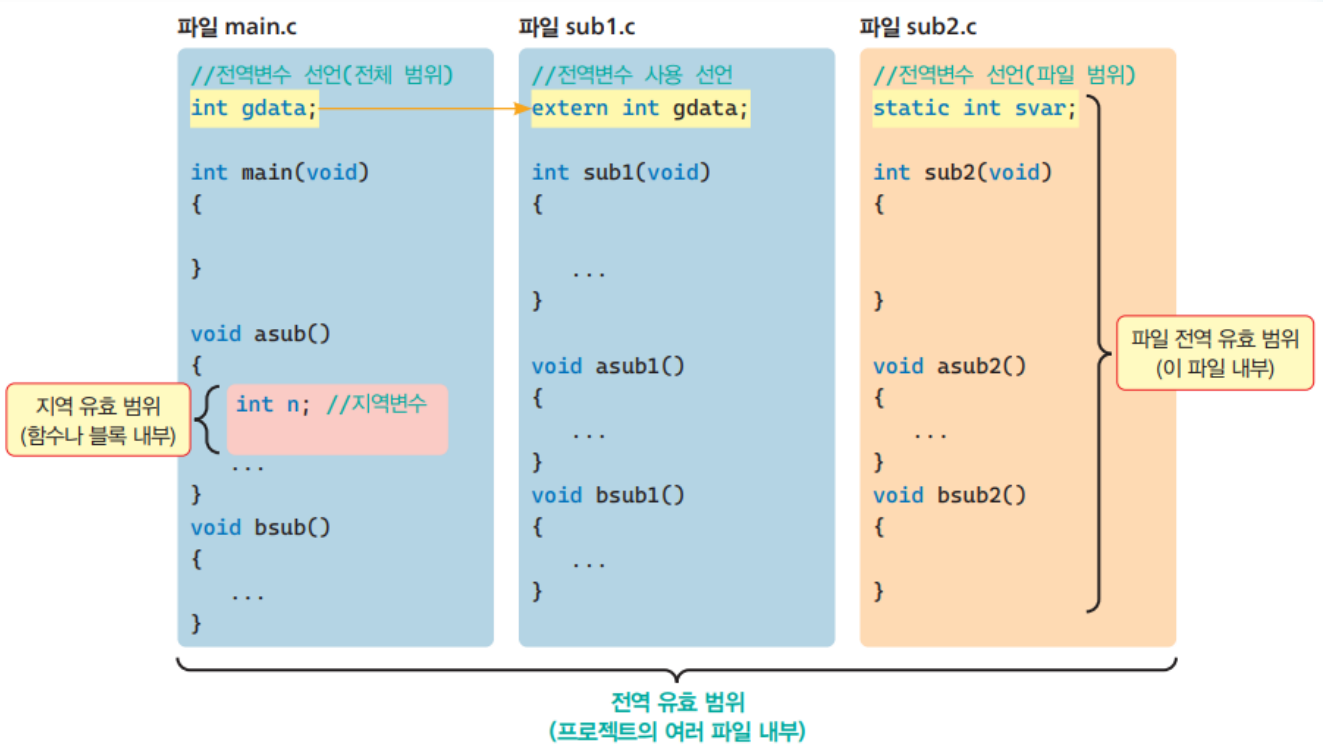
변수 범위와 지역변수

- **변수의 유효 범위(scope)**
  - 선언된 변수가 사용될 수 있는 범위
  - 지역 유효 범위(local scope)와 전역 유효 범위(global scope)로 나눔
- **지역 유효 범위**
  - 함수 또는 블록 내부에서 선언된 변수는 선언된 이후의 함수나 블록에서 사용
- **전역 유효 범위**
  - 프로젝트나 파일에서 사용 가능한 범위
  - 하나의 파일에서만 변수의 참조가 가능할 수 있으며
  - 또는 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능



