초급 C언어

가천대학교
Gachon University


함 수(Function)

함수(function)의 개념

함수(function) 개념

- 프로그램도 모든 기능을 한번에 코딩하지 않고, 일부 기능을 '하청'처럼 나누어 코딩하거나 자동차의 부품처럼 독립시켜 코딩할 수 있다.
- 즉, 한 개의 프로그램을 일을 처리하는 가장 작은 단위인 '함수'로 나누어 코딩할 수 있다.

- 함수를 사용하면 자주 사용되는 코드 블록을 부품화하여 사용하므로 프로그램 내에서 동일한 코드 블록이 여러 번 나오는 것을 방지할 수 있다.
- 또한 일정한 기능의 코드 단위로 프로그램을 작성하므로 기능의 변경 및 교체, 확장시 해당 부분만 수정하면 되므로 프로그램 작성이 쉬워진다.

프로그램 속 함수의 기본적인 요소

- 일정한 기능을 처리할 함수와 함수의 이름(function name)
- 함수가 처리할 때 필요한 정보를 전달하기(인자, argument 또는 파라미터, parameter)
- 함수가 처리한 결과를 돌려 받기(반환 값 또는 리턴 값, return value)

main 함수의 정의 살펴보기

```
int main( ){

    …
   printf("C program₩n");
   …
   return 0;
}
```

- 함수 정의의 구성
  - 헤더 : 리턴값의 자료형과 함수명, ( )안에 필요한 만큼 매개변수로 구성
    예) 리턴값의 자료형: int, 함수명: main, 매개변수: ( )안에 매개변수 없음
  - 몸체 : { } 안에 함수가 처리할 문장을 나열
- 함수 정의 내에서 다른 함수를 호출하여 사용할 수 있음
  - main 함수의 정의 내에서 printf 함수 호출
    - ✓ printf 함수의 정의가 있는 곳으로 " C program₩n"라는 인자를 가지고 가서 작업
    - ✓ printf 함수 정의의 작업 종료 후, main 함수로 되돌아 옴
- 함수의 리턴값 반환
  - main함수의 작업을 종료한 후 자신을 호출했던 운영체제로 되돌아 감
  - 되돌아 갈 때 처리 결과의 반환값으로 0을 돌려 줌
  - 반환 값 0은 정수이므로 리턴 값의 자료형은 int임

함수의 유형

함수의 유형 4가지 살펴보기

#1- 리턴값이 없고 인자도 없는 경우

#2- 리턴값이 있고 인자가 없는 경우

#3- 리턴값이 없고 인자가 있는 경우

#4- 리턴값이 있고 인자도 있는 경우

※ 1~4와 같은 번호는 함수 유형을 구분하기 위해 편의상 붙인 것임
※ 인자(argument)는 함수 호출시 사용되고 매개변수(parameter)는 함수 정의시 사용되는 용어이지만, 인자로 혼용하여 사용하기도 함

함수의 유형#1-리턴값이 없고 인자도 없는 경우

- 자주 사용되는 코드 블록을 함수로 부품화

→ 동일한 코드 블록이 여러 번 나오는 것을 방지

```
int main(){
 ...
  int i;
  for(i = 0 ; i < 20 ; i++){
          printf("-");
  }
  printf("₩n");
  ...
for(i = 0 ; i < 20 ; i++){
          printf("-");
}
printf("₩n");
 ...
for(i = 0 ; i < 20 ; i++){
          printf("-");
}
printf("₩n");

 return 0;
}
```

```
int main(){
 ...
      PrintLine( );
...
      PrintLine( );
...
      PrintLine( );
 ...
      return 0;
}
```

- 함수 호출
  - 호출할 함수 이름을 ( )와 함께 작성
  - 함수 정의에 전달할 정보 : 없음
    - ✓ 인자가 없으면 ( )안을 비워둠

```
void PrintLine( ) {
      int i;
      for(i = 0 ; i < 20 ; i++){
          printf("-");
      }
      printf("₩n");
}
```

