가변 인자 함수

2021년 1월 3일 일요일 오후 11:23

- printf( )같은 함수는 어떻게 만들까?
  - 왜 얘들은 출력/입력할 데이터의 자료형을 한 개나 그 이상을 넣을 수 있지?
  - 심지어는 각 데이터의 자료형이 달라도 됨

가변 인자 함수(variadic function)

- 가변 인자 함수

```
<반환형> <함수명>(<자료형이_정해진_매개변수_목록>, ...);
```

  - 정해지지 않은 수의 매개변수(가변 인자)를 허용하는 함수
    - 2개, 100개도 넣어도 됨
  - 반드시 최소 한 개의 정해진 자료형의 매개변수 필요
  - 가변 인자는 '...'로 표현
- 가변 인자 함수는 언제 사용함?
  - 많이 사용하지는 않지만 가끔 유용한 경우가 있음
    - 예: 메모리에 블록을 크게 잡아두고 거기에 여러 가지 자료형을 저장하는 함수
      - printf( )/scanf( ) 같은 것
- 가변 인자 함수의 예

```c
#include <stdarg.h>

int add_ints(const size_t count, ...)
{
    va_list ap;
    int sum;
    size_t i;

    sum = 0;
    va_start(ap, count);
    {
        for (i = 0; i < count; i++) {
            sum += va_arg(ap, int);
        }
    }
    va_end(ap);

    return sum;
}

int main(void)
{
    int result;

    result = add_ints(1, 16);
    printf("result: %d\n", result);

    result = add_ints(4, 1, 2, 3, 4);
    printf("result: %d\n", result);

    return 0;
}
```

```
result: 16
result: 10
_
```

va_로 시작하는 매크로 함수들

- va_list
  - 가변 인자 목록
  - va_start( ), va_arg( ), va_end( ) 매크로 함수를 사용할 때 필요한 정보가 포함
  - 명시되지 않은 자료형 (구현마다 다름)

```
va_list ap;
```

- va_start( )

```
va_start(<가변 인자 목록>, <가변 인자 시작하기 직전 매개변수>);
```

  - 매크로 함수
  - 함수 매개변수로 들어온 가변 인자들에 접근하기 전에 반드시 호출해야 함
  - va_list에 필요한 초기화를 수행
- va_end( )

```
va_end(<가변 인자 목록>);
```

  - 매크로 함수
  - 함수 매개변수로 들어온 가변 인자들에 접근이 끝난 뒤에 반드시 호출해야 함
  - 사용했던 가변 인자 목록을 정리함
    - 더 이상 가변 인자 목록을 사용할 수 없도록 가변 인자 목록의 값을 수정함

② 중괄호를 사용하여 가독성을 높이기

```
va_list ap;

va_start(ap, count);
{
    /* 코드 생략 */
}
va_end(ap);
```

  - 중괄호 블록을 넣어준 이유
    - 읽기 쉬우라고
    - 그래야 va_end( ) 넣는 걸 안 까먹음
      - 중괄호가 있으면 중괄호 밑에 va_end(ap)가 있다는 걸 알음
        그래서 va_end가 없을 때 발견 가능
- va_arg( )

```
va_arg(<가변 인자 목록>, <얻어올 가변 인자의 자료형>);
```

  - 매크로 함수
  - 가변 인자 목록으로부터 다음 가변 인자를 가져옴
  - 가져올 가변 인자의 자료형은 두 번째 매개변수로 알려줌
  - 예전 표준상의 문제로 가변 인자 목록의 기본 자료형 인자들은 다음과 같이 승격(promotion)됨
    - 모든 정수형은 int로
      - 어떤 자료형을 넣던 간에 정수형은 int로 스택 메모리에 들어간다
    - 모든 부동소수점은 double로
      - Float을 넣어도 스택 메모리에는 double 로 들어감
        - Printf에서 %f을 사용하면 float과 double을 읽어 올 수 있는 이유
  - 따라서, 두 번째 매개변수에는 int나 double을 쓸 것

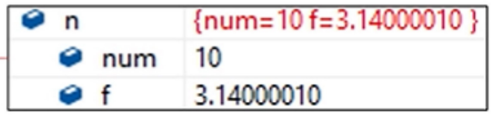
- 구조체도 가변 인자로 넣을 수 있음

```c
typedef struct {
    int num;
    float f;
} number_t;
```

```c
void do_something(const size_t count, ...)
{
    va_list ap;
    number_t n;

    va_start(ap, count);
    {
        n = va_arg(ap, number_t);
        /* 멋진 코드를 여기에 호호호 */
    }
    va_end(ap);
}
```

```c
number_t nums = { 10, 3.14f };
do_something(1, nums);
```



