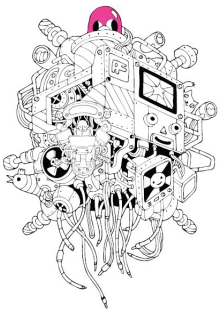
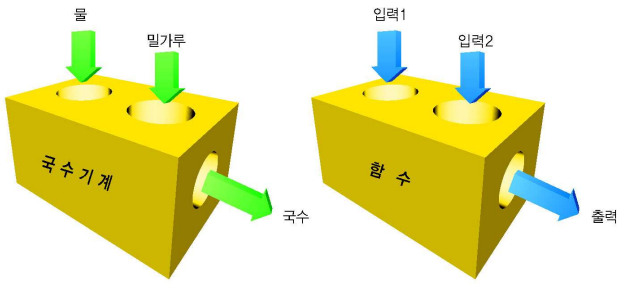


Chapter 09. C언어의 핵심! 함수!

9-1. 함수를 정의하고 선언하기

함수의 의미

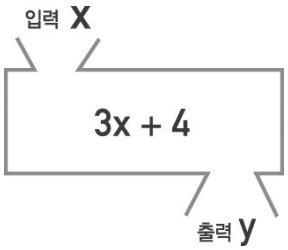
- 함수
  - 함수는 원하는 특정한 작업을 수행하는 독립된 프로그램 단위
- black box



3

함수의 의미

- 함수에 대한 이해
  - 적절한 입력과 그에 따른 출력이 존재 하는 것을 가리켜 함수라 한다.
  - C 언어의 기본 단위는 함수이다.



f(x) = 3x + 4
f(2) = 10

4

함수의 정의

- 함수에 대한 이해

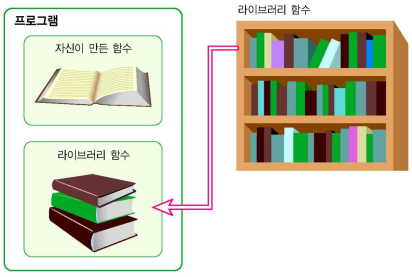
함수의 호출

```
scanf ("%d", &a);
printf ("%d₩n", a);
```

5

함수의 구분

- 함수의 구분
  - 함수는 크게 시스템 정의 함수와 사용자 정의 함수로 구분
- 시스템 정의 함수
  - 라이브러리 함수는 이미 개발도구에 구현되어 있는 프로그램 부품 (component) 함수
  - 예: printf(), scanf() 등
- 사용자 함수
  - 사용자 프로그래머가 필요에 의하여 직접 정의하여 이용하는 함수
  - 개발자가 직접 개발하는 함수



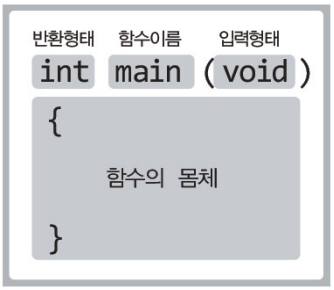
6

함수를 만드는 이유

"Divide and Conquer!"

- 다수의 작은 단위 함수를 만들어서 프로그램을 작성하면 큰 문제를 작게 쪼개서 해결하는 효과를 얻을 수 있다.
- 그러나 함수를 만드는 이유 및 이점은 이보다 훨씬 다양하다.

- main 함수를 포함하여 함수의 크기는 작을수록 좋다.
- 무조건 작다고 좋은 것은 아니지만, 불필요하게 큰 함수가 만들어지지 않도록 주의해야 한다.
- 하나의 함수는 하나의 일만 담당하도록 디자인 되어야 한다.
- 물론 하나의 일이라는 것은 매우 주관적인 기준이다. 그러나 이러한 주관적 기준 역시 프로그래밍에 대한 경험이 쌓이면 매우 명확한 기준이 된다.



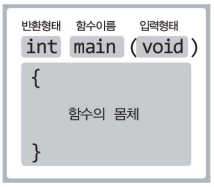
7

main 함수 이외의 가장 간단한 나만의 함수

- "나는 인제대학교 1학년 홍길동입니다."를 출력하는 프로그램
  - 단, 출력은 main 함수에서 하지 않고 print_out이라는 함수에서 수행

```
int main(void) {
    printf("나는 인제대학교 1학년 홍길동입니다.₩n");
    return 0;
}
```

```
int main (void) {
    print_out();        ← 함수호출
    return 0;
}
                        ← 함수정의
void print_out (void) {
    printf("나는 인제대학교 1학년 홍길동입니다.₩n");
}
```

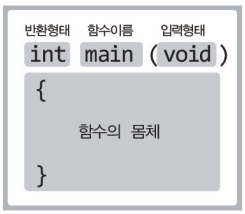


8

연습문제1

- 기본적인 함수 정의와 함수 호출 문제
  - main 함수 이외에 추가로 print_out 함수를 만들 것
  - main 함수에서는 print_out 함수를 호출
  - print_out 함수에서는 "Hello! C functions!"를 3회 출력

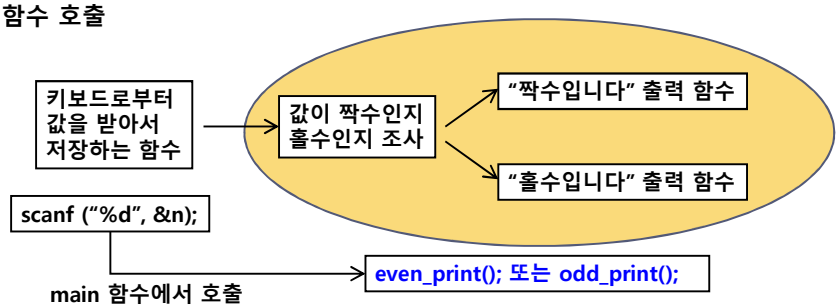
```
Hello! C functions!
Hello! C functions!
Hello! C functions!
```



