Stack Frame

스택 프레임 동작 원리 이해하기

이다영

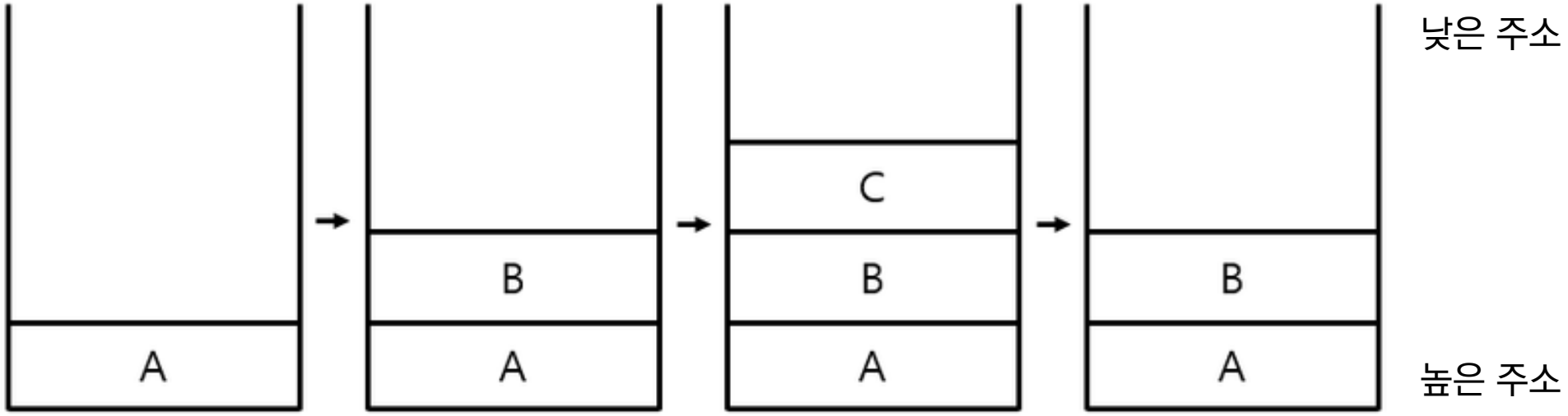
Stack : 스택

스택 메모리의 역할

1. 함수 내의 로컬 변수 임시 저장
2. 함수 호출 시 파라미터 전달
3. 복귀 주소(return address) 저장

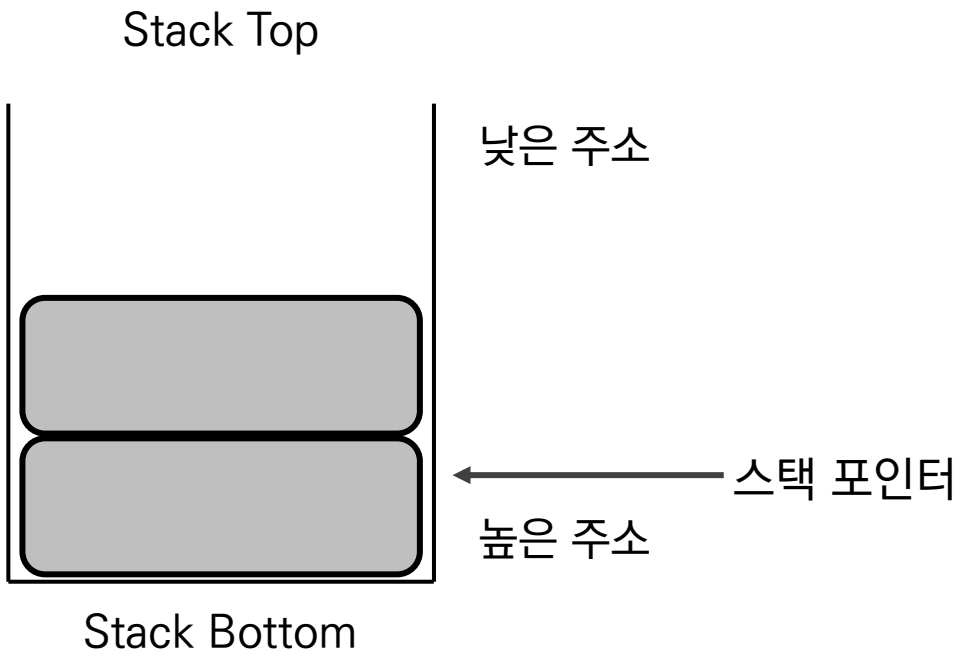
Stack : 스택

FILO 구조
(First In Last Out)