- va_arg는 어떻게 가변 인자 목록에서 자료를 읽지?
  - C에서 실행 중에 자동으로 자료형을 판단하는 기능이 없음
    - 사용자가 int를 읽을지 뭘 읽을지 넣어줘야 한다
  - 따라서 컴파일러가 이와 비슷한 코드를 컴파일할 수 있게 준비해줘야 할 듯
  - va_arg(ap, int); 가 전처리 되면 아래와 같은 코드로 바뀜

```c
void* p = data_block;
int value = *(int*)p;
```

    1. 스택 메모리의 어딘가를 가리키고 있는 데이터 블록(data_block)을 일단 가져옴
    2. 거기서 int형 하나를 읽어 옮
    3. 읽어오기 위해서는 p가 void형이기 때문에 int로 캐스팅하고 읽어 옴
- va_arg( )는 매크로 함수
  - 함수처럼 보이지만 엄밀한 의미의 함수는 아님
    - 스택 프레임을 만들지도
    - 매개변수를 전달하지도
    - 함수 주소로 점프하지도 않음
  - 그 대신 전처리기가 매크로 함수의 구현 코드로 대체시켜 줌

```c
val = va_arg(ap, int);
```

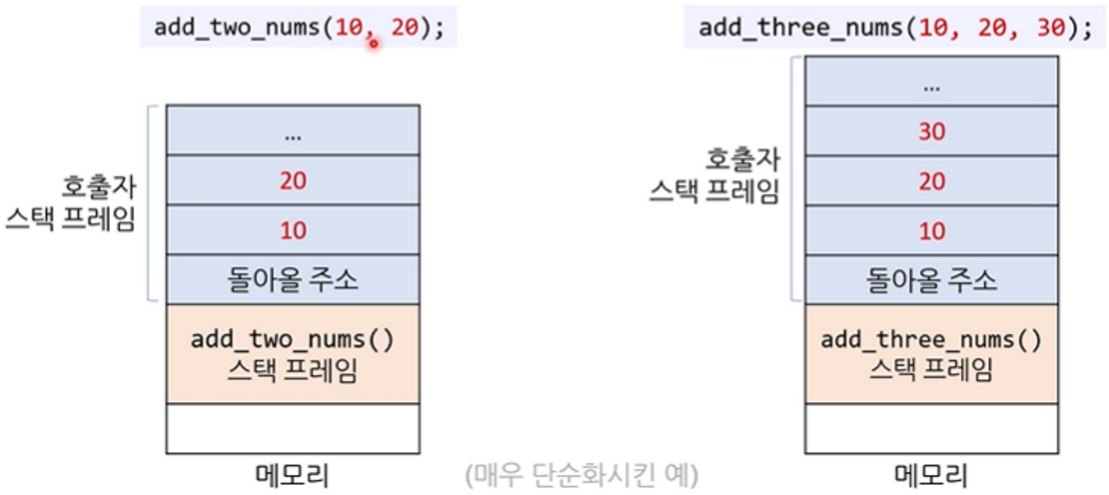
    - 이랬던 코드를 전처리기가

```c
val = *(int*)ap.data;
((int*)ap.data)++;
```

    - 이렇게 바꾸어 준다
      - 코드 설명
        1. 읽어온 데이터 ap.data를 int로 캐스팅하고 역참조하여 val에 저장
        2. 그리고 int 데이터 하나를 읽었으니까 다음 int 데이터로 이동(++연산으로)