9

연습문제2

- 기본적인 함수 정의와 함수 호출 문제
  - 함수 호출

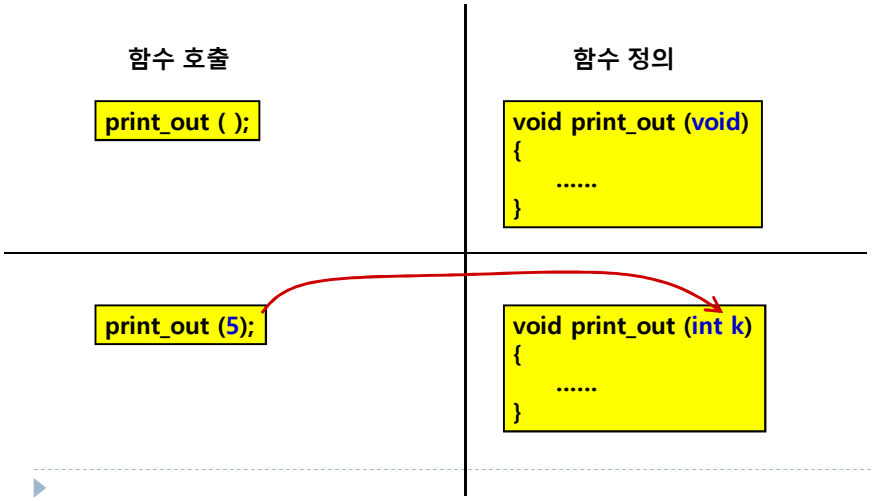


  - 함수 정의
    - scanf, printf 등의 함수는 시스템 정의 함수로써 이미 프로그램 되어 있음
    - even_print, odd_print라는 함수는 시스템에 없는 함수이므로 만들어야 함

10

함수 호출과 정의와의 관계

- 함수 호출과 함수의 정의와는 이름과 형태가 서로 맞아야 함

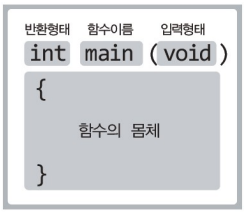


11

연습문제3

- "나는 인제대학교 1학년 홍길동입니다."를 n번 출력하는 프로그램
  - 키보드로부터 몇 번 출력할 것인지를 받아 n에 저장
  - 단, 출력은 main 함수에서 하지 않고 print_out이라는 함수에서 수행
  - print_out 함수를 n번 호출

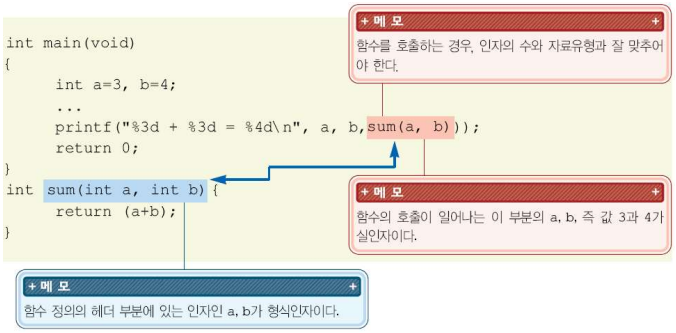
```
몇 번 출력할까요? 3
나는 인제대학교 1학년 홍길동입니다.
나는 인제대학교 1학년 홍길동입니다.
나는 인제대학교 1학년 홍길동입니다.
몇 번 출력할까요? 1
나는 인제대학교 1학년 홍길동입니다.
....
```



12

형식인자와 실인자

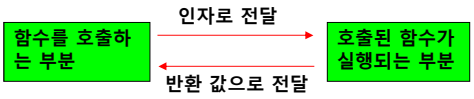
- 형식인자 (formal parameters)
  - 함수 정의 구문에서 기술되는 인자를 형식인자
- 실인자 (real parameters)
  - 함수를 호출할 때 기술되는 인자를 실인자
  - 실인자를 기술할 때는 함수의 헤더에 정의된 자료유형과 일치하도록 해야 함



13

반환 값

- 반환 값 역할
  - 함수 인자가 함수를 호출하는 부분에서 함수를 호출하는 영역으로 자료를 전달하는데 주로 사용
  - 함수 인자는 여러 개를 사용할 수 있으나 반환 값은 하나만 이용
- return 문
  - return 문장은 반환 값을 전달하는 목적과 함께 함수의 작업 종료도 의미
  - 반환 값을 전달 할 때 return 문장을 이용하며, 반환 값이 없는 함수는 return 문장 뒤에 반환 값을 기술하지 않음




```
return ;
return 연산식;          // return a+b;
return (연산식);        // return (a+b);
```

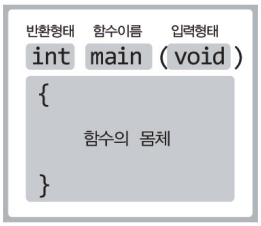
```
int sum (int a, int b)
{
    return (a+b);
}
```

14

두 문제를 한 프로그램으로 작성: 연습문제(Problem1)

- 정수 값 두 개의 합을 구하는 프로그램
  - 키보드로부터 정수 값 2개를 받아 n1, n2에 각각 저장
  - 이 값들의 합을 sum이라는 함수에서 수행하고 그 결과 값을 받아 main 함수에서 출력

```
정수 값 2개를 입력하세요 : 5 3
5와 3의 합은 8입니다.
```