- 함수 정의
  - 함수 이름 : PrintLine
  - 호출한 곳에서 전달 받을 정보 : 없음
    - ✓ 매개 변수가 없으면 ( )을 비워 둠
  - 반환 값 : 없음
    - ✓ 반환 값이 없으면 함수명 앞에 void 라고 작성
  - 함수가 호출되면 함수 정의 내부의 문장을 실행
  - 함수 정의의 실행이 끝나면, 호출 위치로 돌아감

다음 프로그램의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
void PrintLine( ) {

      int i;
      for(i = 0 ; i < 20 ; i++){
          printf("-");
      }
      printf("₩n");

}

int main(){
       PrintLine( );
       PrintLine( );
       PrintLine( );

       return 0;
}
```

[주의]

- 함수 정의의 헤더는 함수 호출 전에 나와야 한다.

[결과]

```
--------------------
--------------------
--------------------
```

함수의 유형#2- 리턴값은 있고 인자는 없는 경우

```
int main() {
  char result;

  result = input();
  printf("%c₩n", result);

  return 0;
}
```

- 함수 호출
  - 호출할 함수 이름을 ( )와 함께 작성
  - 함수 정의에 전달할 정보 : 없음
    - ✓ 인자가 없으면 ( )안을 비워둠
  - 호출한 함수의 리턴값 : 있음
    - ✓ 리턴 값은 대입 연산자를 사용하여 해당 자료형의 변수에 저장

    예) result = input();
    - ✓ 리턴 값은 상황에 따라 사용하지 않을 수 있음

```
char input( ){
    char ch;
   ...
    scanf_s("%c", &ch, sizeof(ch));

    return ch;
}
```

- 함수 정의
  - 함수 이름 : input
  - 호출한 곳에서 전달 받을 정보 : 없음
    - ✓ 매개 변수가 없으면 ( )을 비워 둠
  - 반환 값 : 있음
    - ✓ 결과 값의 자료형을 함수명 앞에 작성

    예) char
    - ✓ 결과값은 return과 함께 작성하여 호출한 곳으로 돌려줌

      → 결과값은 ( )안에 넣어도 됨

      예) return ch;는 return (ch);와 동일한 의미
  - 반환 값은 하나만 가능

다음 프로그램의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
char input() {
        char ch;

        printf("1개의 문자를 입력하세요: ");
        scanf_s("%c", &ch, sizeof(ch));
        return ch;
}
int main() {
        char result;

        result = input();
        printf("입력한 문자 : %c₩n", result);
        return 0;
}
```

[결과]
1개의 문자를 입력하세요 : ***K***
입력한 문자 : K

함수의 유형#3- 리턴값은 없고 인자는 있는 경우

```
int main(){
    int a, b;

    printf(" 2개의 정수를 입력하세요 : ");
    scanf_s("%d %d", &a, &b);

    add( a, b );// 인자

     return 0;
}
```

```
void add(int x, int y){   // 매개 변수

  printf("%d + %d = %d₩n", x, y, x + y);

}
```

- 함수 호출
  - 호출할 함수 이름을 ( )와 함께 작성
  - 함수 정의에 전달할 정보 : 있음
    - ✓ 인자가 있으면 ( )안에 필요한 만큼 상수나 변수를 넣음
  - 호출한 함수의 리턴값 : 없음

- 함수 정의
  - 함수 이름 : add
  - 호출한 곳에서 전달 받을 정보 : 있음
    - ✓ 매개 변수가 있으면 ( )에 필요한 만큼 변수를 선언
  - 반환 값 : 없음
    - ✓ 반환 값이 없으면 함수명 앞에 void 라고 작성

[정리]
- 인자(argument) : 함수를 호출할 때 직접 넘겨주는 값으로 상수 또는 변수
- 매개 변수(parameter) : 인자를 저장하기 위해 함수 정의의 헤더에 있는 변수
- 인자의 값은 순서대로 매개변수에 저장됨
- 함수의 인자 및 매개변수의 개수에는 제한 없음

다음 프로그램의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
void add(int x, int y) {
        printf("%d + %d = %d₩n", x, y, x + y);
}

int main() {
        int a, b;

        printf("2개의 정수를 입력하세요 : ");
        scanf_s("%d %d", &a, &b);
        add(a, b);

        return 0;
}
```

```
[결과]
2개의 정수를 입력하세요 :  2 3
2 + 3 = 5
```

다음 프로그램의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
int add(int x, int y) {
  int result;

  result = x + y;
  return (result);
}

int main() {
  int sum;

  sum = add(2, 3);
  printf("2와 3의 합 : %d₩n", sum);
  return 0;
}
```