지역과 전역

- **프로젝트**
  - 파일 main.c, 그리고 파일 sub1.c와 sub2.c, 3개의 소스 파일로 구성
- **지역변수**
  - 파일 main.c의 함수 asub()에서 선언된 변수 n은 함수 내부에서만 사용
  - 모든 함수 내부에서 선언된 변수는 모두 지역 변수
- **전체 전역변수**
  - 파일 main.c의 상단에 선언된 gdata는 프로젝트 전체 파일에서 사용
  - gdata를 파일 sub1.c에서 사용하려면
    - **먼저 extern int gdata로 선언이 필요**
- **파일 전역변수**
  - 파일 sub2.c의 상단에 선언된 svar는 파일 sub2.c에서만 사용



지역변수는 함수 또는 블록에서 선언된 변수

- **지역변수는 내부변수 또는 자동변수(automatic variables)라고도 부름**
  - 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능
    - **함수의 매개변수도 함수 전체에서 사용 가능한 지역변수**
  - 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의
  - 할당되는 메모리 영역을 스택(stack), 선언된 부분에서 자동으로 생성되고
    - **함수나 블록이 종료되는 순간 메모리에서 자동으로 제거**
      - **이러한 이유에서 지역변수는 자동변수(automatic variable)**
    - **지역변수 선언에서 자료형 앞에 키워드 auto가 사용 가능**
      - **키워드 auto는 생략 가능해 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

오류 발생: error C2065: 'sum' : 선언되지 않은 식별자입니다.

지역변수 선언과 사용

- **for 문 블록에서 선언된 지역변수 m, sum**
  - for 문 블록에서만 사용이 가능
  - 블록 외부에서 참조가 불가능
- **매개변수 param**
  - 함수 sub() 내부에서 모두 사용 가능

실습예제 10-1 Prj01 01localvar.c 지역변수 선언과 사용 난이도: ★

```c
#include <stdio.h>

void sub(int param); //함수 원형

int main(void)
{
   auto int n = 10; //지역변수 선언
   printf("%d\n", n);

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

   printf("%d\n", n); //n 참조 가능
   //printf("%d %d\n", m, sum); //m, sum 참조 불가능

   sub(20); //함수호출

   return 0;
}

void sub(int param) //매개변수인 param도 지역변수로 사용
{
   auto int local = 100; //지역변수 local
   printf("\t%d %d\n", param, local); //param과 local 참조 가능
   //printf("%d\n", n); //n 참조 불가능
}
```