15

두 문제를 한 프로그램으로 작성: 연습문제(Problem2)

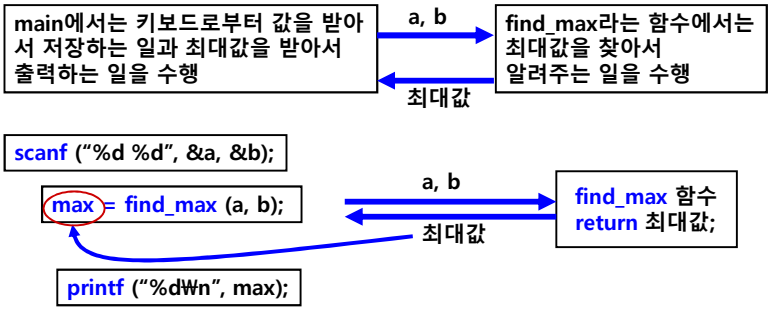
- 키보드로부터 정수 값 2개를 입력 받은 후 이들 중에서 가장 큰 값을 찾아서 출력하는 프로그램을 작성하시오.
  - 정수 값 2개의 입력은 main 함수에서 수행 (아래의 첫 줄)
  - 둘 중 큰 값을 찾는 것은 find_max라는 함수를 만들어서 그곳에서 수행
  - 찾은 큰 값을 출력하는 것은 main 함수에서 수행 (아래의 두 번째 줄)
  - 아래와 같은 출력 형태를 참고

```
정수 값 2개를 입력하세요 : 5 3
5와 3중에서 큰 값은 5입니다.
```

16

실습문제 풀이

- 함수 정의와 함수 호출
  - 함수 호출

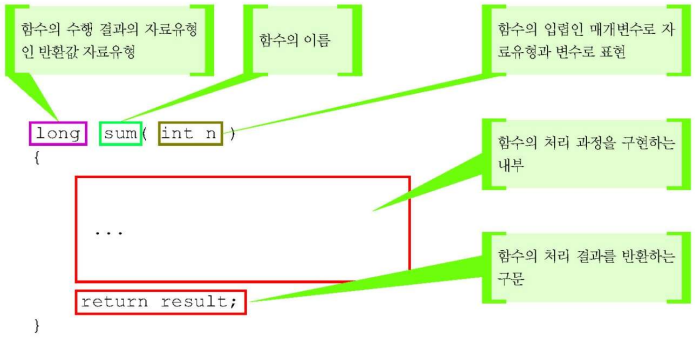


  - 함수 정의
    - scanf, printf 등의 함수는 시스템 정의 함수로써 이미 프로그램 되어 있음
    - find_max라는 함수는 개발자가 임의로 만든 함수로써 프로그램을 만들어야 함

17

함수 정의 정리

- 함수의 머리(헤더)는 함수의 반환값 자료유형과 함수의 이름, 그리고 괄호 사이에 인자 목록으로 구성



18

함수의 입력과 출력: printf 함수도 반환을 합니다.

```
int main(void)
{
    int num1, num2;
    num1=printf("12345\n");
    num2=printf("I love my home\n");
    printf("%d %d \n", num1, num2);
    return 0;
}
```

- printf 함수도 사실상 값을 반환한다. 다만 반환값이 필요 없어서 반환되는 값을 저장하지 않았을 뿐이다.
- printf 함수는 출력된 문자열의 길이를 반환한다.

```
12345
I love my home
6 15
```

실행결과

PrintfOutput.c

- 함수가 값을 반환하면 반환된 값이 함수의 호출문을 대체한다고 생각하면 된다.
- 예를 들어서 아래의 printf 함수 호출문이 6을 반환한다면,
- num1=printf("12345 \ n");
- 함수의 호출결과는 다음과 같이 되어 대입연산이 진행된다.
- num1=6;

19

함수의 구분

유형 1: 전달인자 있고, 반환 값 있다! 전달인자(○), 반환 값(○)

유형 2: 전달인자 있고, 반환 값 없다! 전달인자(○), 반환 값(×)

유형 3: 전달인자 없고, 반환 값 있다! 전달인자(×), 반환 값(○)

유형 4: 전달인자 없고, 반환 값 없다! 전달인자(×), 반환 값(×)

전달인자와 반환 값의 유무에 따른 함수의 구분!

- 전달인자는 호출하는 함수가 호출되는 함수에게 전달하는 것
- 반환 값은 호출된 함수가 호출한 함수로 넘겨주는 값

20

전달인자 반환 값 모두 있는 경우

전달인자는 int형 정수 둘이며, 이 둘을 이용한 덧셈을 진행한다.
덧셈결과는 반환이 되며, 따라서 반환형도 int형으로 선언한다.
마지막으로 함수의 이름은 Add라 하자!

덧셈이 우선 진행되고 그 결과가 반환됨

```
A. B. C.
int Add (int num1, int num2)
{
    int result = num1 + num2;
D. return result;
}
```

A. 반환형
B. 함수의 이름
C. 매개변수
D. 값의 반환

```
int Add(int num1, int num2)
{
    return num1+num2;
}

int main(void)
{
    int result;
    result = Add(3, 4);
    printf("덧셈결과1: %d \n", result);
    result = Add(5, 8);
    printf("덧셈결과2: %d \n", result);
    return 0;
}
```

함수호출이 완료되면 호출한 위치로 이동해서 실행을 이어간다.