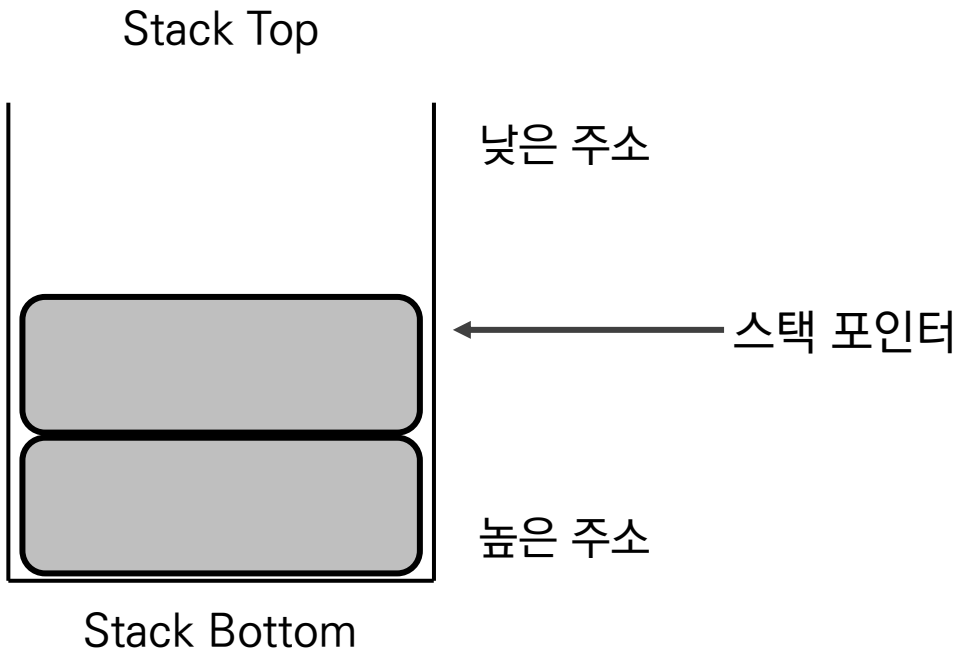
Stack : 스택

스택의 특징 ① PUSH



Stack : 스택

스택의 특징 ② POP



Stack Frame : 스택 프레임

스택 프레임이란?

ESP(스택 포인터)가 아닌
EBP(베이스 포인터) 레지스터를 사용하여
스택 내의 로컬 변수, 파라미터, 복귀 주소에
접근하는 기법

Stack Frame : 스택 프레임

스택 프레임 구조

```
PUSH EBP          : 함수 시작(EBP를 사용하기 전에 기존의 값을 스택에 저장)
MOV EBP, ESP      : 현재의 ESP(스택포인터)를 EBP에 저장
```

함수의 프롤로그

```
                  : 함수 본체
...               : 여기서 ESP가 변경되더라도 EBP가 변경되지 않으므로
                  : 안전하게 로컬 변수와 파라미터를 엑세스 할 수 있음
```

```
MOV ESP, EBP      : ESP를 정리 (함수 시작했을 때의 값으로 복원시킴)
POP EBP           : 리턴되기 전에 저장해 놓았던 원래 EBP 값으로 복원
RETN              : 함수 종료
```

함수의 에필로그

Stack Frame : 스택 프레임

실습을 통해 스택 프레임 동작 이해하기

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

Stack Frame : 스택 프레임

1. main() 함수 시작 시 스택 상태

```
ESP 0019FF2C
EBP 0019FF70
```

0019FF2C	004012A8	RETURN to StackFra.004012A8 from StackFra.ma
0019FF30	00000001	
0019FF34	004946D8	
0019FF38	00498380	
0019FF3C	35D58E2B	

Stack Frame : 스택 프레임

2. main() 함수 시작 & 스택 프레임 생성

```
004010A0  $ 55      PUSH EBP          ⟶ EBP 값을 스택에 백업
004010A1  . 8BEC    MOV EBP,ESP
```

```
ESP 0019FF28
EBP 0019FF28
```

```
0019FF28  0019FF70   ⟶ Main() 함수 시작할 때 EBP가 가지고 있던 초기값
0019FF2C  004012A8   RETURN to StackFra.004012A8 from StackFra.ma
0019FF30  00000001
0019FF34  004946D8
0019FF38  00498380
```

Stack Frame : 스택 프레임

3. 로컬변수 세팅

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
004010A3 . 83EC 08        SUB ESP,8          ⟶ 8바이트 메모리 공간 확보
004010A6 . C745 F8 050000( MOV DWORD PTR SS:[EBP-8],5
004010AD . C745 FC 0A0000( MOV DWORD PTR SS:[EBP-4],0A
```

⟶ [EBP-8] = 로컬변수 a
[EBP-4] = 로컬변수 b

⟶ a = 5
b = 10

Stack Frame : 스택 프레임

4. mul() 함수 파라미터 입력 & mul 함수 호출

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
004010B4 | . 8B45 FC       | MOV EAX,DWORD PTR SS:[EBP-4]
004010B7 | . 50            | PUSH EAX
004010B8 | . 8B4D F8       | MOV ECX,DWORD PTR SS:[EBP-8]
004010BB | . 51            | PUSH ECX
004010BC | . E8 BFFFFFFF   | CALL StackFra.mul
```

