Cykor 1주차 과제

2025350039 김성현

목차

- 전체 구조
- Push
- Pop
- Prologue
- Epilogue
- AI edit

전체 구조

- push, pop, prologue, epilogue 기능을 함수로 구현
- 함수 내부에서 직접 매개변수와 지역변수의 값을 할당한 배열을 각각 prologue, epilogue 함수로 전달하여 call_stack을 작성하는 데 활용
- func2와 func3에는 func1과 마찬가지로 prologue와 epilogue를 배치

```
*/
void push(int push_list[], int len, char type){ ... }

void pop(int len, char type){ ... }

void prologue(int param_list[], int local_list[], int par_len, int loc_len){ ... }

void epilogue(int par_len){ ... }

void print_stack(){ ... }


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;
    // 매개변수, 지역변수를 할당한 배열을 제작해서 수동 초기화
    int param_list[3] = { arg1,arg2,arg3 };
    int local_list[1] = { var_1 };

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    prologue(param_list, local_list, 3, 1);

    print_stack();


    func2(11, 13);

    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    epilogue(2);

    print_stack();
}
```

prologue

- **전달 인자**

- **매개변수 배열** : 함수 별로 매개변수들의 값과 개수가 매번 달라지기 때문에 배열 형식으로 초기화하여 전달
- **매개변수 배열 길이** : 변수 개수 직접 계산 필요 X -> 외부에서 길이 초기화 해서 push 함수에 전달
- **지역변수 배열** : 함수 별로 지역변수들의 값과 개수가 매번 달라지기 때문에 배열 형식으로 초기화하여 전달
- **지역변수 배열 길이** : 변수 개수 직접 계산 필요 X -> 외부에서 길이 초기화 해서 push 함수에 전달

```
void prologue(int param_list[], int local_list[],
              int par_len, int loc_len) {
    //push 함수에 형식에 맞춰 집어넣기 위해 딱히 
    int NULL_list[1] = {};
    //매개 변수 push
    push(param_list, par_len, 'p');
    //반환 주소 push
    push(NULL_list, 1, 's');
    //SFP push
    push(NULL_list , 1, 'f');
    //지역 변수 push
    push(local_list, loc_len, 'l');
}
```

prologue

- **작동방식**

- **매개변수 push → 반환주소 push → SFP push → 지역변수 push**
- **NULL_list** : push 함수는 전달 인자로 (int[ ], int, char)를 요구하는데, proglogue 내부의 4가지 push를 모두 push 함수 하나에서 처리하기 위해 매개변수 개수 맞추는 용도로 빈 배열 삽입.
- **type** : push 함수 가장 마지막에 입력하는 문자는 push 함수에서 연산 방식을 결정하는 인자. 'p' 는 매개변수, 'l'는 지역변수, 's'는 반환주소, 'f'는 SFP를 의미.

```
void prologue(int param_list[], int local_list[],
              int par_len, int loc_len) {
    //push 함수에 형식에 맞춰 집어넣기 위해 딱히 ㄹ
    int NULL_list[1] = {};
    //매개 변수 push
    push(param_list, par_len, 'p');
    //반환 주소 push
    push(NULL_list, 1, 's');
    //SFP push
    push(NULL_list , 1, 'f');
    //지역 변수 push
    push(local_list, loc_len, 'l');
}
```

push

- **전달 인자**

- **push할 변수들 값이 저장된 배열** : push 해야 할 변수들의 값과 개수가 매번 달라지기 때문에 배열 형식으로 초기화하여 전달

- **해당 배열의 길이** : 센티넬 값을 활용해 변수 개수를 구하려 시도 -> 불안정성 & 고정된 시나리오이기에 굳이 push 함수에서 변수 개수를 구할 필요 X -> 외부에서 초기화 된 값을 활용해 코드 간략화 목적으로 사용 -> 변수 개수만큼 반복해서 call_stack에 할당

- **push하는 변수의 종류 식별 문자** : 매개변수, 지역변수, SFP, 귀환주소를 push 할 때 과정이 전부 다르므로 타입을 입력 받아 push 함수 1개에서 4가지 전부 처리할 수 있도록 조건문 설정.

```
void push(int push_list[], int len, char type) {

    //변수, 함수 이름 뒤에 숫자를 붙이기 위해 임시로 이름 저장하는 버퍼
    char temp[10];

    //지역 변수와 함수 이름 뒤의 숫자는 push 함수 종료 시에도 초기화 X
    static int local = 1;
    static int func = 1;

    //매개 변수, 지역 변수 구분해서 call_stack과 stack_info 설정

    if (type == 'p') {
        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = push_list[i - 1];
            sprintf_s(temp, "arg%d", i);
            strcpy_s(stack_info[SP], temp);
        }
    }
    else if (type == 'l') {
        for (int i = 0; i < len; i++) {
            //지역변수에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = push_list[i];
            sprintf_s(temp, "var_%d", local);
            strcpy_s(stack_info[SP], temp);
            local++;
        }
    }
    else if (type == 'f'){

        for (int i = len; i > 0; i--) {
            //sfp에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = FP;
            sprintf_s(temp, "func%d SFP", func);
            strcpy_s(stack_info[SP], temp);
            func++;
            FP = SP;

        }
    }
    else if (type == 's') {
        //귀환 주소의 call_stack값인 -1 할당, stack_info에 Return Address 할당
        SP++;
        call_stack[SP] = -1;
        strcpy_s(stack_info[SP], "Return Address");
    }
```

