12. 변수의 유효범위

9주차

변수의 유효 범위(scope)

- 유효 범위 : 변수의 참조가 유효한 범위
- 변수의 유효 범위 구분
  - 지역 유효 범위(local scope) : 함수 또는 블록 내부에서 선언되어 해당 블록 내부에서만 변수의 참조가 가능
    - 지역변수 (내부변수, 자동(auto)변수)
    - 함수의 매개변수도 함수 전체에서 사용 가능한 지역변수
    - 선언 후 초기화하지 않으면 쓰레기값이 저장되므로 주의
    - 지역변수는 stack에 생성되었다가 블록 종료시 자동 제거
  - 전역 유효 범위(global scope) : 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능
    - 전역변수 (외부변수)
    - 지역변수와 동일한 이름의 전역변수는 참조 불가능 (이렇게 쓰지 말자!)
    - static으로 선언하면 파일 내에서만 참조 가능. (반드시 상수로만 초기화 가능!)
    - 다른 파일의 전역변수를 사용하려면 extern 선언 사용
보통의 선언문 앞에 extern 붙이고 초기값 주지 않기.
    - 프로그램 어디에서든지 수정할 수 있으므로 사용이 편하나
파일간 경계가 불명확해지므로 사용을 지양!

지역 변수 예제 (localvar.c)

```
#include <stdio.h>
void sub(int param);
int main(void) {
    auto int n = 10;
    printf("%d₩n", n);

    for (int m = 0, sum = 0; m<3; m++) {
        sum += m;
        printf("₩t%d %d₩n", m, sum);
    }
    printf("%d₩n", n);
    //printf("%d %d₩n", m, sum);
    sub(20);
    return 0;
}
void sub(int param) {
    auto int local = 100;
    printf("₩t%d %d₩n", param, local); //param과 local 참조 가능
    //printf("%d₩n", n);
}
```

지역변수의 auto는 생략가능

m, sum은 for 문 내부의 블록 지역변수

n은 참조 가능

m, sum 참조 불가능

매개변수인 param도 지역 변수와 같이 사용

n은 참조 불가능

[결과]

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

전역 변수 예제

[globalvar.c]

```
#include <stdio.h>
double getArea(double);
double getCircum(double);
double PI = 3.14;
int gi;
int main(void) {
    double r = 5.87;
    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);
    printf("gi: %d₩n", gi);
    return 0;
}
double getArea(double r) {
    return r*r*PI;
}
```

전역변수 gi는 초기값을 주지 않아도 자동으로 default value인 0이 저장됨

전역변수 PI와 같은 이름의 지역변수 선언

지역변수 PI 참조

전역변수 gi 기본 값

전역변수 PI 참조

[결과]

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

다른 파일의 전역변수 PI 사용을 위해 extern 선언

[circumference.c]

```
extern double PI;
double getCircum(double r) {
    return 2 * r * PI;
}
```

피보나치 수열 (fibonacci.c)

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
int count;
void fibonacci(int prev_number, int number);
int main(void) {
    auto int prev_number = 0, number = 1;
    printf("피보나츠를 몇 개 구할까요?(3 이상) >> ");
    scanf("%d", &count);
    if (count <= 2)
        return 0;
    printf("1 ");
    fibonacci(prev_number, number);
    printf("₩n");
}
void fibonacci(int prev_number, int number) {
    static int i = 1;
    while (i++ < count) {
        int next_num = prev_number + number;
        prev_number = number;
        number = next_num;
        printf("%d ", next_num);
        fibonacci(prev_number, number);
```

전역변수를 표준입력으로 저장

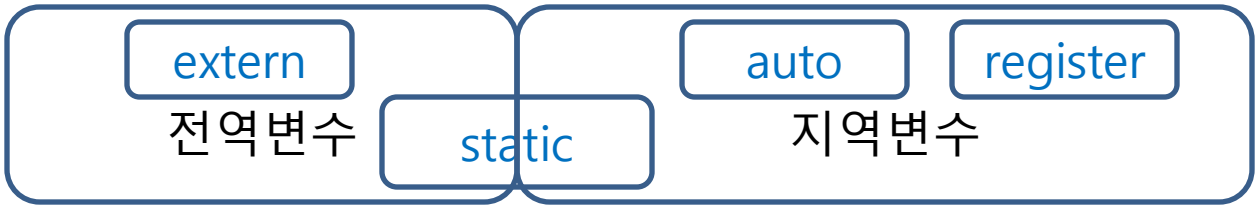
정적 지역 변수

[결과]