SimpleAddFunc.c

실행결과

```
덧셈결과1: 7
덧셈결과2: 13
```

21

전달인자나 반환 값이 존재하지 않는 경우

```
void ShowAddResult(int num)   // 인자전달 (O), 반환 값 (X)
{
    printf("덧셈결과 출력: %d \n", num);
}
```

```
int ReadNum(void)   // 인자전달 (X), 반환 값 (O)
{
    int num;
    scanf("%d", &num);
    return num;
}
```

```
void HowToUseThisProg(void)   // 인자전달 (X), 반환 값 (X)
{
    printf("두 개의 정수를 입력하시면 덧셈결과가 출력됩니다. \n");
    printf("자! 그럼 두 개의 정수를 입력하세요. \n");
}
```

22

4가지 함수 유형을 조합한 예제

```
int Add(int num1, int num2)    // 인자전달 (O), 반환 값 (O)
{
    return num1+num2;
}

void ShowAddResult(int num)    // 인자전달 (O), 반환 값 (X)
{
    printf("덧셈결과 출력: %d \n", num);
}

int ReadNum(void)    // 인자전달 (X), 반환 값 (O)
{
    int num;
    scanf("%d", &num);
    return num;
}

void HowToUseThisProg(void)    // 인자전달 (X), 반환 값 (X)
{
    printf("두 개의 정수를 입력하시면 덧셈결과가 출력됩니다. \n");
    printf("자! 그럼 두 개의 정수를 입력하세요. \n");
}
```

```
int main(void)
{
    int result, num1, num2;
    HowToUseThisProg();
    num1=ReadNum();
    num2=ReadNum();
    result = Add(num1, num2);
    ShowAddResult(result);
    return 0;
}
```

실행결과

```
두 개의 정수를 입력하시면 덧셈결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
12 24
덧셈결과 출력: 36
```

SmartAddFunc.c

23

값을 반환하지 않는 return

```
void NoReturnType(int num)
{
    if(num<0)
        return;    // 값을 반환하지 않는 return문!
    . . . .
}
```

- **return문에는 '값의 반환'과 '함수의 탈출'이라는 두 가지 기능이 담겨있다.**
- **위의 예제에서 보이듯이 값을 반환하지 않는 형태로 return문을 구성하여 값은 반환하지 않되 함수를 빠져나가는 용도로 사용할 수 있다.**

24

함수의 정의와 그에 따른 원형의 선언

```
int Increment(int n)
{
   n++;
   return n;
}

int main(void)
{
   int num=2;
   num=Increment(num);
   return 0;
}
```

앞서 본 함수

```
int main(void)
{
   int num=2;
   num=Increment(num);
   return 0;
}

int Increment(int n)
{
   n++;
   return n;
}
```

본적 없는 함수

컴파일 진행 방향

컴파일이 위에서 아래로 진행이 되기 때문에 **함수의 배치순서는 중요**하다. 컴파일 되지 않은 함수는 호출이 불가능하다.

```
int Increment(int n);

int main(void)
{
   int num=2;
   num=Increment(num);
   return 0;
}

int Increment(int n)
{
   n++;
   return n;
}
```

함수의 선언

함수의 정의

이후에 등장하는 함수에 대한 정보를 컴파일러에게 제공해서 이후에 등장하는 함수의 호출문장이 컴파일 가능하게 도울 수 있다.
이렇게 제공되는 함수의 정보를 가리켜 '**함수의 선언**'이라 한다.

```
int Increment(int n);   // 함수의 선언
int Increment(int);     // 위와 동일한 함수선언, 매개변수 이름 생략 가능
```

25

다양한 종류의 함수 정의 1

```
int main(void)
{
    printf("3과 4중에서 큰 수는 %d 이다. \n", NumberCompare(3, 4));
    printf("7과 2중에서 큰 수는 %d 이다. \n", NumberCompare(7, 2));
    return 0;
}

int NumberCompare(int num1, int num2)
{
    if(num1>num2)
        return num1;
    else
        return num2;
}
```

중간에도 얼마든지 return문이 올 수 있다.

NumCompare.c

실행결과

```
3과 4중에서 큰 수는 4 이다.
7과 2중에서 큰 수는 7 이다.
```

```
printf("3과 4중에서 큰 수는 %d 이다. \n", NumberCompare(3, 4));
printf("7과 2중에서 큰 수는 %d 이다. \n", NumberCompare(7, 2));

printf("3과 4중에서 큰 수는 %d 이다. \n", 4);
printf("7과 2중에서 큰 수는 %d 이다. \n", 7);
```

위의 두 문장 안의 NumberCompare 함수 호출 이후 왼쪽과 같이 된다.

26

다양한 종류의 함수 정의 2

```
int AbsoCompare(int num1, int num2);   // 절댓값이 큰 정수 반환
int GetAbsoValue(int num);     // 전달인자의 절댓값을 반환

int main(void)
{
    int num1, num2;
    printf("두 개의 정수 입력: ");
    scanf("%d %d", &num1, &num2);
    printf("%d와 %d중 절댓값이 큰 정수: %d \n",
        num1, num2, AbsoCompare(num1, num2));
    return 0;
}
int AbsoCompare(int num1, int num2)
{
    if(GetAbsoValue(num1) > GetAbsoValue(num2))
        return num1;
    else
        return num2;
}
int GetAbsoValue(int num)
{
    if(num<0)
        return num * (-1);
    else
        return num;
}
```

AbsoCompare.c

```
if(GetAbsoValue(num1) > GetAbsoValue(num2))
    . . . .

if( 5 > 9 )
    . . . .
```

GetAbsoValue 함수호출 이후

이 예제에서 보이듯이 함수의 호출문장은 어디에든 놓일 수 있다.