push

- **작동방식**
- **'l'의 경우** : SP를 하나 올려 변수를 push 할 공간 확보 → 해당 SP의 call_stack에 전달받은 변수 배열에 있는 변수 값 할당 → 이때 'l'는 위쪽부터 push되어야 하므로 앞번호부터 할당 →변수 명을 stack_info에 입력하기 위해 sprintf와 strcpy사용 → 변수명 뒤에 들어가는 숫자는 push 함수 종료 후에 초기화 되면 안되므로 static 선언
- **'s'의 경우** : SP를 하나 올려 변수를 push 할 공간 확보 → 귀환주소의 타입이므로 call_stack에 -1을 저장 → 귀환주소의 타입이므로 stack_info에 "Return Address"를 저장

```
void push(int push_list[], int len, char type) {

    //변수, 함수 이름 뒤에 숫자를 붙이기 위해 임시로 이름 저장하는 버퍼
    char temp[10];

    //지역 변수와 함수 이름 뒤의 숫자는 push 함수 종료 시에도 초기화 X
    static int local = 1;
    static int func = 1;

    //매개 변수, 지역 변수 구분해서 call_stack과 stack_info 설정

    if (type == 'p') {
        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = push_list[i - 1];
            sprintf_s(temp, "arg%d", i);
            strcpy_s(stack_info[SP], temp);
        }
    }
    else if (type == 'l') {
        for (int i = 0; i < len; i++) {
            //지역변수에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = push_list[i];
            sprintf_s(temp, "var_%d", local);
            strcpy_s(stack_info[SP], temp);
            local++;
        }
    }
    else if (type == 'f'){

        for (int i = len; i > 0; i--) {
            //sfp에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = FP;
            sprintf_s(temp, "func%d SFP", func);
            strcpy_s(stack_info[SP], temp);
            func++;
            FP = SP;

        }
    }
    else if (type == 's') {
        //귀환 주소의 call_stack값인 -1 할당, stack_info에 Return Address 할당
        SP++;
        call_stack[SP] = -1;
        strcpy_s(stack_info[SP], "Return Address");
    }
```

push

- **'p'의 경우** : SP를 하나 올려 변수를 push 할 공간 확보 → 해당 SP의 call_stack에 전달받은 변수 배열에 있는 변수 값 할당 → 이때 'p'는 오른쪽부터 push되어야 하므로 뒤 번호부터 할당 →변수 명을 stack_info에 입력하기 위해 sprintf와 strcpy사용 → 변수명 뒤에 들어가는 숫자는 push 함수 종료 후에 초기화 되면 안되므로 static 선언
- **'f'의 경우** : SP를 하나 올려 변수를 push 할 공간 확보 → SFP의 타입이므로 이전 FP를 call_stack에 저장 → 함수 명을 stack_info에 입력하기 위해 sprintf와 strcpy사용 → 변수명 뒤에 들어가는 숫자는 push 함수 종료 후에 초기화 되면 안되므로 static 선언 → 이후 FP를 SP값으로 끌어올려 새 함수의 FP설정.

```
void push(int push_list[], int len, char type) {

    //변수, 함수 이름 뒤에 숫자를 붙이기 위해 임시로 이름 저장하는 버퍼
    char temp[10];

    //지역 변수와 함수 이름 뒤의 숫자는 push 함수 종료 시에도 초기화 X
    static int local = 1;
    static int func = 1;

    //매개 변수, 지역 변수 구분해서 call_stack과 stack_info 설정

    if (type == 'p') {
        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = push_list[i - 1];
            sprintf_s(temp, "arg%d", i);
            strcpy_s(stack_info[SP], temp);
        }
    }
    else if (type == 'l') {
        for (int i = 0; i < len; i++) {
            //지역변수에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = push_list[i];
            sprintf_s(temp, "var_%d", local);
            strcpy_s(stack_info[SP], temp);
            local++;
        }
    }
    else if (type == 'f'){

        for (int i = len; i > 0; i--) {
            //sfp에 값을 할당하면서 SP값을 올림
            SP++;
            call_stack[SP] = FP;
            sprintf_s(temp, "func%d SFP", func);
            strcpy_s(stack_info[SP], temp);
            func++;
            FP = SP;

        }
    }
    else if (type == 's') {
        //귀환 주소의 call_stack값인 -1 할당, stack_info에 Return Address 할당
        SP++;
        call_stack[SP] = -1;
        strcpy_s(stack_info[SP], "Return Address");
    }
```

