21. 가변 인자(variable argument)

- printf()는 **가변 인자 리스트(variable argument list)**를 사용하는 C의 표준 함수입니다.
- 가변 인자에 대한 이해는 후에 C++의 가변 매크로와 가변 템플릿template을 이해하는 바탕이됩니다.
- 가변 인자를 받는 함수를 정의하면, 함수의 인자를 가변적으로 지정할 수 있습니다.
- 가변 인자 함수를 작성하기 위해서는 생략 심벌(ellipse symbol: **...**)과 형 **va_list**와 3개의 매크로 **va_start**, **va_arg**와 **va_end**를 사용합니다.
- stdarg.h를 포함(include)해야 합니다.

- 가변 인자를 사용하는 함수 Sum()을 만들어 봅시다.
- 함수 Sum()의 첫 번째 파라미터는 자신을 제외한 가변인자의 개수이며, 나머지는 정수 가변 인자입니다. Sum()은 정수 가변 인자의 합을 리턴합니다.
- Sum()은 다음과 같은 프로토타입(prototype)을 가집니다.

```
int Sum(int n, ...);
```

- 프로토타입에 사용된 심벌 ... 에 주목하세요.
- 이것은 Sum()함수가 가변 인자를 파라미터로 가짐을 명시합니다.
- 예를 들어 Sum(3,*5,6,7*); 의 결과는 18이 되어야 합니다.

- 이제 Sum()의 몸체(body)를 작성합니다.
- 먼저 va_list형의 변수를 한 개 선언해야 합니다(일반적으로 ap(Argument Pointer)로 선언합니다).
- 그리고 합을 저장할 변수 s를 0으로 초기화시키고, 인덱스 변수 i와 임시 변수 t를 선언합니다.

```
int Sum(int n,...) {
    int s=0,i,t;
    va_list ap;
}//Sum
```

• 다음에 va_start 매크로를 사용하여 va_list형의 변수 ap와 Sum()함수의 첫 번째 파라미터를 지정합니다.
• 이것은 ap가 Sum()의 두 번째 인자의 시작 주소를 가리키게 합니다.

```
int Sum(int n,...) {
    int s=0,i,t;
    va_list ap;

    va_start(ap,n);//이 문장은 ap가 n이후의 첫 인자의 시작 주소를 가리
                   //키게 한다.
}//Sum
```

• va_start 매크로 이후에 원하는 형의 변수 값을 읽기 위해 va_arg매크로를 ap와 함께 사용합니다.
• 예를 들어 정수 값을 읽고 싶으면 va_arg(ap,int); 실수 값을 읽고 싶으면 va_arg(ap,float); 를 사용합니다.
• 이것은 실제 ap가 두 번째에 지정된 형(type)만큼을 읽고 ap포인터를 sizeof(type)만큼 증가시키는 일을 합니다.

```
int Sum(int n,...) {
    int s=0,i,t;
    va_list ap;

    va_start(ap,n);
    for (i=0;i<n;++i) {
        t=va_arg(ap,int);//ap가 가리키는 곳에서 int만큼을 읽어온다.
                //ap는 sizeof(int)만큼 증가한다.
        s+=t;
    }//for
}//Sum
```

• 인자를 꺼내는 작업이 끝났으면, va_end 매크로를 사용하여 끝임을 명시하고, 나머지 작업을 합니다. va_end는 va_start와 쌍을 맞추기 위해 사용합니다.

```
#include <stdarg.h>
...
int Sum(int n,...) {
    int s=0,i,t;
    va_list ap;

    va_start(ap,n);
    for (i=0;i<n;++i) {
        t=va_arg(ap,int);
        s+=t;
    }//for
    va_end(ap);//(void)0; 이 문장이 없으면, 함수가 제대로 동작하지 않
          //는다.
    return s;
}//Sum
```

- 이 함수를 사용하는 예를 살펴봅시다.

```
printf("%i₩r₩n", Sum(3,5,6,7) );
```

- 위 문장의 출력 결과는 다음과 같습니다.

```
18
```

• 아래 문장의 출력결과는 얼마일까요?

printf("%i₩r₩n", Sum(2,5,6,7));

• 비록 3개의 가변 인자가 있음에도 불구하고 첫 번째 파라미터가 2이므로 결과는 11입니다.

• 아래 문장의 결과는 얼마일까요?

printf("%i₩r₩n", Sum(3,5L,6,7));

• 위 문장의 결과는 Visual Studio 6에서 11이었습니다. 그 이유는 다음과 같이 설명할 수 있습니다.
• 두 번째 파라미터의 형이 long임에 주목하세요. Sum()의 내부에서 va_arg(ap,int)를 사용하여 정수(integer)만을 꺼내므로, 결과는 5+0+6이 되어 11이 된 것이었습니다.