[결과]
2와 3의 합 : 5

※ 전달받은 result의 값을 sum에 저장하여 사용

다음 프로그램의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
int add(int x, int y) {
  int result;

  result = x + y;
  return (result);
}

int main() {

  printf("2와 3의 합 : %d₩n", add(2, 3));
  return 0;
}
```

[결과]
2와 3의 합 : 5

※ 전달받은 result의 값을 다른 변수에 받아 사용하지 않고 직접 사용할 수 있음

함수의 선언

함수 선언

- 이 문제를 해결하기 위해서 뒤에 나오는 함수의 원형을 미리 알려주는 함수 선언을 사용한다.
  ※ 함수의 원형 : - 매개변수의 자료형과 개수 정보
  - 함수 정의의 헤더 부분에 포함되어 있음
- 함수 선언의 형태와 위치
  - 함수 헤더에 ;을 추가하여 프로그램 상단 main 함수 정의 전에 써 준다.

  예) int add(int x, int y) ;

```
#include <stdio.h>

int add(int x, int y);

int main( ){
    int sum;

    sum = add(2, 3);
    ...
    return 0;
}

int add(int x, int y){
    ...
}
```

[실습] 다음 조건을 만족하는 프로그램을 작성하시오.

[결과]

근로 시간은? (종료시 -1) : ***12***

오늘의 임금은 140000원

근로 시간은? (종료시 -1) : ***7***

오늘의 임금은 70000원

근로 시간은? (종료시 -1) : ***-1***

프로그램을 종료합니다.

[조건]

- 근로자의 오늘 하루 근로 시간을 입력받아 하루 임금을 계산하여 출력
- -1을 입력시 반복 종료 및 프로그램 종료

- 임금 계산 조건 : 시급제와 일급제를 혼용
  - 시급은 시간당 10,000원
  - 8시간 이상 근로시
    - ✓ 8시간에 대한 일급으로 100,000원을 적용 후
    - ✓ 남은 시간은 시급 적용

- 임금 계산은 다음과 같은 형태의 함수 호출과 해당 함수 선언을 포함함
  - ✓ 호출할 함수명 : get_pay
  - ✓ 인자 : 입력받은 근로시간(hours), 일급 십만원(day_rate), 시급 만원(hour_rate)
  - ✓ 반환값 : 임금 조건에 따라 계산된 오늘의 임금

[실습] 다음과 같은 결과가 나오도록 프로그램을 작성하시오. (1)

[결과]
*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
****
메뉴를 선택하세요 : ***0***

*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
****
메뉴를 선택하세요 : ***4***

*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
****
메뉴를 선택하세요 : ***1***
홀짝을 구분할 정수를 입력해주세요 : ***67***
67은(는) 홀수입니다.

*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
****
메뉴를 선택하세요 : ***1***
홀짝을 구분할 정수를 입력해주세요 : ***68***
68은(는) 짝수입니다.

[실습] 다음과 같은 결과가 나오도록 프로그램을 작성하시오. (2)

```
*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
********************
메뉴를 선택하세요 : 2
수식을 입력하시오(예: 2 + 5)
>>10 + 3
10 + 3 = 13


*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
********************
메뉴를 선택하세요 : 2
수식을 입력하시오(예: 2 + 5)
>>10 - 3
10 - 3 = 7
```

```
*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
********************
메뉴를 선택하세요 : 2
수식을 입력하시오(예: 2 + 5)
>>10 * 3
10 * 3 = 30


*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
********************
메뉴를 선택하세요 : 2
수식을 입력하시오(예: 2 + 5)
>>10 / 3
10 / 3 = 3
```

```
*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
********************
메뉴를 선택하세요 : 2
수식을 입력하시오(예: 2 + 5)
>>10 $ 3
지원되지 않는 연산자입니다.