지역변수 n은 함수 main 내부에서만 사용 가능한 지역변수

지역변수 m, sum은 for 문 내부에서만 사용 가능한 블록 지역변수

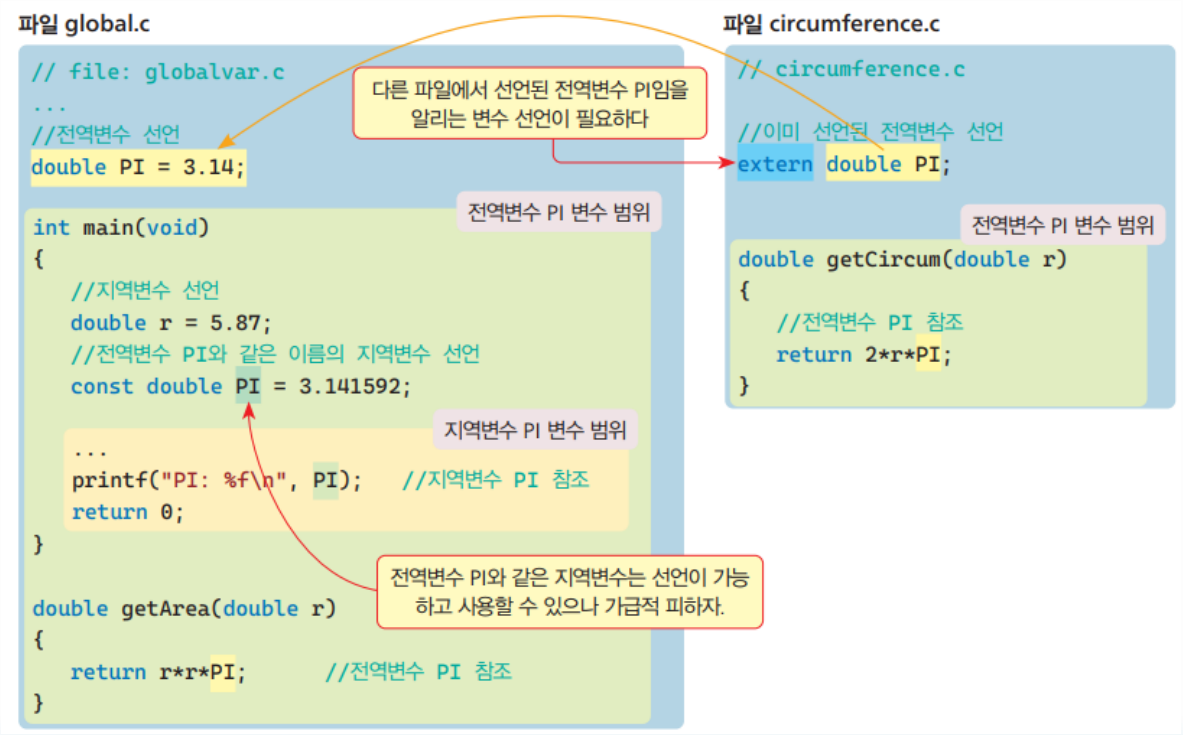
매개변수 param과 지역변수 local 모두 함수 sub 내부에서만 사용 가능

결과

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

전역변수(global variable)

- **함수 외부에서 선언되는 변수**
  - 전역변수는 외부변수라고도 부름
  - 모든 함수나 블록에서 참조 가능
  - 초기 값이 자료형에 맞는 0으로 저장
  - 함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언 가능
    - **함수내부나 블록에서 그 이름을 참조하면 지역변수로 인식**
- **프로젝트의 다른 파일에서도 참조가 가능**
  - 키워드 extern을 사용하여 이미 다른 파일에서 선언된 전역변수임을 선언
    - **extern 참조선언 구문에서 자료형 double 등은 생략 가능**
    - **이미 존재하는 전역변수의 유효 범위를 확장**



전역변수와 지역변수의 선언과 사용

실습예제 10-2 | Prj02 | 02-1global.c, 02-2circumference.c | 전역변수와 지역변수의 선언과 사용 | 난이도: ★

소스코드 02-1global.c

```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
}
```

전역변수 PI 선언하여 3.14 저장, 변수 PI의 유효 범위는 이 파일 및 프로젝트 전체

전역변수 gi 선언하여 초기값을 대입하지 않았으나 기본값인 0이 저장되고 변수 gi의 유효 범위는 이 파일 및 프로젝트 전체

지역변수 PI 선언, 그러나 이 변수는 전역변수 PI와 이름이 충돌, 함수 내부에서 지역변수가 우선하므로 전역변수는 참조가 불가능

소스코드 02-2circumference.c

```c
//이미 외부에서 선언된 전역변수임을 알리는 선언
extern double PI;

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

외부 파일에서 선언된 전역변수 PI를 사용하려는 문장

r은 매개변수이며, PI는 외부 파일에서 선언된 전역변수

결과

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

전역변수 gi는 초기값이 저장되지 않았으나 자동으로 0이 저장됨.

LAB
1에서 100 사이의 난수 알아 맞히기를 두 파일로 구현

- **전역변수 guess**
  - 시스템이 정한 정답이 저장되는 전역변수
  - 파일 lab1numguess.c에서 선언,
    - **파일 lab2testnum.c에서도 사용**
- **지역변수 input**
  - 입력한 정수가 저장되는 함수 main()의 지역변수