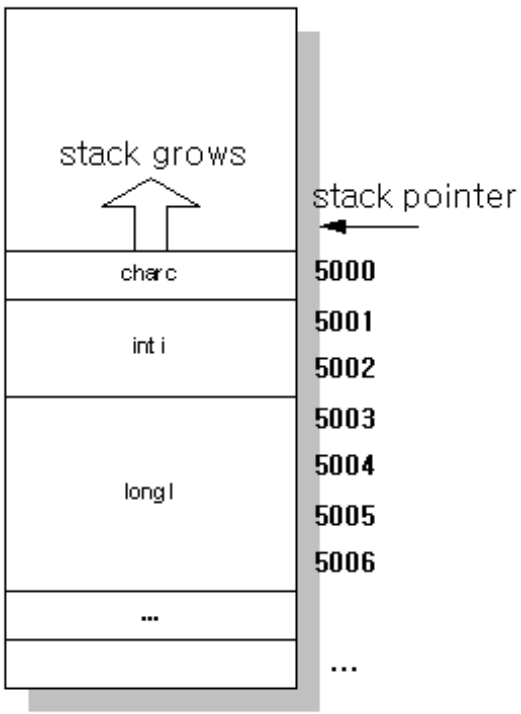
- 위 코드를 Visual Studio 2013에서 실행하면 결과가 18이 됩니다.
- 가변 인자를 전달하는 표준 방식이 변경되었기 때문인데, 정수 호환 타입을 가변 인자로 전달하면, 모두 4바이트의 메모리를 차지합니다.
- 이것을 **승격promotion**이라고 합니다.

스택 동작

• 다음 함수를 호출했을 때, 스택의 상태를 그려봅시다. 이 함수는 문자형, 정수형, 긴 정수형 인자를 가집니다.

```
int Add(char c, int i, long l);
```

• 스택의 상태는 아래 그림과 같습니다. 이 예에서는 int가 2바이트라고 가정합니다.



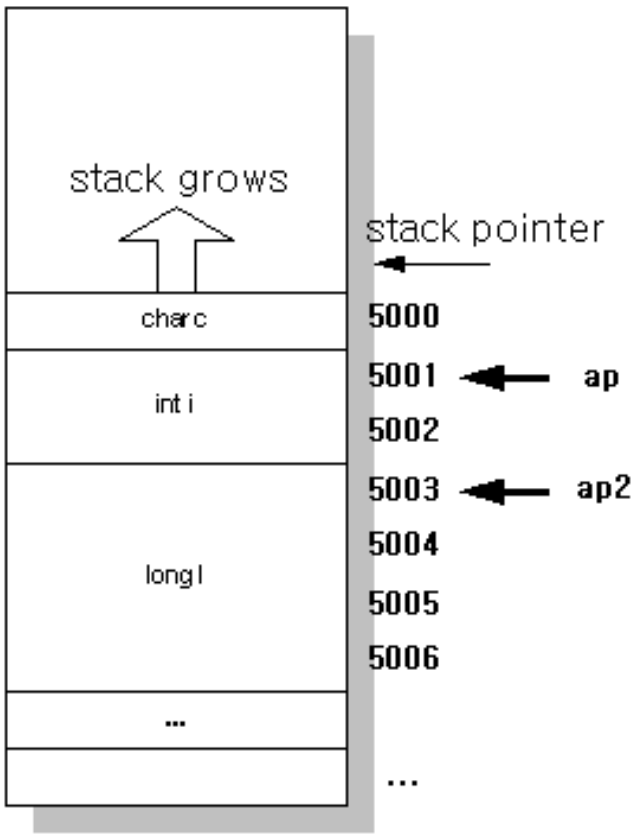
• 후에 int의 크기가 플랫폼마다 달라지는 특성이 문제가 되어, 모든 정수 호환 타입은 4바이트를 사용하는 표준이 정해졌습니다.
• 승격promotion이 적용된 컴파일러에서는 char, int 및 long이 모두 4바이트를 차지하므로 메모리의 구조는 다릅니다.

스택은 상위 주소에서 하위 주소로 자랍니다. 그러므로 먼저 l이 5003~5006번지에 할당되고, i가 5001~5002번지에 할당되고, 마지막으로 c가 5000번지에 할당됩니다.

