Cykor 1주차 과제 보고서

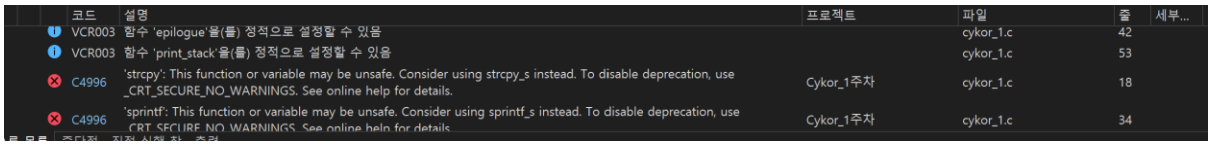
2024350223_원정재

우선 만들어야 하는 4개의 에필로그 함수, 프롤로그 함수, push 함수, pop 함수를 모두 따로 함수로 선언하여 func1,func2,func3,main 내부에 함수 호출을 통하여 코드를 구현하였습니다.

보고서의 설명방식은 제 코드의 부분적 사진이 먼저 있고 그와 관련된 설명이 그 사진 아래에 적혀있는 방식입니다.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#define STACK_SIZE 50
```

코드 윗부분부터 하나하나 자세하게 설명해보자면 우선 아래 사진과 같은 오류로 인해 익숙한, strcpy와 sprintf를 쓰다보니 마치 scanf 함수를 처음 쓰면 나오는 오류처럼 #define _CRT_SECURE_NO_WARNINGS를 추가해주었습니다, strcpy와 sprintf의 역할 및 사용이유는 이후에 해당 부분 함수에서 자세하게 다루도록 하겠습니다.



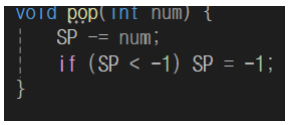
| 코드 | 설명 | 프로젝트 | 파일 | 줄 | 세부... |
|---|---|---|---|---|---|
| VCR003 | 함수 'epilogue'을(를) 정적으로 설정할 수 있음 | | cykor_1.c | 42 | |
| VCR003 | 함수 'print_stack'을(를) 정적으로 설정할 수 있음 | | cykor_1.c | 53 | |
| C4996 | 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details. | Cykor_1주차 | cykor_1.c | 18 | |
| C4996 | 'sprintf': This function or variable may be unsafe. Consider using sprintf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details. | Cykor_1주차 | cykor_1.c | 34 | |

또한 <string.h> 라는 이미 존재하는 헤더파일을 따로 또 include 해주었는데 그 이유는

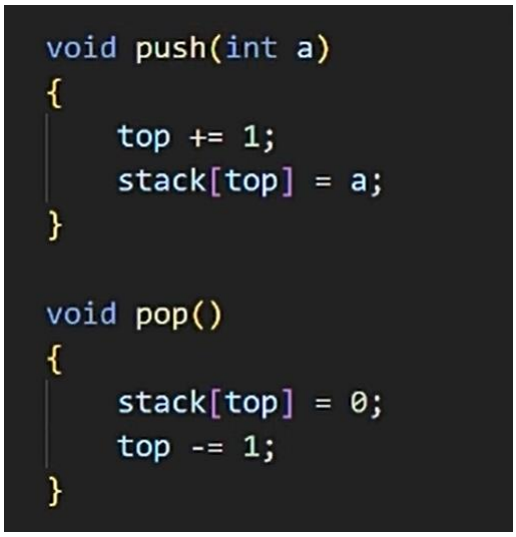
```
void push(int num, char* info) {
    call_stack[++SP] = num;
    strcpy(stack_info[SP], info);
}
```

이제 과제로 직접 짜야하는 4가지 함수들을 차례로 살펴보겠습니다. 우선 push 함수입니다. Push 함수는 원래 자료구조 stack에서도 자주 보는 간단한 함수인데 1주차 강의영상에서 call stack을 짜면서 선배님께서 강조하신 부분은 항상 call_stack 전역변수와 그에 대한 설명인 stack_info 전역변수가 항상 하나의 세트로 같이 push되고 pop 되는 일심동체로 움직여야 한다는 점이었습니다. 그래서 간단한 push 함수지만 이 코드에서 중요한 부분은 SP의 값을 증가해주는 연산자를 이용해 두 개의 전역변수의 인덱스 부분에 SP를 넣어주는 부분입니다. 이때 stack_info 같은 경우에는 문자열이기 때문에 strcpy 라는 문자열 관련 문자열 복사 함수를 사용하였고 이 때문에 #define

_CRT_SECURE_NO_WARNINGS 이것이 필요합니다. 물론 안전한 strncpy를 사용하여도 됩니다.




두번째 함수인 pop 함수를 보면 한마디로 스택에 쌓인 값을 제거하는 함수인데




위의 사진은 선배님이 1주차 강의 영상에서 자료구조 stack 의 일반적인 push함수와 pop 함수를 소스코드로 설명해주신 부분을 찍어온 것입니다. 이를 가져온 이유는 제가 구현한 부분 pop 함수와의 차이점을 간략하게 짚고 넘어가기 위해서 입니다.

총 2가지 치이점이 있는데 먼저 실제 데이터 삭제 없이 SP 위치만 변경해본 것입니다. 선배님은 삭제한 위치를 0으로 초기화 시켜주었지만 저는 그 과정을 없애보았습니다. 이번 과제에서 요구하는 최종결과 출력 자체만을 위한 코드로 보면 무의미하다고 생각했기 때문입니다. 하지만 이는 스택 메모리에 남아있는 데이터가 메모리 잔재로 남아 공격자에게 유출될 위험성이 있긴 합니다.

다음으로 pop 함수에 매개변수를 추가하여 매번 1개씩 요소를 제거하는 것이 아닌 효율적으로 제거할 요소의 개수를 받아 한번에 SP의 위치를 조정했습니다.

SP-=num 즉 num 만큼 SP 의 값을 바꾸지만 if (SP < -1) 을 통해 SP 의 값의 최솟값인 -1을 안전상의 이유로 보장하는 코드를 추가하였습니다.