```
1에서 100 사이에서 한 정수가 결정되었습니다.
이 정수는 무엇일까요? 입력해 보세요. : 50
입력한 수보다 작습니다. 다시 입력하세요. : 25
입력한 수보다 작습니다. 다시 입력하세요. : 13
입력한 수보다 큽니다. 다시 입력하세요. : 19
입력한 수보다 큽니다. 다시 입력하세요. : 22
입력한 수보다 큽니다. 다시 입력하세요. : 23
정답입니다.
```

Lab 10-1 | lab1-1numguess.c, lab1-2testnum.c | 난이도: ★★

소스코드 | lab1-1numguess.c

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#include <stdlib.h> //rand(), srand()를 위한 헤더파일 포함
#include <time.h>   //time()을 위한 헤더파일 포함

#define MAX 100

int guess; //정답인 전역변수 선언

int main(void)
{
    [          ] //지역변수 선언

    srand((long)time(NULL));
    guess = rand() % MAX + 1; //정답 지정

    printf("1에서 %d 사이에서 한 정수가 결정되었습니다.\n", MAX);
    printf("이 정수는 무엇일까요? 입력해 보세요. : ");

    while (scanf("%d", &input)) {
       switch (testNum(input))
       {
       [          ]
          puts("정답입니다.");
          break;
       case -1:
          printf("입력한 수보다 작습니다. 다시 입력하세요. : ");
          break;
       case 1:
          printf("입력한 수보다 큽니다. 다시 입력하세요. : ");
          break;
       }
    }

    return 0;
}
```

소스코드 | lab1-2testnum.c

```c
int testNum(int input)
{
    [          ] //전역변수 선언

    int result = 0;
    if (input > guess)
       result = -1;
    else if (input < guess)
       result = 1;
    else
       result = 0;

    return result;
}
```

다른 파일에서 선언된 전역변수 guess를 선언하여 사용

정답

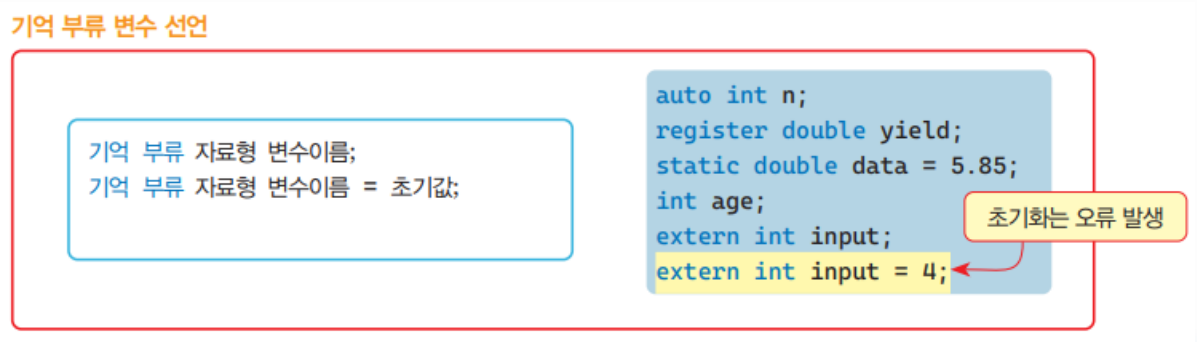
```c
int input; //지역변수 선언
   case 0:
extern guess; //전역변수 선언
```

변수의 4가지 기억 부류

- **auto, register, static, extern**
  - 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정
  - 자동변수인 auto는 일반 지역변수로 생략될 수 있으므로 주의가 필요
  - auto와 register는 지역변수에만 이용이 가능
  - static은 지역과 전역 모든 변수에 이용 가능
  - extern은 전역변수에만 사용이 가능