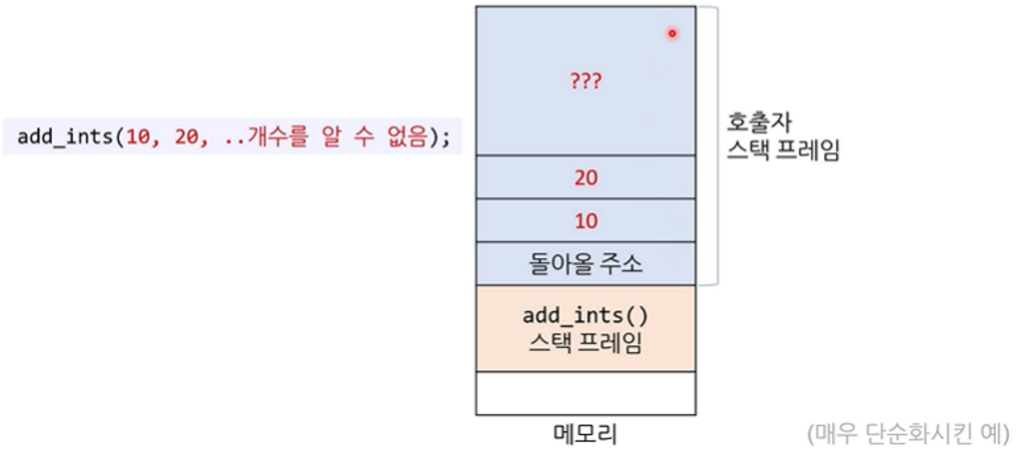
가변 인자 함수가 매개변수를 읽어오는 방법

- 어떻게 다른 수의 매개변수를 스택에 넣지?
  - 어떤 함수를 호출할 때마다 매개변수를 순서대로 스택에 넣어 줌



- 스택은 위에서 아래로 증가한다
  - 위에가 주소 크기가 크고, 아래 주소 크기가 작음
- 첫 번째 매개변수가 제일 마지막에 스택 프레임에 푸쉬 된다.

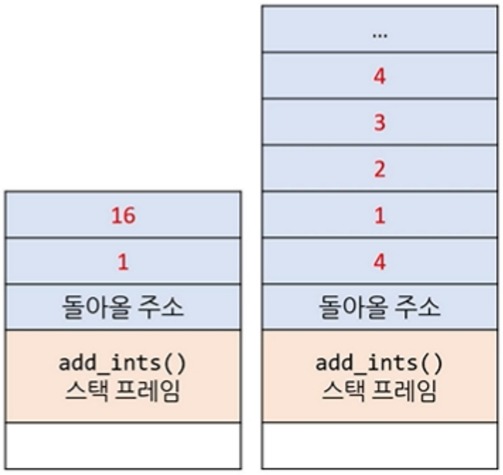
○ 가변 인자 함수는 매번 매개변수 개수가 바뀌는데 어떻게?



○ 가변 인자 함수를 호출하는 호출자는 매개변수를 몇 개를 넣어야 하는지 앎

```
add_ints(1, 16);          /* 2개 */
add_ints(4, 1, 2, 3, 4); /* 5개 */
```

- 따라서 그냥 다 넣어줌



- 문제는 호출을 받는 가변인자 함수는 나한테 몇 개가 들어오는지 알 수가 없다는 것

• 첫 번째 매개변수가 저장된 스택 메모리 위치는?

○ add_ints( ) 함수의 스택 프레임 바로 전

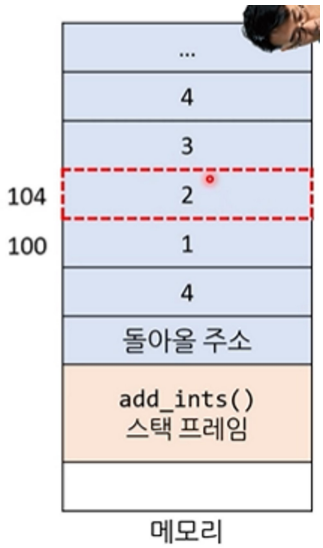
○ 따라서 호출받은 함수가 어디서 첫 번째 변수를 읽어와야 하는지 알 수 있다

```
add_ints(1, 16);          /* 2개 */
add_ints(4, 1, 2, 3, 4); /* 5개 */
```

- 가변 인자 함수는 이렇게 인자를 읽어온다
  1. va_start(ap, count)에서 가변 인자 시작 직전 매개변수(int 형)에 기초해서 가변 인자 목록의 시작 메모리 주소를 계산

```
va_start(ap, count);
```

```
ap.data = (char*)&count + sizeof(int);
```

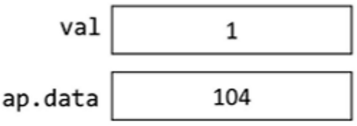


     - 실제 코드에선 sizeof(int)가 아닌 sizeof(count)가 들어가는 경우가 많다
  2. va_arg(ap, int);가 호출될 때마다 int 크기만큼 더해가며 읽을 위치를 변경하면 됨

```
va_arg(ap, int);
```

```
val = *(int*)ap.data;
((int*)ap.data)++;
```

     - 만약 ap.data가 1이 들어있는 메모리를 카리킨다면
     - 거기서 int 크기만큼 읽어 val 에 저장하고
     - 자기 자신을 int 크기만큼 증가시킴



   - 따라서 va_list는 스택 메모리에서 위치를 가리키는 포인터 같은 것을 가지고 있을 수 밖에

함수에서 매개변수로 가변 인자만 받을 수 있을까?