```
void prologue(char* function) {

    push(-1, "Return Address");


    char SFP[20];
    sprintf(SFP, "%s SFP", function);
    push(FP, SFP);


    FP = SP;
}
```

3번째로 프롤로그 함수인데 선배님께서 강의영상에서 설명해주신 프롤로그 함수의 역할 3가지가 지금 함수 본문 코드 상에서 1줄씩 띄어서 구분되어 있는 각각의 3블록에 대응됩니다. -1의 값을 가지는 반환주소값을 push 해주는 1단계 , 나중에 돌아갈 SFP 값을 가지는 (즉, 이전 FP 값을) 스택에 저장하는 2단계, 그리고 지금 push 함수가 사용되면서 새로 갱신된 SP 값을 새로운 FP로 만드는 3단계 입니다. 이때 코드상에서는 sprintf 함수를 사용하였는데 1학년 c언어 시간에 배운 함수로는 strcpy(SFP, function); strcat(SFP, " SFP"); 이렇게 2개의 함수를 사용하여도 되지만 검색해보니 간결성을 위해 이 함수 하나만을 사용해도 똑같은 효과를 낼 수 있습니다.

```
void epilogue() {

    SP = FP;


    FP = call_stack[SP];


    pop(2);
}
```

4번째 함수인 에필로그 함수를 살펴보면 이것도 프롤로그 함수처럼 크게 3가지 부분입니다. 가장 먼저 지역변수를 제거하기 위해 SP의 값을 FP 값으로 옮겨서 지역변수에 해당하는 위치들을 제거하는 1단계, 저장된 SFP 값으로 FP 값을 옮겨두는 2단계, 스택에 들어있는 SFP와 반환주소를 pop 해내는 3단계입니다. 여기서 살펴보고 갈 부분은

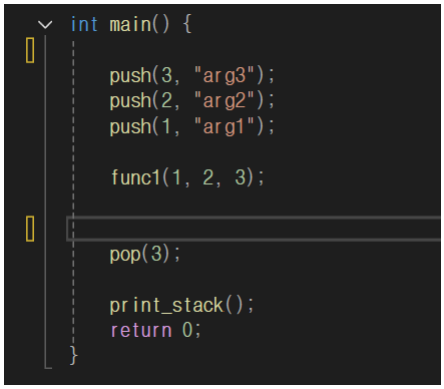
```
for (int i = SP; i >= 0; i--) {
    if (call_stack[i] != -1)
        printf("%d : %s = %d", i, stack_info[i], call_stack[i]);
    else
        printf("%d : %s", i, stack_info[i]);
```

위의 사진이 이미 선배님께서 주신 print_stack 함수의 일부분인데 최종 출력결과를 일부분만 보면