표 10-1 기억 부류 종류와 유효 범위

기억 부류 종류	전역	지역
auto	X	O
register	X	O
static	O	O
extern	O	X



레지스터 변수

- **레지스터가 모자라면 일반 지역변수로 할당**
  - 레지스터 변수는 처리 속도를 증가시키려는 변수에 이용
  - 특히 반복문의 횟수를 제어하는 제어변수에 이용하면 효과적

실습예제 10-3 | Prj03 | 03regvar.c | 레지스터 변수의 선언과 사용 | 난이도: ★

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

레지스터 변수 sum 선언하면서 초기값으로 0 저장, 초기값이 없으면 쓰레기 값 저장

레지스터 블록 지역변수 count 선언하면서 초기값으로 1 저장, count는 max까지 반복하면서 함수 몸체인 sum += count를 실행

결과

```
양의 정수 입력 >> 8
합: 36
```

정적 변수와 static

- **정적변수**
  - 정적 지역변수(static global variable)
  - 정적 전역변수(static local variable)
  - 초기 생성된 이후 메모리에서 제거되지 않으므로 지속적으로 저장 값을 유지하거나 수정할 수 있는 특성
  - 프로그램이 시작되면 메모리에 할당되고, 프로그램이 종료되면 메모리에서 제거
  - 초기 값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '₩0' 또는 NULL 저장
  - 초기화는 단 한번만 수행
    - **한번 초기화된 정적변수는 프로그램 실행 중간에 더 이상 초기화되지 않는 특성**
      - **주의할 점은 초기화는 상수로만 가능**
- **정적 지역변수**
  - 함수나 블록에서 정적으로 선언되는 변수
  - 선언된 블록 내부에서만 참조 가능
  - 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리
  - 변수 유효 범위(scope)는 지역변수와 같으나
    - **할당된 저장공간은 프로그램이 종료되어야 메모리에서 제거**
    - **즉 함수에서 이전에 호출되어 저장된 값을 유지하여 이번 호출에 사용**

정적 지역변수와 자동 지역변수의 차이

실습예제 10-4 | Prj04 | 04slocal.c | 정적 지역변수와 자동 지역변수의 차이 | 난이도: ★

```c
#include <stdio.h>

void reset(void);          //함수원형
void count(void);          //함수원형

int main(void)
{
                           //자동 지역변수
   for (int i = 1; i <= 5; i++)
   {
      reset();
      count();
   }
}

void reset(void)                    //매번 1을 지정
{
   auto int n = 1;                  //자동 지역변수
   printf("자동 지역변수 n: %2d\n", n);
   n++;
}

void count(void)                    //이전 값을 계속 누적
{
   static int s = 10;               //정적 지역변수
   printf("\t정적 지역변수 s: %2d\n", s);
   s++;
}
```

자동 지역변수 n을 선언하면서 초기값으로 1 저장, 함수가 호출되어 실행될 때마다 다시 선언되고 초기값으로 n에 1이 저장되며, 함수가 종료되면 이 변수는 메모리에서 사라지므로, 호출될 때마다 1만 사용됨

정적 지역변수 s를 선언하면서 초기값으로 10 저장, 첫 번째 호출되어 실행될 때 초기값으로 s에 10이 저장되고 함수가 종료되어도 그 변수는 계속 유지되는 특성

결과

```
자동 지역변수 n:  1
        정적 지역변수 s: 10
자동 지역변수 n:  1
        정적 지역변수 s: 11
자동 지역변수 n:  1
        정적 지역변수 s: 12
자동 지역변수 n:  1
        정적 지역변수 s: 13
자동 지역변수 n:  1
        정적 지역변수 s: 14
```