- 우리의 목적은 인자의 직접적인 참조를 사용하지 않고 i와 l을 참조하려는 것입니다.
- 이것은 c의 주소를 알면 가능합니다.
- 포인터 변수 ap가 c를 가리키도록 한 다음(ap=5000), c의 크기 sizeof(c)만큼을 더 하면 (ap=ap+sizeof(c)), ap는 5001을 가리킵니다.
- 즉 정수 i를 가리키는 것이지요.
- 다시 ap가 l을 가리키도록 하려면, i의 크기만큼을 더하면 될 것입니다.

ap+=sizeof(i)

- 이제는 ap가 5003이 되어 l을 가리키게 됩니다.

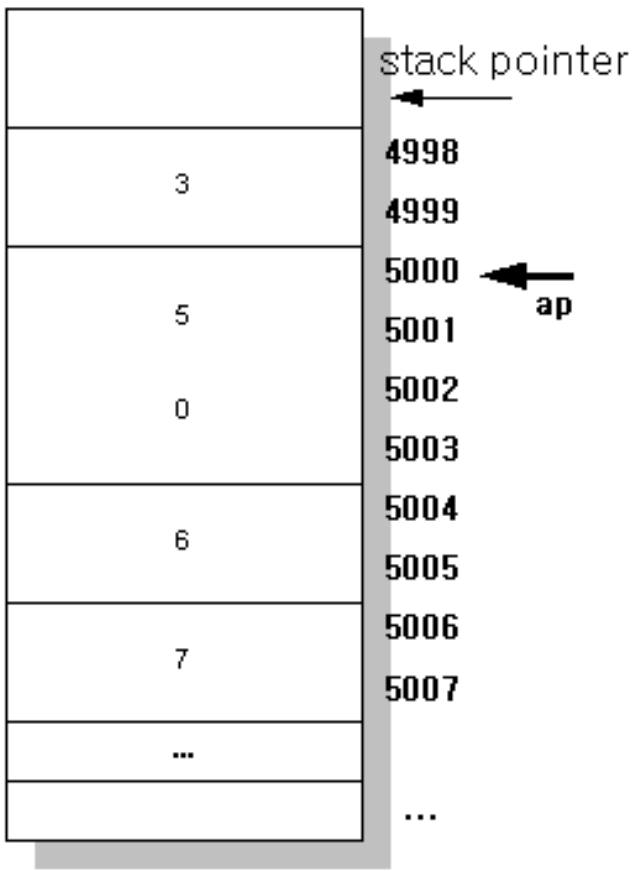


ap는 i(5001)을 가리킵니다. 이것은 5000에 sizeof(c)를 더한 결과입니다. ap2는 l(5003)을 가리킵니다. 이것은 ap에 sizeof(i)를 더한 결과입니다.

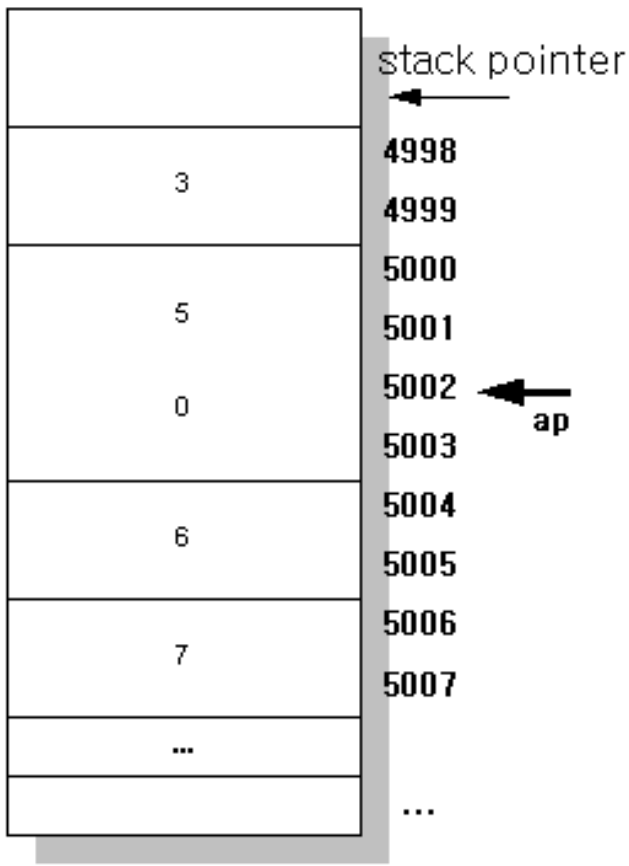
1개의 형과 3개의 매크로

printf("%i₩r₩n", Sum(3,5L,6,7));