실행결과

```
두 개의 정수 입력: 5 -9
5와 -9중 절댓값이 큰 정수: -9
```

27

실습 문제 (Lab1)

- **키보드로부터 1개의 정수를 입력 받아, 1부터 입력 받은 정수까지의 합을 구해 출력하는 프로그램을 작성하시오.**
  - **정수 n값 입력은 main 함수에서 하고, 합의 계산은 t_sum이라는 함수를 만들어 수행하도록 하시오.**
  - **최종 합의 출력은 다시 main 함수에서 수행하시오.**
  - **입출력의 형태는 아래와 같다. (단, 입출력 숫자는 다를 수 있음)**

정수 값 1개를 입력하세요 (>0) : 5
1부터 5까지의 합은 15입니다.

28

실습 문제 (Lab2)

- 키보드로부터 평수를 제곱 미터로 바꾸어 출력하는 프로그램을 작성하시오.
  - 평수 값은 main 함수에서 표준입력으로 받으시오.
  - 평수를 제곱 미터로 바꾸는 것은 main 함수에서 수행하지 말고 pyung이라는 함수를 만들어 이 함수에서 수행하도록 하시오.
  - 바꾼 결과 출력은 main 함수에서 수행하도록 하시오.
  - 평수와 제곱 미터의 관계: 1 평 = 3.3 제곱 미터
  - 입출력의 형태는 다음과 같도록 하시오. (단, 입출력 숫자는 다를 수 있음)

```
면적을 평으로 입력하세요 :  25
입력한 25.00000평은 82.50000 제곱 미터입니다.
```

29

실습 문제 (Lab3)

- 세 개의 정수를 인자로 전달받아서 그 중 가장 큰 수를 반환하는 함수와 가장 작은 수를 반환하는 함수를 구현하여라. 그리고 이 함수들을 호출하는 적절한 main 함수도 작성하여라.
  - 가장 큰 수를 반환하는 함수 명: FindMax()
  - 가장 작은 수를 반환하는 함수 명: FindMin()
  - 세 개의 정수를 입력 받고, 출력하는 것은 main() 내에 구현할 것

```
세 개의 정수 값을 입력하세요: 100 300 -30
세 정수(100 300 -30) 중에서 제일 큰 수: 300
세 정수(100 300 -30) 중에서 제일 작은 수: -30
```

30

실습 문제 (Lab4)

- 인자로 전달된 수만큼의 피보나치 수열을 출력하는 함수를 구현하여라. 예를 들어서 프로그램 사용자가 5를 입력하면 0에서부터 시작하여 총 5개의 피보나치 수열을 출력해야 한다.
  - 피보나치 수열은 다음과 같다
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
  - 피보나치 수열은 0과 1에서 시작하며, 세 번째 이후의 수열은 이전 두 값의 합으로 구성된다.
  - 피보나치 수열을 생성하는 함수 명: Generate_Fibonacci()
  - 피보나치 수열의 개수는 2개 이상이어야 함

31

실습 시간 (2019년 6월 5일)

- 예제 문제: PrintfOutput.c, SimpleAddFunc.c, SmartAddFunc.c, NumCompare.c, AbsoCompare.c
- 실습 문제: Lab3, Lab4

32

9-2. 변수의 존재기간과 접근범위 - 지역변수

함수 내에만 존재 및 접근 가능한 "지역변수"

```
int SimpleFuncOne(void)
{
    int num=10;    // 이후부터 SimpleFuncOne의 num 유효
    num++;
    printf("SimpleFuncOne num: %d \n", num);
    return 0;    // SimpleFuncOne의 num이 유효한 마지막 문장
}

int SimpleFuncTwo(void)
{
    int num1=20;    // 이후부터 num1 유효
    int num2=30;    // 이후부터 num2 유효
    num1++, num2--;
    printf("num1 & num2: %d %d \n", num1, num2);
    return 0;    // num1, num2 유효한 마지막 문장
}
```

```
int main(void)
{
    int num=17;    // 이후부터 main의 num 유효
    SimpleFuncOne();
    SimpleFuncTwo();
    printf("main num: %d \n", num);
    return 0;    // main의 num이 유효한 마지막 문장
}
```

- 함수 내에 선언되는 변수를 가리켜 지역변수라 한다.
- 지역변수는 선언된 이후로부터 함수 내에서만 접근이 가능하다.
- 한 지역(함수) 내에 동일한 이름의 변수 선언 불가능하다.
- 다른 지역(함수)에 동일한 이름의 변수 선언 가능하다.
- 해당 지역을 빠져나가면 지역변수는 소멸된다. 그리고 호출될 때마다 새롭게 할당된다.

실행결과 LocalVariable.c

```
SimpleFuncOne num: 11
num1 & num2: 21 29
main num: 17
```

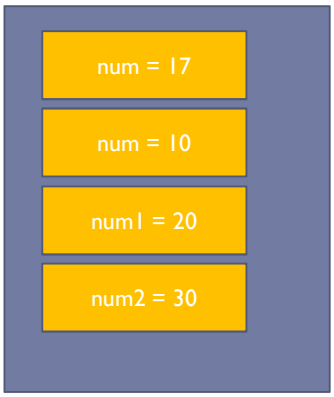
34

메모리 공간의 할당과 소멸 관찰하기

```
int SimpleFuncOne(void)
{
    int num=10;    // 이후부터 SimpleFuncOne의 num 유효
    num++;
    printf("SimpleFuncOne num: %d \n", num);
    return 0;    // SimpleFuncOne의 num이 유효한 마지막 문장
}

int SimpleFuncTwo(void)
{
    int num1=20;    // 이후부터 num1 유효
    int num2=30;    // 이후부터 num2 유효
    num1++, num2--;
    printf("num1 & num2: %d %d \n", num1, num2);
    return 0;    // num1, num2 유효한 마지막 문장
}
```