```
====== Current Call Stack ======
10 : var_2 = 200    <=== [esp]
9 : func2 SFP = 4    <=== [ebp]
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================
```

Call_stack [i]== -1 일 경우는 else 문이 실행되어 stack_info만 출력되고 call_stack 부분은 출력되지 않는다는 것을 볼 수 있습니다. 그래서 func1 SFP 옆에는 -1 이라는 값이 생략된 것이고 func2 SFP 옆에는 =4 라는 값이 출력되어 있습니다.

이제는 main() 부터 시작해서 func1, func2, func3 내부의 코드구현을 살펴보고 최종적으로 6개의 printf_stack() 함수를 통해 터미널 창에 나오는 결과를 ppt의 예제 과제 결과 코드랑 비교해 보겠습니다.



```
int main() {

    push(3, "arg3");
    push(2, "arg2");
    push(1, "arg1");

    func1(1, 2, 3);


    pop(3);

    print_stack();
    return 0;
}
```

우선 main 함수에서 func1 함수를 호출하기전 func1의 매개변수 3개를 오른쪽에서 왼쪽 순서로 3개를 차례대로 push() 해줍니다.

그러면 func1 함수에 이제 진입하는데 **(재귀함수 개념에서도 중요한건데 여기서 중요한 건 지금 func1 함수를 코드 도중에 만났기 때문에 아직 main 함수는 끝나지 않았으며 stack에 남아있고 func1 다음부분인 pop(3) 이전에 멈춰있다고 생각하면서 코드를 짜야 한다는 것입니다)**

```
void func1(int arg1, int arg2, int arg3)
    int var_1 = 100;


    prologue("func1");


    push(var_1, "var_1");

    print_stack();


    push(13, "arg2");
    push(11, "arg1");

    func2(11, 13);


    pop(2);

    print_stack();


    epilogue();
}
```

Func1 함수의 내부에서 프롤로그 함수를 만나서 위에서 설명한 총 3단계의 프롤로그 과정을 진행하고 이제 func1의 내부 지역변수를 push 해줍니다. 이때 func1 입장에서 main 이전에 FP는 새로 갱신된 값이 없으므로 -1 이라는 것이 중요합니다.

이러면 func1의 정보들이 들어있는 stack을 1번째 printf_stack()으로 확인해줍니다.

그 후 func1 함수에서는 func2를 호출하기전 func2 함수의 매개변수 2개를 오른쪽에서 왼쪽 순서로 push() 해줍니다. 그 후 fun2()를 호출합니다.

```
void func2(int arg1, int arg2) {
    int var_2 = 200;


    prologue("func2");


    push(var_2, "var_2");

    print_stack();


    push(77, "arg1");

    func3(77);


    pop(1);

    print_stack();


    epilogue();
```

이제 다시 func2 함수 내부에서 프롤로그 함수를 만나서 3단계 프롤로그 과정을 진행 후 func2의 내부 지역변수를 push 해줍니다.

이러면 이제 func2 의 내부 지역변수까지 stack에 쌓인 모습을 2번째 print_stack()을 통해 확인합니다.

이번에도 유사하게 func3 함수를 호출하기전  func3의 매개변수 1개를 push() 해줍니다.

```
void func3(int arg1) {
    int var_3 = 300;
    int var_4 = 400;


    prologue("func3");


    push(var_3, "var_3");
    push(var_4, "var_4");

    print_stack();