- 함수의 첫 번째 매개변수로 가변 인자 못 씀
  - 가변 인자(...) 앞에 자료형이 특정된 매개변수가 반드시 있어야 함
  - 가변 인자 뒤에 자료형이 정해진 매개변수가 있으면 안 됨
    - 함수가 정확히 어느 오프셋에서 읽어와야 하는지 컴파일 중에 특정 불가

```
void do_something(..., int);      /* 컴파일 오류 */
void do_something(int, ..., int); /* 컴파일 오류 */
void do_something(int, int, ...); /* OK */
```

  - 즉, 가변 인자 아닌 것을 우선 차례대로 읽음
    - 그 뒤, 가변 인자는 va_arg( )가 시키는 대로 하나씩 주소를 늘려가며 읽는 것
- 왜 그럴까?
  1. 가변 인자가 몇 갠지 가변 인자 함수는 모름
  - 실제 가변 인자가 몇 개 들어왔는지 호출된 함수는 모름
  - 그래서 앞의 예에선 매개변수 count로 제어

```
int add_ints(const size_t count, ...);
```

```
result = add_ints(1, 16);

result = add_ints(4, 1, 2, 3, 4);
```

  - 단, 스택 메모리의 어느 위치부터 가변 인자가 시작되는지는 앎

        - 가변 인자 함수는 스택 메모리에 가변 인자가 시작하는 부분부터 하나씩 읽는다
    2. 가변 인자의 자료형을 가변 인자 함수는 모름
    - 어떤 형의 가변 인자인지는 실행 중 결정
    - 그래서 가변 인자 함수 자체는 스택의 어떤 위치를 어떤 형으로 읽어야할 지 모름
    - 따라서, 정해진 자료형으로 넘겨주는 매개변수로부터 알아내야 함
        - add_int등 특정한 자료형만 전달하는 가변 인자 함수 말고
          int, double등 섞어서 매개변수로 전달하려면 어떻게 해야 할까?
            - printf를 보면 알 수 있다!

- printf의 첫 번째 매개변수
    - printf의 첫 번째 매개변수는 변환 문자열
        - 이 문자열이 앞서 본 가변 인자 함수가 알지 못하는 정보를 다 알려줌
            - 가변 인자 목록에 인자가 몇 개인지
            - 각 인자의 자료형이 무엇인지

```
int printf(const char* format, ...);
```

```
printf("%d", score);           /* 정수 하나를 읽어줘! */
printf("%s %f", msg, number);  /* 문자열 하나랑 부동소수점 하나를 순서대로 읽어줘! */
```

        - 잘못된 서식 지정자를 넣어주면 경고 준 이유도 바로 이거임

```
printf("%s", score); /* int score = 10; */
```

```
main.c:8:15: warning: format specifies type 'char *' but the argument has type
'int' [-Wformat]
        printf("%s", score);
                ~~   ^~~~~
                %d
```

            - 함수 호출상으로 보면 잘못된 서식 지정자를 넣는지 알 수 없다
            - 컴파일러가 우리를 위해 많은 일을 해주는 거임. printf가 워낙 많이 쓰이기도 하고
        - 변환 문자열에 들어간 서식 지정자 수가 매개변수보다 많으면 경고
            - 실제 개수 이상으로 읽어버리면 그 전의 스택 주소에 있는 값을 읽게 됨

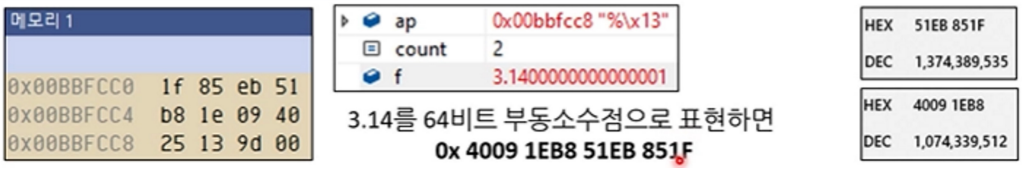
```
printf("%d %s", score); /* int score = 10; */
```

```
main.c:8:14: warning: more '%' conversions than data arguments [-Wformat]
        printf("%d %s", score);
                 ~^
```

- 데이터를 잘못된 자료형으로 읽으면 어떻게 될까?
    - 엉뚱한 값을 읽게 된다

```
result = add(2, 1374389535, 1074339512);
```

```
int add(const size_t count, ...)
{
    va_list ap;
    /* 코드 생략 */
    f = va_arg(ap, double);
    /* 코드 생략 */
}
```



        - 두 이상한 32비트의 int값을 64비트 double 하나에다 읽으니까 3.14가 되었다