*******메뉴*******
1. 홀수 짝수 구분
2. 사칙 연산
3. 종료
********************
메뉴를 선택하세요 : 3
메뉴 프로그램을 종료합니다.
계속하려면 아무 키나 누르십시오 . . .
```

[실습] 다음과 같은 결과가 나오도록 프로그램을 작성하시오. (3)

[조건]

- 3번 종료 메뉴를 선택할 때까지 다음을 반복
  - 1번 메뉴를 선택하면 입력 받은 정수의 홀짝 여부를 출력
  - 2번 메뉴를 선택하면 2개의 정수와 사칙연산 기호를 입력 받아 그 계산 결과를 출력하되 사칙 연산 기호가 아닌 경우, 지원되지 않는 연산자임을 출력
  - 1~3번 외의 숫자를 입력하면, 다시 메뉴를 보여줌

- do ~ while문과 switch문을 사용

- 1~2번 메뉴는 함수 정의와 함수 선언을 사용
  - 1번 메뉴는 리턴값이 없고 인자는 있는 함수 유형을 사용, ex. void odd_even(int x)
  - 2번 메뉴는 리턴값이 없고 인자도 없는 함수 유형을 사용, ex. void calculate()

함수의 인자 전달 방법

다음 코드의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
void PrintCount();

int main(){
        int count = 0;

        printf("main: count = %d₩n", count);

        PrintCount();
        return 0;
}

void PrintCount(){
        int count = 100;

        printf("PrintCount: count = %d₩n", count);
}
```

[결과]
main: count = 0
PrintCount: count = 100

지역 변수(1)

- 서로 다른 함수에서 동일한 이름의 변수를 선언하면 서로 다른 변수가 되며, 그 함수 내에서만 사용 가능하다.
- 이러한 변수를 '지역 변수'라고 한다.

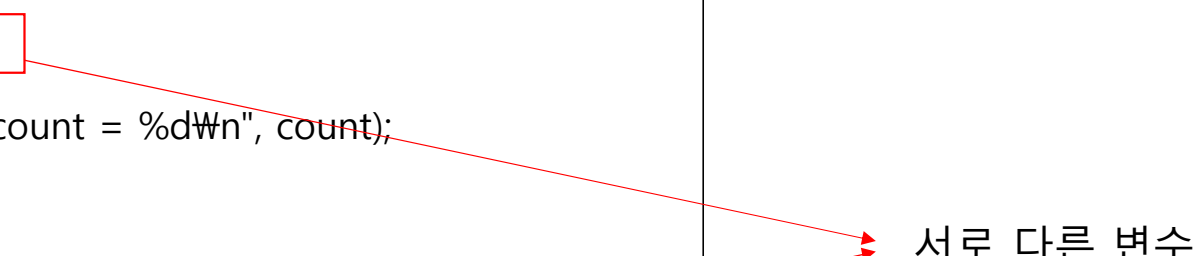
```
int main(){
        int count = 0;

        printf("main: count = %d₩n", count);

        PrintCount();
        return 0;
}

void PrintCount(){
        int count = 100;

        printf("PrintCount: count = %d₩n", count);
}
```



지역 변수(2)

- 지역 변수는 함수 정의 안에서 만들어지고,
- 그 함수 안에서만 사용되는 변수이며,
- 함수가 리턴할 때 항상 자동으로 없어진다.

```
int main(){
        int count = 0;  // 지역변수의 생성

        printf("main: count = %d₩n", count);

        return 0;       // 지역변수의 제거
}
```

- 함수의 매개변수도 지역 변수이다.

```
int add(int x, int y){ // 지역변수 x, y의 생성

   int result;          // 지역변수 result의 생성

   result = x + y;
   return result;      // 지역변수 x, y, result의 제거
 }
```