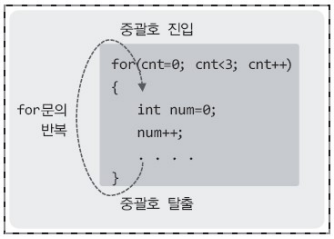
```
int main(void)
{
    int num=17;    // 이후부터 main의 num 유효
    SimpleFuncOne();
    SimpleFuncTwo();
    printf("main num: %d \n", num);
    return 0;    // main의 num이 유효한 마지막 문장
}
```

메모리 공간



35

다양한 형태의 지역변수



- **for문의 중괄호 내에 선언된 변수도 지역변수이다. 그리고 이 지역변수는 for문의 중괄호를 빠져나가면 소멸된다.**
- **따라서 for문의 반복횟수만큼 지역변수가 할당되고 소멸된다.**

AnotherLocalVar.c

LocalValHideVal.c

실행결과

```
if문 내 지역변수 num: 17
main 함수 내 지역변수 num: 1
```

주석처리 후 실행결과

```
if문 내 지역변수 num: 11
main 함수 내 지역변수 num: 11
```

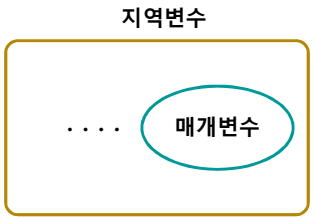
지역변수는 외부에 선언된 동일한 이름의 변수를 가린다.

```
int main(void)
{
    int num=1;
    if(num==1)
    {
        int num=7; // 이 행을 주석처리 하고 실행결과 확인하자!
        num+=10;
        printf("if문 내 지역변수 num: %d \n", num);
    }
    printf("main 함수 내 지역변수 num: %d \n", num);
    return 0;
}
```

if문 내에 선언된 변수 num이 main 함수의 변수 num을 가린다.

36

지역변수의 일종인 매개변수



매개변수는 일종의 지역변수이다.

- 매개변수도 선언된 함수 내에서만 접근이 가능하다.
- 선언된 함수가 반환을 하면, 지역변수와 마찬가지로 매개변수도 소멸된다.

37

9-3. 전역변수, static 변수, register 변수

전역변수의 이해와 선언방법

```
void Add(int val);
int num;    // 전역변수는 기본 0으로 초기화됨

int main(void)
{
    printf("num: %d \n", num);
    Add(3);
    printf("num: %d \n", num);
    num++;   // 전역변수 num의 값 1 증가
    printf("num: %d \n", num);
    return 0;
}

void Add(int val)
{
    num += val;    // 전역변수 num의 값 val만큼 증가
}
```

- 전역변수는 함수 외부에 선언
- 프로그램의 시작과 동시에 메모리 공간에 할당되어 종료 시까지 존재
- 별도의 값으로 초기화하지 않으면 0으로 초기화됨
- 프로그램 전체 영역 어디서든 접근 가능

```
num: 0
num: 3
num: 4
```

실행결과

GlobalVariable.c

지역변수의 이름이 전역변수의 이름을 가린다.

LocalValHideGlobalVal.c

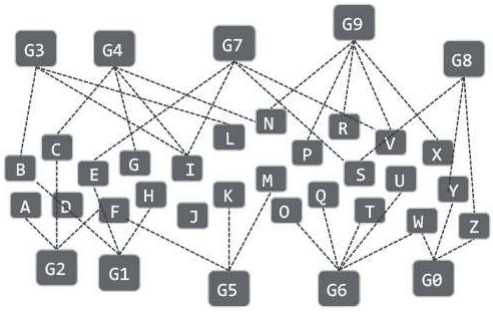
```
int Add(int val);
int num=1;

int main(void)
{
    int num=5;
    printf("num: %d \n", Add(3));
    printf("num: %d \n", num+9);
    return 0;
}
int Add(int val)
{
    int num=9;
    num += val;
    return num;
}
```

실행결과

```
num: 12
num: 14
```

39

전역변수! 많이 써도 되는가?



G0~G9의 전역변수와 함수와의 접근관계의 예시

전역변수! 많이 쓰면 좋지 않다.

전역변수의 변경은 전체 프로그램의 변경으로 이어질 수 있으며 전역변수에 의존적인 코드는 프로그램 전체 영역에서 찾아야 한다. 어디서든 접근이 가능한 변수이므로...

40

지역변수에 static 선언을 추가한 static 변수

```
void SimpleFunc(void)
{
    static int num1=0;    // 초기화하지 않으면 0 초기화
    int num2=0;     // 초기화하지 않으면 쓰레기 값 초기화
    num1++, num2++;
    printf("static: %d, local: %d \n",num1, num2);
}

int main(void)
{
    int i;
    for(i=0; i<3; i++)
        SimpleFunc();
    return 0;
}
```

- 선언된 함수 내에서만 접근이 가능 (지역변수 특성)
- 딱 1회 초기화되고 프로그램 종료 시까지 메모리 공간에 존재 (전역변수 특성)

StaticLocalVariable.c

실행결과

```
static: 1, local: 1
static: 2, local: 1
static: 3, local: 1
```

"난 사실 전역변수랑 성격이 같아. 초기화하지 않으면 전역변수처럼 0으로 초기화되고, 프로그램 시작과 동시에 할당 및 초기화되어서 프로그램이 종료될 때까지 메모리 공간에 남아있지! 그럼 왜 이 위치에 선언되었냐고? 그건 접근의 범위를 SimpleFunc로 제한하기 위해서야!"

static 지역변수의 발언!