    epilogue();
}
```

그 후 func3에 대한 3단계 프롤로그를 진행한 후 func3 의 지역변수 2개를 push() 해줍니다.

그러면 최종적으로 func3까지 스택에 차있는 모습을 볼 수 있는 3번째 print_stack() 함수가 나옵니다.

그러면 이제 드디어 func3의 첫 에필로그 함수가 작동하는데 에필로그 함수 역시 위의 설명에서 3단계라고 이해할 수 있었습니다. 즉, func3의 지역변수를 제거하고 FP를 이전 함수의 SFP로 옮긴 뒤(func2로 갑니다) pop(2)를 통해 반환주소값과 SFP 를 스텍에서 빼내는 것입니다.

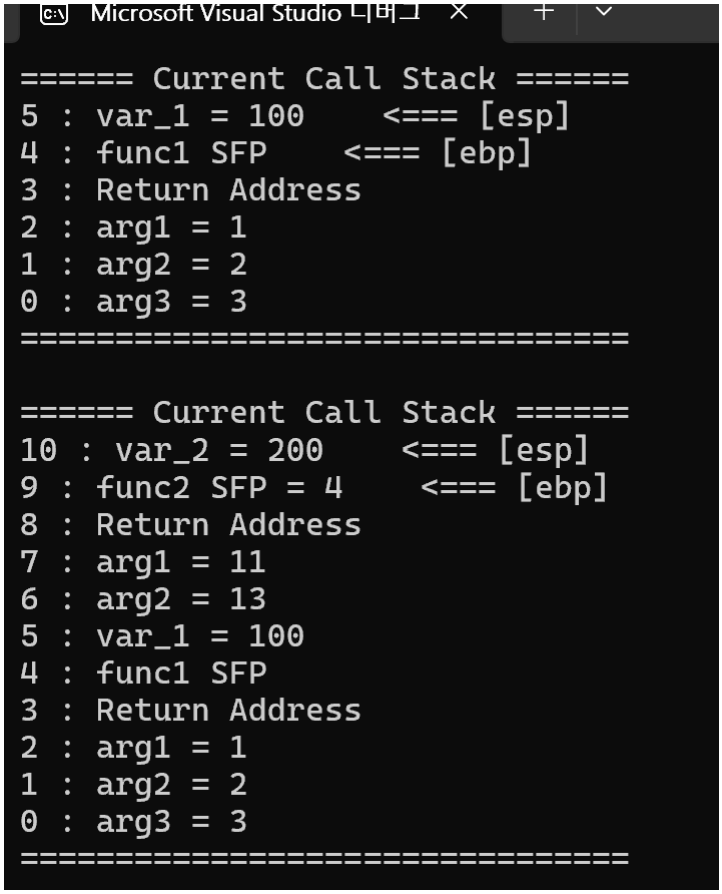
**이러면 코드상에서 어디까지 온것이냐 하면 이제 func2 의 pop(1) 을 할 차례입니다. 이건 func2 의 내부에서 func3를 호출하기전 func3의 매개변수 1개를 push() 해줬었기 때문에 func2에서 이 func3의 매개변수 1개를 pop() 해주는 역할까지 해야한다는 것입니다.**

이제 4번째 print_stack을 통해 func3 관련된 정보들이 없어진 걸 확인 할 수 있습니다.

그 후 func2의 에필로그 함수를 통해 func2의 정보들을 매개변수 전까지 위와 같이 빼냅니다. 그러면 이제 func2 함수가 완전히 끝나고 func1 의 pop(2) 로 돌아오게 됩니다.

여기서 5번째 print_stack을 통해 func1과 관련된 정보만 스텍에 남아있는 것을 확인합니다.

그 후 똑같이 func1 함수의 에필로그를 수행하면서 func1 함수를 끝내고 main() 함수로 돌아오게 됩니다. 그러면 main 함수에서 func1의 매개변수 3개 정보를 pop(3) 해주고 마지막 6번째 print_stack() 함수를 통해 비어있는 스택이 출력됩니다.




```
====== Current Call Stack ======
15 : var_4 = 400    <=== [esp]
14 : var_3 = 300
13 : func3 SFP = 9    <=== [ebp]
12 : Return Address
11 : arg1 = 77
10 : var_2 = 200
9 : func2 SFP = 4
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
==========
```

```
====== Current Call Stack ======
10 : var_2 = 200    <=== [esp]
9 : func2 SFP = 4    <=== [ebp]
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

====== Current Call Stack ======
5 : var_1 = 100    <=== [esp]
4 : func1 SFP    <=== [ebp]
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================

Stack is empty.

C:\Users\rudy0\source\repos\Cykor_1주차
이 창을 닫으려면 아무 키나 누르세요...
```