다음 코드의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
void Swap(int x, int y);

int main(){
   int a = 10;
   int b = 20;

   printf("Swap 전의 a = %d, b = %d₩n", a, b);
   Swap(a, b);
   printf("Swap 후의 a = %d, b = %d₩n", a, b);

   return 0;
}

void Swap(int x, int y){
   int temp;

   temp = x;
   x = y;
   y = temp;
}
```

[결과]
Swap 전의 a = 10, b = 20
Swap 후의 a = 10, b = 20

[참고] 두 변수의 값을 서로 교환할 때

- x = y;
  y = x;

→ x에 기존 값 대신 y값이 저장됨
→ y에 원래 x 값 대신 y값이 저장됨
→ x, y 두 변수의 값이 서로 교환되지 않음

- temp = x;
  x = y;
  y = temp;

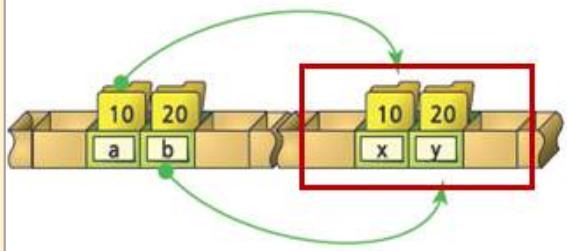
→ temp에 x의 기존 값을 저장하여 보존
→ x에 y값을 저장
→ y에 temp에 저장되어 있던 x의 기존 값을 저장
→ x, y 두 변수의 값이 서로 교환됨

Swap 함수의 수행 과정

1) 함수를 호출하면서 인자의 값이 전달됨

```
int main(){
    ...
    Swap(a, b);
    ...
}


void Swap(int x, int y){
    ...
}
```



2) Swap함수를 실행하면서 x와 y값이 맞교환됨

```
void Swap(int x, int y){
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

3) - Swap함수를 리턴함
- 리턴시 지역변수 x, y, temp는 해제

- main함수의 a, b값은 변경 없음

```
int main(){
    ...
    Swap(a, b);
    ...
}
```

함수의 인자 전달 방법 - 값에 의한 호출

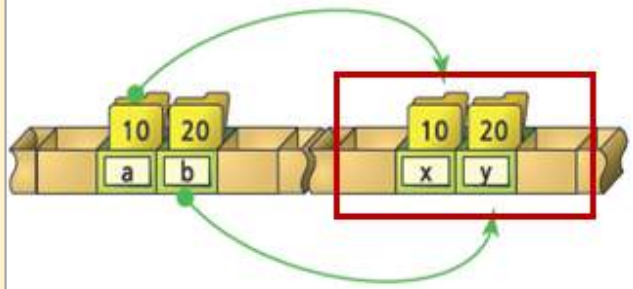
- 함수를 호출하면서 인자의 값이 전달됨

→ 호출된 함수에서는 매개변수 기억장소가 확보되고, 전달된 값이 매개변수에 저장됨

→ 매개 변수는 지역 변수이므로 인자와 매개변수는 서로 다른 기억 장소를 사용

```
int main(){
    ...
    Swap(a, b);
    ...
}


void Swap(int x, int y){
    ...
}
```



→ 호출된 함수에서 매개변수 값을 변경하더라도, 인자는 변경되지 않음

→ 함수 간 독립성 보장

함수 호출시 인자 전달 방법의 비교

- 값에 의한 호출 방식
  - 호출된 함수에 변수의 값을 전달함
  - 함수 간 독립성을 보장하므로 많이 사용됨
  - 문제점 : 호출된 함수에서 자신을 호출한 함수의 인자를 수정할 수 없다.

- 주소에 의한 호출 방식
  - 호출된 함수에 변수의 값 대신 변수의 주소를 전달함
  - 주소에 의한 호출 방식의 필요성
    - ✓ 호출된 함수에서 자신을 호출한 함수의 인자를 변경하고 싶다면 변수의 주소(포인터)를 이용하여야 함

호출 함수에게 변경된 결과 값을 돌려주는 방법

- 함수 호출(function call)시 값에 의한 전달(call by value)
  - 함수 간 독립성 보장
  - 문제점: 호출된 함수에서는 자신을 호출한 함수에서 보낸 변수를 수정할 수 없다.