프로그램이 실행되면 static 지역변수는 해당 함수에 존재하지 않는다.

41

static 지역변수는 좀 써도 되나요?

- 전역변수가 필요한 이유 중 하나는 다음과 같다.
  - 선언된 변수가 함수를 빠져나가도 계속해서 메모리 공간에 존재할 필요가 있다.
- 함수를 빠져나가도 계속해서 메모리 공간에 존재해야 하는 변수를 선언하는 방법은 다음 두 가지이다.
  - 전역변수, static 지역 변수
- static 지역변수는 접근의 범위가 전역변수보다 훨씬 좁기 때문에 훨씬 안정적이다.
  - static 지역변수를 사용하여 전역변수의 선언을 최소화하자.

AddToTotal.c: 전역변수 total ➔ static 지역변수 total

42

보다 빠르게! register 변수

```
int SoSimple(void)
{
    register int num=3;
    . . . .
}
```

- register는 힌트를 제공하는 키워드이다. 컴파일러는 이를 무시하기도 한다.
- 그리고 레지스터는 CPU 내부에 존재하는 메모리이기 때문에 접근이 가장 빠른 메모리 장치이다.

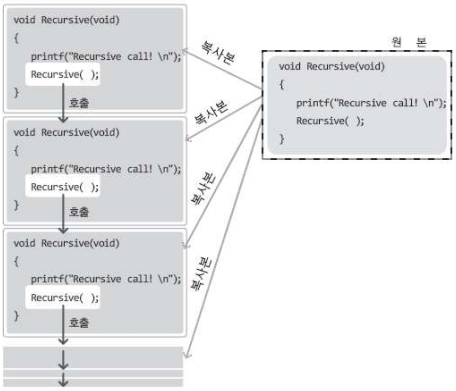
"이 변수는 내가 빈번히 사용하거든, 그래서 접근이 가장 빠른 레지스터에 저장하는 것이 성능향상에 도움이 될 거야"

register 변수 선언의 의미

43

9-4. 재귀함수에 대한 이해

재귀함수의 기본적인 이해



재귀함수 호출의 이해!

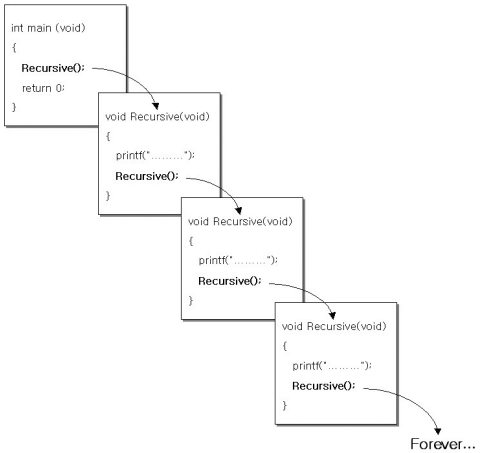
```
void Recursive(void)
{
    printf("Recursive call! \n");
    Recursive();   // 나! 자신을 재 호출한다.
}
```

자기자신을 재호출하는 형태로 정의된 함수를 가리켜 재귀함수라 한다.

45

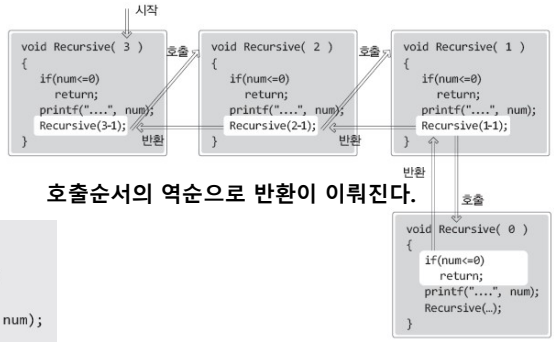
재귀 함수

- 탈출 조건의 필요성
  - 무한 재귀 호출을 피하기 위해서



46

탈출조건이 존재하는 재귀함수의 예



호출순서의 역순으로 반환이 이뤄진다.

```
void Recursive(int num)
{
    if(num<=0)      // 재귀의 탈출조건
        return; //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}

int main(void)
{
    Recursive(3);
    return 0;
}
```

실행결과

```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```

RecursiveFunc.c

Factorial 계산 (1/4)

- 재귀 함수 Design 사례
  - 팩토리얼(factorial) 계산을 위한 알고리즘

$$n! = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \ldots\ldots \times 2 \times 1}_{(n-1)!}$$

$$n! = n \times (n-1)!$$

$$f(n) = \begin{cases} n \times f(n-1) & n\text{이 1 이상인 경우} \\ 1 & n\text{이 0인 경우} \end{cases}$$

f(n) = n!
2! = 2*(2-1)! = 2*1! = 2*1 = 2
1! = 1*(1-1)! = 1*0! = 1*1 = 1
0! = 0*(0-1)! = ? (NO!!!)
0! = 더 이상 계산 없이 1

Factorial 계산 (2/4)

- 재귀 함수 Design 사례
  - 알고리즘을 코드로 옮기기 위한 pseudo code와 C 코드

```
// 시작 조건 : n은 0 이상이다.
시작(START) : fac(n) 호출
    1. 만약에 n이 0이면 1을 반환
    2. 그렇지 않다면 n * fac(n-1)을 반환
끝(END)
```

```
int fac(int n)
{
    if (n==0)
        return 1;
    else
        return n * fac(n-1);
}
```

49

Factorial 계산 (3/4)