LAB 지역변수와 정적 변수의 사용

- 출력 결과를 예상

Lab 10-2 Lab2svar.c 난이도: ★★

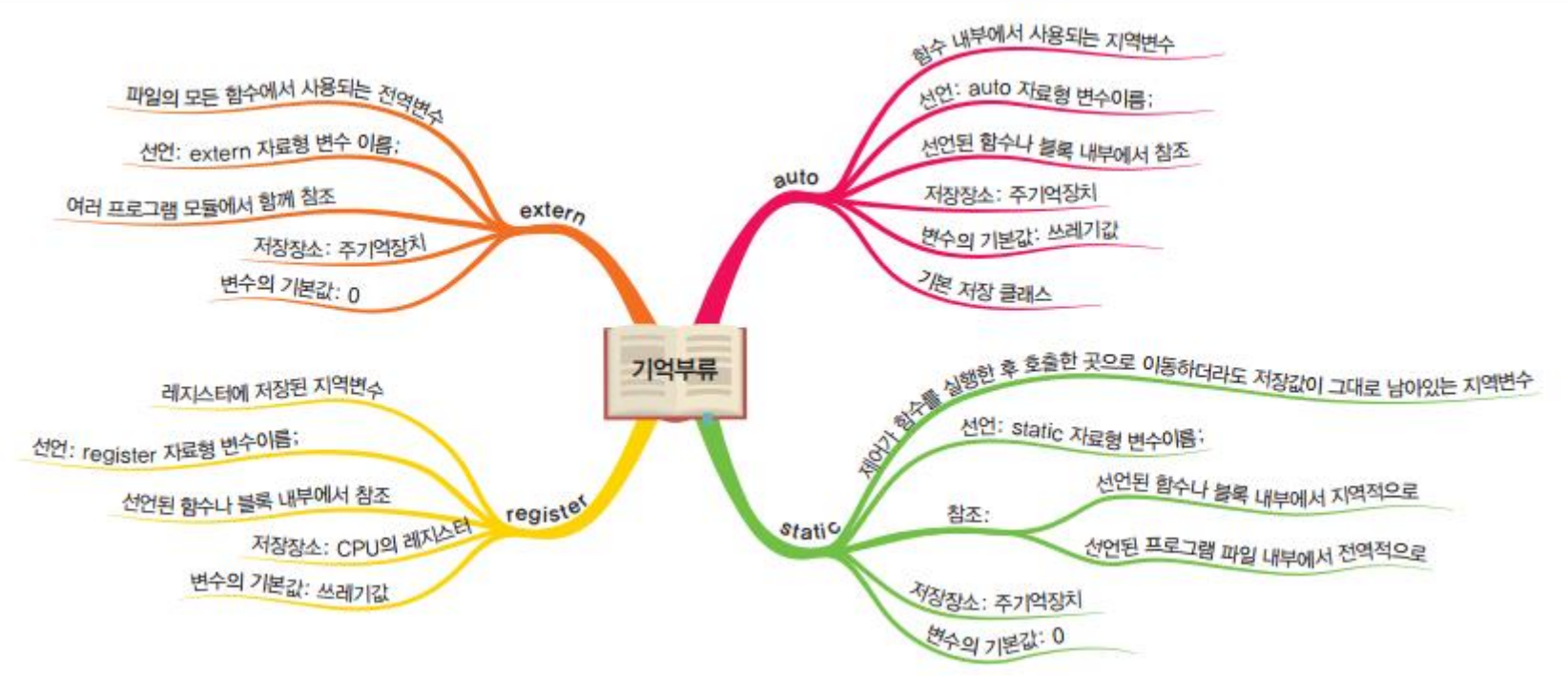
```c
#include <stdio.h>

void process();

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

   return 0;
}

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

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

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

정답