→ [EBP-4] = b에 있는 10을 스택에 넣음
[EBP-8] = a에 있는 5를 스택에 넣음

'파라미터의 역순 저장'

→ CPU는 무조건 '해당 함수가 종료될 때 복귀할 주소(return address')'를 스택에 저장함.

```
004010BC | . E8 BFFFFFFF   | CALL StackFra.mul
004010C1 | . 83C4 08       | ADD ESP,8
```

```
ESP 0019FF14
EBP 0019FF28

0019FF14  004010C1  RETURN to StackFra.ma
0019FF18  00000005
```

Stack Frame : 스택 프레임

5. mul() 함수 시작 & 스택 프레임 생성

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
00401080  r$ 55      PUSH EBP
00401081  .  8BEC    MOV EBP,ESP
```

```
ESP 0019FF10
EBP 0019FF10
```

```
0019FF10  r0019FF28  ⟶ 초기 EBP 값을 스택에 백업
0019FF14  00401061   RETURN to StackFra.main+21 from
0019FF18  00000005
```

Stack Frame : 스택 프레임

6. mul() 함수의 로컬 변수 x,y 세팅

```c
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
00401083  . 83EC 08     SUB ESP,8
00401086  . 8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
00401089  . 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
0040108C  . 8B4D 0C     MOV ECX,DWORD PTR SS:[EBP+C]
0040108F  . 894D F8     MOV DWORD PTR SS:[EBP-8],ECX
```

→ [EBP+8] = 파라미터 a
[EBP-4] = 로컬변수 x
[EBP+C] = 파라미터 b
[EBP-8] = 로컬변수 y

Stack Frame : 스택 프레임

7. IMUL 연산

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
00401092 | . 8B45 FC     | MOV EAX,DWORD PTR SS:[EBP-4]
00401095 | . 0FAF45 F8   | IMUL EAX,DWORD PTR SS:[EBP-8]
```

→ EAX = 5
EAX = EAX(5) * y(10) = 50

→ EAX = 산술 연산에 사용되는 레지스터 & 리턴 값으로 사용됨
-> 함수 리턴 직전에 EAX에 어떤 값을 입력하면 그대로 리턴 값이 됨.

Stack Frame : 스택 프레임

8. 스택 프레임 해제 & mul() 함수 종료

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
00401099 | . 8BE5      | MOV ESP,EBP  ⟶  ESP 값 복원
0040109B | . 5D        | POP EBP      ⟶  EBP 값 복원
```

```
ESP 0019FF14
EBP 0019FF28
```

```
0019FF14  004010C1  RETURN to StackFra.main+21 from StackFra.
0019FF18  00000005
0019FF1C  0000000A
```

```
0040109C └. C3        RETN
```

Stack Frame : 스택 프레임

9. mul() 함수 파라미터 제거

```c
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
00401041 | . 83C4 08        | ADD ESP,8
```

Stack Frame : 스택 프레임

10. printf() 함수 호출

```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
00401004   . 50              PUSH EAX  ⟶ 50
00401005   . 68 00214000      PUSH StackFra.00402100  ⟶ %d\n
0040100A   . E8 71FFFFFF      CALL StackFra.printf  ⟶ printf()
0040100F   . 83C4 08          ADD ESP,8  ⟶ 함수 파라미터 정리
```

Stack Frame : 스택 프레임

11. 리턴 값 세팅 & 스택 프레임 해제 & main() 함수 종료

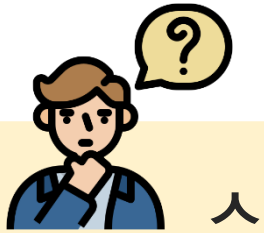
```
#include <stdio.h>

int mul(int a, int b)
{
int x = a, y = b;

return (x * y);
}

int main()
{
int a = 5, b = 10;

printf("%d₩n", mul(a, b));

return 0;
}
```

```
004010D2 | . 33C0      XOR EAX,EAX  ⟶ 0
004010D4 | . 8BE5      MOV ESP,EBP
004010D6 | . 5D        POP EBP
004010D7 | . C3        RETN
```

```
ESP 0019FF2C
EBP 0019FF70
```

main() 함수 시작할 때의 스택 모습과 완벽히 동일

Stack Frame : 스택 프레임



스택 프레임이란?

수시로 변경되는 ESP 레지스터 대신 EBP 레지스터를 사용하여
로컬 변수, 파라미터, 복귀 주소 등을 관리하는 방법

스택 프레임을 이해하면?

함수 파라미터, 함수 로컬 변수, 리턴 값 등이 처리되는 동작 원리를 이해하면서
디버깅 실력이 같이 향상됨!

감사합니다 ☺