```
피보나츠를 몇 개 구할까요?(3 이상) >> 10
1 1 2 3 5 8 13 21 34 55
```

변수 4가지 기억부류(storage class)

- 기억부류 : 할당되는 메모리 영역, 생성/제거 시기, 유효 범위가 결정됨
- auto
  - 일반 지역변수의 특징. 생략 가능.
- register
  - 지역변수에만 이용 가능. 일반 메모리 영역이 아닌 cpu 내부 메모리일 레지스터에 잡힘 -> 속도 빠름.
  - 주소가 없으므로 & 연산자 사용 불가.
  - 보통 반복제어 변수 등에 사용.
- static
  - 지역과 전역 모든 변수에 이용 가능. 생성/제거 시기가 프로그램 생성/제거와 일치.
  - 초기값은 상수로만 줄 수 있음. 초기화는 1회만 됨.
- extern
  - 전역변수에만 사용 가능
  - 실제 선언이 아니라 컴파일러에게 변수가 이미 어딘가에 있음을 알리는 역할
  - 실제 선언이 아니므로 선언하면서 초기화 불가.



레지스터 변수
(registervar.c)

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

int main(void) {
    register int sum = 0;
    int max;
    printf("양의 정수 입력 >> ");
    scanf("%d", &max);
    for (register int count = 1; count <= max; count++)
        sum += count;
    printf("합: %d₩n", sum);
    return 0;
}
```

레지스터 지역변수 선언

레지스터 블록 지역변수 선언

[결과]
양의 정수 입력 >> 7
합: 28

정적 지역 변수
(staticlocal.c)

```
#include <stdio.h>
void increment(void); //함수원형
int main(void) {
    for (int count = 0; count < 3; count++)
        increment();
}

void increment(void) {
    static int sindex = 1;
    auto int aindex = 1;

    printf("정적 지역변수 sindex: %2d,₩t", sindex++);
    printf("자동 지역변수 aindex: %2d₩n", aindex++);
}
```

3전 반복 호출

정적 지역변수

[결과]

```
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 전역 변수
(staticgvar.c)

```
#include <stdio.h>

static int svar;
int gvar;

void increment();
void testglobal();
//void teststatic();

int main(void) {
    for (int count = 1; count <= 5; count++)
        increment();
    printf("함수 increment()가 총 %d번 호출되었습니다.₩n", svar);
    testglobal();
    printf("전역 변수: %d₩n", gvar);
    //teststatic();
}
void increment() { svar++; }
```

정적 전역변수

var은 다른 파일에서 extern 선언하여 사용 가능.
svar은 다른 파일에서 extern 선언하여 사용 불가능.

[결과]
함수 increment()가 총 5번 호출되었습니다.
전역 변수: 10

메모리 영역

- 메모리 영역 : 변수의 유효범위(scope), 생존기간(life time)에 결정적 역할
- 변수의 기억부류(storage class) : 할당되는 메모리 공간이 달라짐

(1) 데이터 영역
- 전역변수와 정적변수가 할당되는 저장공간
- 메모리 주소가 낮은 값에서 높은 값으로 저장 장소가 할당됨
- 프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리 영역 확보

(2) 힙 영역
- 동적 할당(dynamic allocation)되는 변수가 저장되는 공간
- 데이터 영역과 스택 영역 사이에 위치

(3) 스택 영역
- 함수 호출에 의한 형식 매개변수, 지역변수가 할당되는 저장공간
- 메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당 (함수 호출과 종료에 따라 메모리가 할당되었다가 다시 제거되는 작업 반복)

- 힙 영역과 스택영역은 프로그램이 실행되면서 영역 크기가 계속적으로 변함

전역변수와 지역변수 활용
(storageclass.c)

```
#include <stdio.h>
void infunction(void);
void outfunction(void);
int global = 10;
static int sglobal = 20;

int main(void) {
    auto int x = 100;
    printf("%d, %d, %d₩n", global, sglobal, x);
    infunction(); outfunction();
    infunction(); outfunction();
    infunction(); outfunction();
    printf("%d, %d, %d₩n", global, sglobal, x);
    return 0;
}
void infunction(void) {
    auto int fa = 1;
    static int fs;
    printf("₩t%d, %d, %d, %d₩n", ++global, ++sglobal, fa, ++fs);
}
```

global : 외부에서 사용 가능.
sglobal : 외부에서 사용 불가능.

fs : 0으로 초기화되고, 재호출시에도 값이 유지됨.

[out.c]

```
#include <stdio.h>
void outfunction() {
    extern int global;
    printf("₩t₩t%d₩n", ++global);
}
```

[결과]

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