- 자신을 호출한 함수에게 변경된 결과 값을 돌려 주는 방법
  - 변경된 결과값이 1개일 경우 : return 문 사용
  - 변경된 결과값이 2개 이상일 경우
    - 전역 변수(global variable) 활용
    - 포인터(pointer) 활용

변경된 결과값이 1개일 경우 – return 문 사용

```
#include <stdio.h>
int get_larger(int x, int y);
int main(){
        int result;

        result = get_larger(5, 10);
        printf("max:%d₩n", result);

        return 0;
}
```

```
int get_larger(int x, int y){
    int max;

    if (x > y){
        max=x;
    }
    else{
        max=y;
    }

    return max;
}
```

변경된 결과값이 2개 이상일 경우 – 전역 변수 활용

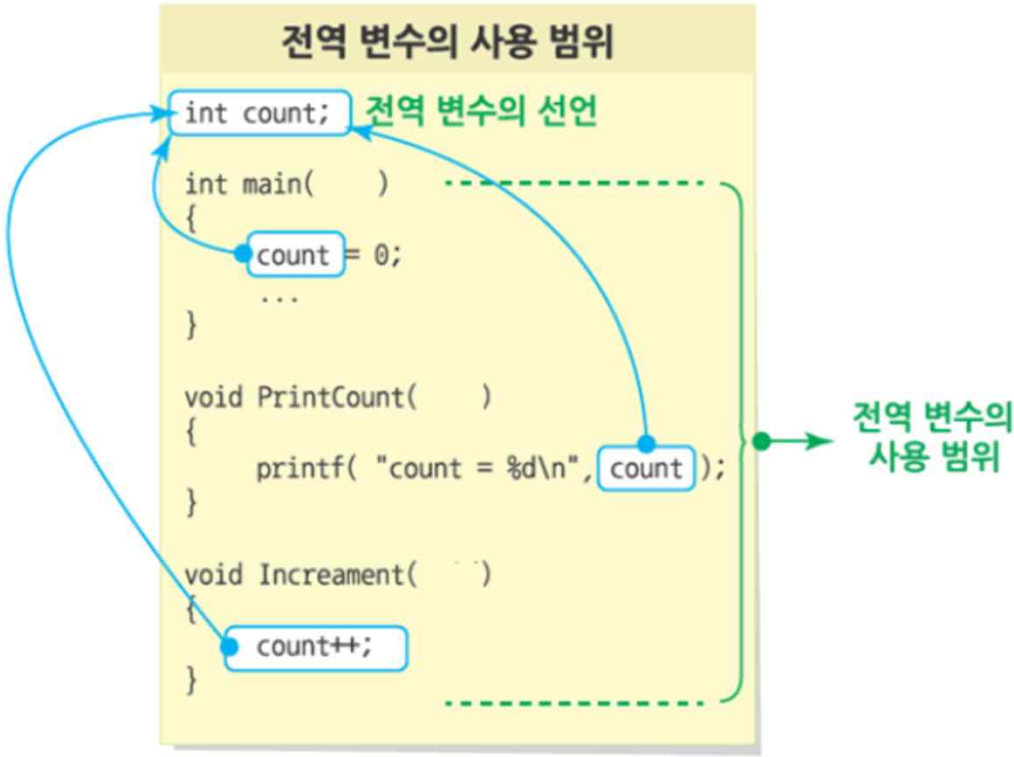
- 전역 변수는 프로그램이 수행되는 동안 여러 함수에서 사용할 수 있는 변수이다.
- 자신을 호출한 함수에게 2개 이상의 변경된 결과 값을 돌려 주기 위해 전역 변수를 활용할 수 있다.

전역 변수(1)

- 전역변수(global variable)
  - 함수 정의 외부에 선언된 변수
  - 프로그램내의 모든 함수에서 사용할 수 있는 변수

- 프로그램이 시작될 때 한 번만 생성되고,
- 프로그램이 수행되는 동안 여러 함수에서 사용되다가,
- 프로그램이 종료될 때 비로소 해제된다.