```
int Factorial(int n)
{
    if(n==0)
        return 1;
    else
        return n * Factorial(n-1);
}

int main(void)
{
    printf("1! = %d \n", Factorial(1));
    printf("2! = %d \n", Factorial(2));
    printf("3! = %d \n", Factorial(3));
    printf("4! = %d \n", Factorial(4));
    printf("9! = %d \n", Factorial(9));
    return 0;
}
```

- C 언어가 재귀적 함수호출을 지원한다는 것은 그만큼 표현할 수 있는 범위가 넓다는 것을 의미한다!
- C 언어의 재귀함수를 이용하면 재귀적으로 작성된 식을 그대로 코드로 옮길 수 있다.

실행결과

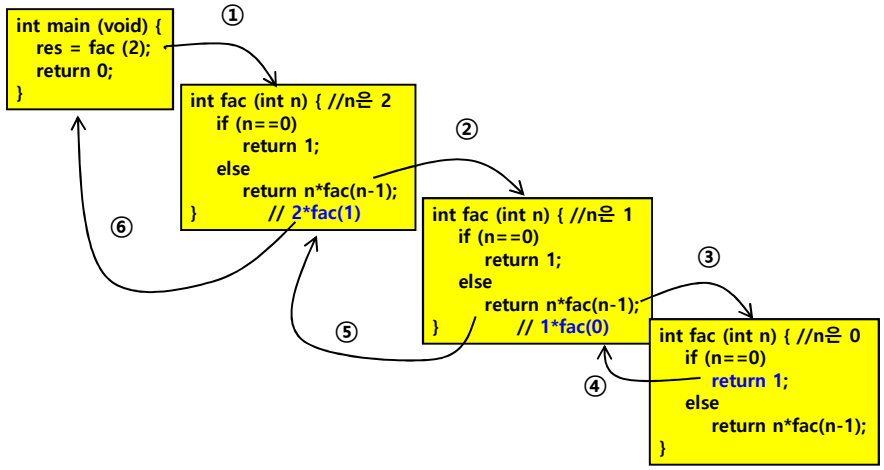
```
1! = 1
2! = 2
3! = 6
4! = 24
9! = 362880
```

RecursiveFactorial.c

50

Factorial 계산 (4/4)

- fac 함수 호출 및 복귀 상황



51

실습 문제 (Lab5)

- 키보드로부터 양의 정수 하나를 입력 받아 그 숫자만큼 Hello!를 출력하는 프로그램을 작성한다.
  - 재귀함수를 사용하여 프로그래밍하며, 재귀함수의 이름은 RecurPrint로 할 것
  - 즉, main 함수에서는 RecurPrint를 한 번만 호출하고, RecurPrint 함수 내에서 다시 RecurPrint 함수를 호출하는 형태로 프로그래밍 할 것

```
양의 정수 하나를 입력 : 4
4. Hello!
3. Hello!
2. Hello!
1. Hello!
```

52

실습 문제 (Lab6)

- 두 개의 정수를 입력 받아서 최대 공약수(GCD)를 구하는 프로그램을 작성하라.
  - 정수 num1과 num2의 최대 공약수는 num1이나 num2보다 클 수 없다.
  - num1과 num2의 최대공약수로 num1 또는 num2를 나누면 나머지가 0이 된다.
  - 함수 명: int Find_GCD (int num1, int num2)

```
두 개의 정수를 입력: 6  9
두 수의 최대공약수: 3
```

- 최소공배수(LCM) 구하는 방법은??

53

실습 문제 (Lab7)

- 필자가 좋아하는 것 중의 하나가 금요일 저녁 퇴근길에 DVD 한편을 빌리고, 동네 슈퍼에서 군것질거리를 사가지고 집에 들어가는 것이다. 오늘은 금요일이다. 잔돈을 하나도 남기지 않고 이 세 가지 물건을 하나 이상 반드시 구매하려면 어떻게 구매를 진행해야 하겠는가? 물론 여기에는 여러가지 경우의 수가 있을 것이다. 필자가 어떠한 선택을 할 수 있는지 여러분이 제시해주기 바란다.
  - DVD 한편 대여료가 1,500원 (변수의 상수화)
  - 크림빵: 500원, 새우깡: 700원, 콜라: 400원 (변수의 상수화)
  - 만족시키는 조합을 찾지 못했을 때는 "만족시키는 조합이 없습니다 " 라고 출력할 것.
  - 함수 형태: void find_opt (int num), num: 잔돈

```
현재 당신이 소유하고 있는 금액 (>= 5000 && <= 7000): 5000
크림빵 1개, 새우깡 2개, 콜라 4개
크림빵 2개, 새우깡 3개, 콜라 1개
크림빵 4개, 새우깡 1개, 콜라 2개
어떻게 구입하시겠습니까?
```

54

실습 문제 (Lab8)

- 프로그램 사용자로부터 초(second)를 입력 받은 후, 이를 (시, 분, 초)의 형태로 출력하는 프로그램을 작성하여라.
  - 함수 형태: void find_time (int second)

```
초(second) 입력: 3615
[h: 1, m: 0, s: 15]
```

55

실습 시간 (2019년 6월 12일)

- 예제 문제: LocalVariable.c, AnotherLocalVal.c, LocalValHideVal.c, GlobalVariable.c, LocalValHideGlobalVal.c, StaticLocalVariable.c, RecursiveFunc.c, RecursiveFactorial.c,
- 실습 문제: Lab5, Lab6, Lab7, Lab8

56

Chapter 09가 끝났습니다. 질문 있으신지요?