```
1 0
1 5
1 10
```

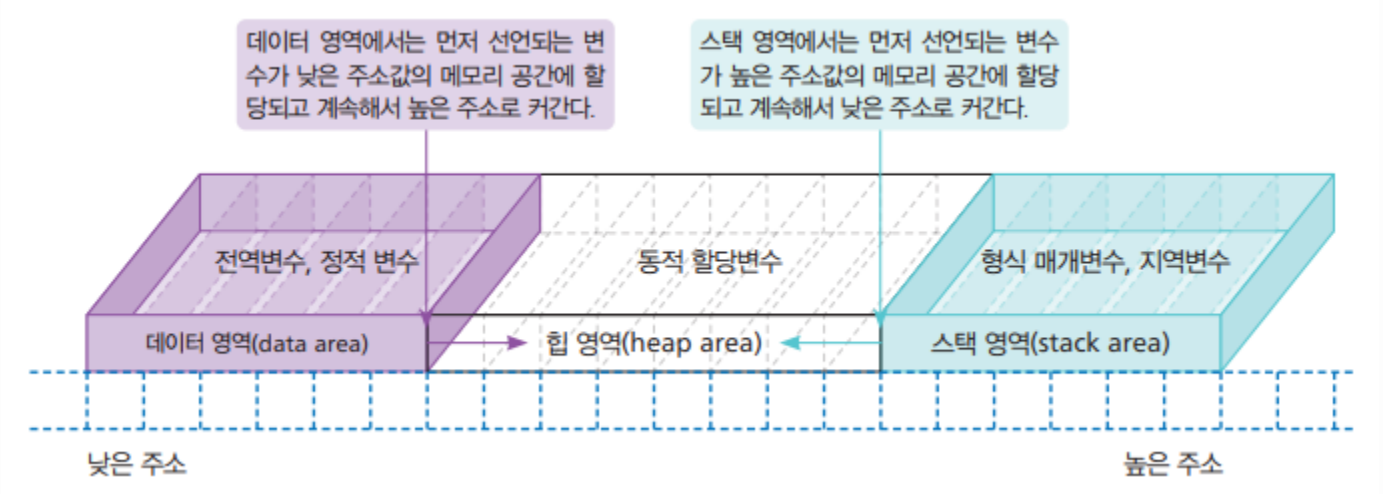
데이터, 스택, 힙 영역

- **메인 메모리의 영역은 프로그램 실행 과정에서**
  - 데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역 세 부분으로 구분
- **기억부류**
  - 변수의 유효범위(scope)와 생존기간(life time)을 결정
  - 변수의 저장공간의 위치가 데이터(data) 영역, 힙(heap) 영역, 스택(stack) 영역인지 결정
  - 초기 값도 결정



메모리 영역

- **데이터 영역**
  - 전역변수와 정적변수가 할당되는 저장공간
- **힙 영역**
  - 동적 할당(dynamic allocation)되는 변수가 할당되는 저장공간
- **스택 영역**
  - 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간



변수 활용 기준

- **일반적으로 전역변수의 사용을 자제하고 지역변수를 주로 이용**
  - ① 실행 속도를 개선하고자 하는 경우
    - **제한적으로 특수한 지역변수인 레지스터 변수를 이용**
  - ② 함수나 블록 내부에서 함수나 블록이 종료되더라도 계속적으로 값을 저장할 경우
    - **정적 지역변수를 이용**
  - ③ 해당 파일 내부에서만 변수를 공유하고자 하는 경우
    - **정적 전역변수를 이용**
  - ④ 프로그램의 모든 영역에서 값을 공유하고자 하는 경우
    - **전역변수를 이용**
    - **가능하면 전역변수의 사용을 줄이는 것이 프로그램의 이해를 높일 수 있으며**
    - **발생할 수 있는 프로그램 문제를 줄일 수 있음**

<table>
<tr><th>선언 위치</th><th>상세 종류</th><th colspan="2">키워드</th><th>유효 범위</th><th>기억 장소</th><th>생존 기간</th></tr>
<tr><td rowspan="2">전역</td><td>전역변수</td><td>참조선언</td><td>extern</td><td>프로그램 전역</td><td rowspan="2">메모리
(데이터 영역)</td><td rowspan="2">프로그램
실행 시간</td></tr>
<tr><td>정적 전역변수</td><td colspan="2">static</td><td>파일 내부</td></tr>
<tr><td rowspan="3">지역</td><td>정적 지역변수</td><td colspan="2">static</td><td rowspan="3">함수나
블록 내부</td><td rowspan="2">레지스터</td><td rowspan="2">함수 또는 블록
실행 시간</td></tr>
<tr><td>레지스터 변수</td><td colspan="2">register</td></tr>
<tr><td>자동 지역변수</td><td colspan="2">auto(생략가능)</td><td>메모리
(스택 영역)</td></tr>
</table>

변수의 종류에 따른 생존 기간과 유효 범위

- **전역변수와 정적 변수**
  - 모두 생존 기간이 프로그램 시작 시에 생성되어 프로그램 종료 시에 제거
- **자동 지역변수와 레지스터 변수**
  - 함수가 시작되는 시점에서 생성되어 함수가 종료되는 시점에서 제거

구분	종류	메모리 할당 시기	동일 파일 외부 함수에서의 이용	다른 파일 외부 함수에서의 이용	메모리 제거 시기
전역	전역변수	프로그램 시작	O	O	프로그램 종료
	정적 전역변수	프로그램 시작	O	X	프로그램 종료
지역	정적 지역변수	프로그램 시작	X	X	프로그램 종료
	레지스터 변수	함수(블록) 시작	X	X	함수(블록) 종료
	자동 지역변수	함수(블록) 시작	X	X	함수(블록) 종료

- **초기 값**

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

전역변수와 지역변수의 선언과 참조

- 전역변수 g, sg 지역변수 fa, fs의 선언과 사용

Prj06	06-1sclass.c 06-2out.c	전역변수와 지역변수의 선언과 참조

06-1sclass.c