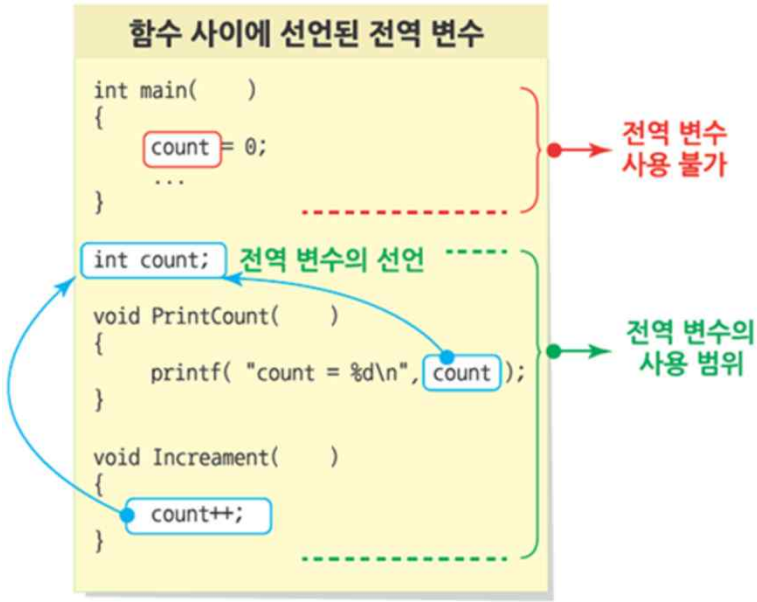
※ 지역변수(local variable)
  - 함수 정의 내부에 선언된 변수(매개 변수도 포함)
  - 해당 함수 정의 내에서만 사용할 수 있는 변수



전역 변수(2)

- 전역 변수의 성질
  - 전역 변수의 선언 다음에 정의된 함수에서만 전역 변수를 사용할 수 있다.
    - 따라서, 전역 변수의 선언은 파일의 시작 부분에 넣어주는 것이 좋다.

  - 전역 변수는 따로 초기화하지 않으면 자동으로 0으로 초기화된다.
  - 전역 변수와 지역 변수 이름이 동일하면, 지역 변수가 우선시 된다.



[실습] 다음 코드의 결과를 예측하여 보고 실행하여 비교하시오.

```
#include <stdio.h>
int value = 10; // 전역 변수

int main() {
    int value = 25; // 지역 변수

     printf("value = %d₩n", value);
     return 0;
}
```

[결과]
value = 25

※ 같은 이름의 전역 변수와 지역 변수가 있을 때 함수내에서는 지역 변수 값이 출력된다.

[실습] 다음 결과가 나오도록 주어진 프로그램을 변경하시오.

[결과]
Swap 전의 a = 10, b = 20
Swap 후의 a = 20, b = 10

[조건]
- main 함수의 지역 변수 a, b를 전역 변수로 바꾸어서
- swap 함수 실행 후에 a, b의 맞 교환 된 값이 main 함수에서 출력되도록
- 오른쪽에 주어진 코드를 수정

```
#include <stdio.h>
void Swap(int x, int y);

int main(){
   int a = 10;
   int b = 20;

   printf("Swap 전의 a = %d, b = %d₩n", a, b);
   Swap(a, b);
   printf("Swap 후의 a = %d, b = %d₩n", a, b);

   return 0;
}

void Swap(int x, int y){
   int temp;

   temp = x;
   x = y;
   y = temp;
}
```

호출 함수에게 변경된 결과 값을 돌려주는 방법

- 함수 호출(function call)시 값에 의한 전달(call by value)
  - 함수 간 독립성 보장
  - 문제점: 호출된 함수에서는 자신을 호출한 함수에서 보낸 변수를 수정할 수 없다.

- 자신을 호출한 함수에게 변경된 결과 값을 돌려 주는 방법
  - 변경된 결과값이 1개일 경우 : return 문 사용
  - 변경된 결과값이 2개 이상일 경우
    - 전역 변수(global variable) 활용

      → 전역 변수를 사용하면 호출 함수에게 변경된 결과 값을 여러 개 돌려 줄 수 있지만 모든 함수에서 접근하여 사용하므로 오류 수정의 어려움이 있어서 실무에서는 제한적으로 쓰임.

    - 포인터(pointer) 활용

      → 실무에서 자주 사용되는 포인터에 의한 함수 호출 방식은 '포인터' 부분에서 다룸

T h a n k y o u


TECHNOLOGY
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Velit ex
Incabo ipsum. labore sed tempora ratione asperiores des
labore sed tempora rati [gert one bore sed temi