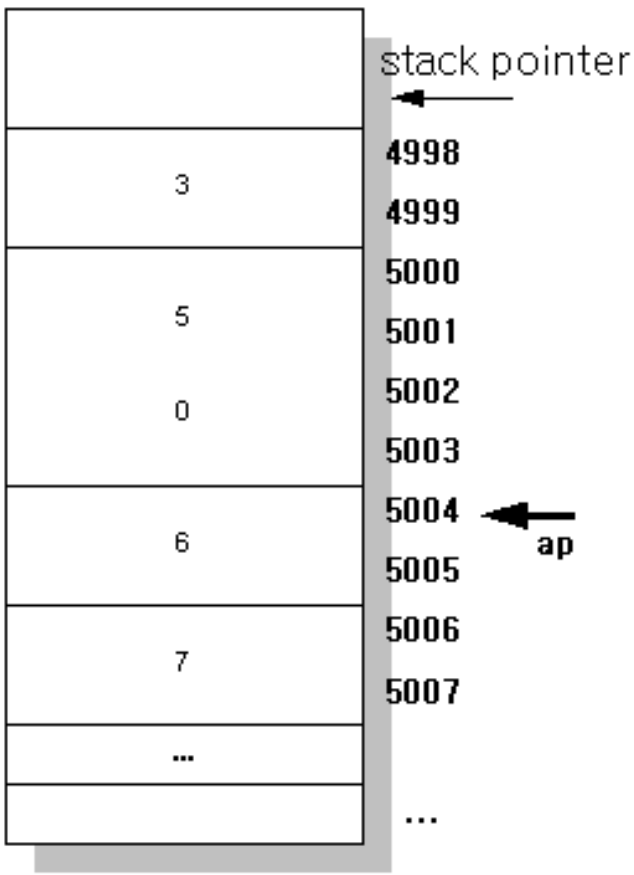
- 이제 위의 문장에서, 결과가 11이 찍힌 과정을 그림으로 그려보겠습니다.
- Win16 환경에서 Visual Studio 6 혹은 DOS 환경에서 볼런드 C++ 3.1을 사용했다고 가정합니다.



va_start(ap,n)은 ap가 [5000] 즉 3L의 상위 바이트를 가리키도록 합니다.
va_arg(ap,int)는 [5000]~[5001]의 5를 리턴하고 ap를 2(sizeof(int)) 증가시킵니다.



두 번째 va_arg(ap,int) 호출은 0을 리턴합니다. ap는 2증가되어 [5004]를 가리킵니다.



3번째 va_arg(ap,int)호출은 6을 리턴합니다. ap는 [5006]으로 설정됩니다.

다른 예(another example)

• 문맥 자유 문법(context free grammar)을 처리하기 위해 문법을 2차원 문자 배열에 저장하는 함수를 설계해 봅시다.

```
#include <string.h>
#include <stdarg.h>

#define MAX_GRAMMAR_NUM         30
#define MAX_GRAMMAR_LEN         20

int nGrammar;//number of valid grammars
char strGrammar[MAX_GRAMMAR_NUM][MAX_GRAMMAR_LEN];

void SetGrammar(int n, ...) {
    va_list ap;
    char* arg;
    int i;
```

```
    nGrammar=n;
    va_start(ap,n);
    for (i=0;i<nGrammar;++i) {
        arg=va_arg(ap,char*);
        strcpy(strGrammar[i],arg);
    }//for
    va_end(ap);
}//SetGrammar
```

• 전역 변수 strGrammar[]는 각각의 문자열을 유지하며, 이를 위해 strcpy()함수를 사용하고 있습니다.

• 스택에는 문자열의 시작 주소가 들어 있으므로, va_arg(ap,char*) 를 사용하여 차례대로 문자열을 꺼내고 있습니다.

- main()에서는 다음과 같이 문법 문자열을 설정할 수 있습니다.

```
...
void main() {

    SetGrammar(9,"1->23",
                 "2->2",
                 "3->24",
                 "4->25",
                 "5->65",
                 "6->7",
                 "7->8",
                 "8->9[3]",
                 "9->9");
}//main
```

실습문제

1. printf()의 이스케이프 절차(escape sequence) %d, %f, %ld등이 가변인자 리스트로 어떻게 동작하는지 설명하세요.

2. printf()와 똑같이 동작하는 MyPrintf()를 작성하세요.

3. printf()외에 가변인자 리스트를 사용하는 모든 표준 함수에 대해 설명하세요.

4. 윈도우즈 플랫폼의 구조적 예외 처리(structure exception handling)에 사용된 ...과 가변인자 리스트의 ...를 문법적으로 어떻게 구분할 수 있을까요?

@