```c
#include <stdio.h>

void in(void);
void out(void);

/* 전역변수 */
int g = 10;
/* 정적 전역변수 */
static int sg = 20;

int main(void)
{
   auto int a = 100;    /* main() 함수의 자동 지역변수 */

   printf("%d %d %d\n", g, sg, a);
   in(); out();
   in(); out();
   in(); out();
   printf("%d %d %d\n", g, sg, a);

   return 0;
}

void in(void)
{
   /* in() 함수의 자동 지역변수 */
   auto int fa = 1;
   /* in() 함수의 정적 지역변수 */
   static int fs;

   printf("\t%d %d %d %d\n", ++g, ++sg, ++fa, ++fs);
}
```

전역변수 g를 선언하면서 10으로 초기화, 전역변수는 프로젝트 어디에서나 사용 가능하나 다른 파일에서 사용하려면 extern int g; 선언이 필요

변수 fs는 정적 지역변수로, 첫 번째 호출에서 초기화가 없으므로 기본값인 0이 저장되며, 이후에는 함수가 종료돼도 제거되지 않고 값이 계속 유지되며 함수 in()에서만 사용 가능

06-2out.c

```c
#include <stdio.h>

void out()
{
   extern int g, sg;

   printf("\t\t\t%d\n", ++g);

   //외부 파일에 선언된 정적 전역변수이므로 실행 시 오류
   //printf("%d\n", ++sg);
}
```

다른 파일 06-1sclass.c에 선언된 전역변수 g와 sg를 사용하려는 선언

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

탭 이후에 출력되는 행은 함수 in()에서 출력되는 부분

이 열에 출력되는 행은 함수 out()에서 출력되는 부분

Lab 은행계좌의 입출금 구현

- **전역변수 total**
  - 전역변수 total에는 초기 금액과 계좌 잔고가 저장
- **두 함수 save()와 withdraw()의 정적 지역변수 amount를 사용**
  - 매개변수 금액의 입출금을 구현
  - 정적 지역변수 amount를 사용하여 총입금액과 총출금액을 관리하여 출력

Lab3bank.c 난이도: ★★

```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);
}
```

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

감사합니다.