pop

- **전달 인자**
- **pop할 변수들의 개수** : pop 해야 할 변수들의 값과 개수가 매번 달라지기 때문에 배열 형식으로 초기화하여 전달
- **pop하는 변수의 종류 식별 문자** : 매개변수, 지역변수, SFP, 귀환주소를 pop 할 때 과정이 전부 다르므로 타입을 입력 받아 pop 함수 1개에서 4가지 전부 처리할 수 있도록 조건문 설정.

```
void pop(int len, char type) {

    if (type == 'l') {
        while (SP != FP) {
            //지역변수에 값을 할당하면서 SP값을 올림
            call_stack[SP] = 0;
            SP--;
        }
    }
    else if (type == 'f') {
        FP = call_stack[SP];
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 's') {
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 'p') {

        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 SP값을 올림
            call_stack[SP] = 0;
            SP--;
        }

    }
```

pop

- **작동방식**
- **'l'의 경우** : 코드 안정성을 위해 먼저 call_stack을 0으로 초기화 → SP를 FP위치까지 1씩 감소시키며 반복
- **'f'의 경우** : FP를 현재 call_stack 값인 SFP값으로 바꿔서 이전 함수의 FP로 복구 → SFP값을 정리하기 위해 call_stack값을 0으로 초기화 → SP값을 정리하여 SFP 최종 정리

```
void pop(int len, char type) {

    if (type == 'l') {
        while (SP != FP) {
            //지역변수에 값을 할당하면서 SP값을 올림
            call_stack[SP] = 0;
            SP--;
        }
    }
    else if (type == 'f') {
        FP = call_stack[SP];
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 's') {
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 'p') {

        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 SP값을 올림
            call_stack[SP] = 0;
            SP--;
        }

    }
```

pop

- **작동방식**
- **'s'의 경우** : call_stack에 반환 주소 값인 -1이 담겨있으므로 0으로 초기화 → SP를 1만큼 감소시켜 반환주소 값 정리
- **'p'의 경우** : 지역 변수와 달리 변수와 다른 함수를 구분해 줄 FP값이 존재 X → 외부에서 매개변수 개수를 초기화 → call_stack 값을 0으로 초기화 하고, SP값을 1씩 내리는 과정의 반복을 정확히 매개변수의 개수만큼 실행

```
void pop(int len, char type) {

    if (type == 'l') {
        while (SP != FP) {
            //지역변수에 값을 할당하면서 SP값을 올림
            call_stack[SP] = 0;
            SP--;
        }
    }
    else if (type == 'f') {
        FP = call_stack[SP];
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 's') {
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 'p') {

        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 SP값을 올림
            call_stack[SP] = 0;
            SP--;
        }

    }
```

epilogue

- **전달인자**

- **매개 변수의 개수** : 값을 제거하는 pop 함수의 특성상 push 함수와 달리 외부에서 변수 값 등을 입력 받을 필요가 X → 그러나 매개변수의 경우 FP로 변수의 개수를 계산하는 것이 불가 → 외부에서 초기화 하여 입력

```
void epilogue(int par_len) {

    //지역 변수 pop
    pop(0,'l');
    //SFP pop
    pop(0,'f');
    //반환 주소 pop
    pop(0,'s');
    //매개 변수 pop
    pop(par_len, 'p');
}
```

epilogue

- **작동방식**
- **지역변수 pop → SFP pop → 반환주소 pop → 매개변수 pop**
- **prologue와 비교하여** : prologue 함수의 경우 p-s-f-l 순으로 push를 진행함 **/** epilogue 함수의 경우 pop을 l-f-s-p 순으로 진행함 (stack이 LIFO 구조이기 때문)

```
void epilogue(int par_len) {

    //지역 변수 pop
    pop(0,'l');
    //SFP pop
    pop(0,'f');
    //반환 주소 pop
    pop(0,'s');
    //매개 변수 pop
    pop(par_len, 'p');
}
```

AI edit

- **AI 수정 추천**

- **pop 함수** : 코드 구조가 유사한 타입이 's' 일 때와 'f'일 때를 하나로 합치고, call_stack 뿐 아니라 stack_info까지 초기화를 하도록 추천

```
void pop(int len, char type) {

    if (type == 'p') {
        while (SP != FP) {
            //매개변수에 값을 할당하면서 S
            call_stack[SP] = 0;
            SP--;
        }
    }
    else if (type == 's') {
        FP = call_stack[SP];
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 'f') {
        call_stack[SP] = 0;
        SP--;
    }
    else if (type == 'l') {

        for (int i = len; i > 0; i--) {
            //매개변수에 값을 할당하면서 S
            call_stack[SP] = 0;
            SP--;
        }

    }
```

```
void pop(int len, char type) {
    if (type == 'l') {
        for (int i = 0; i < len; i++) {
            strcpy_s(stack_info[SP], "");
            call_stack[SP] = 0;
            SP--;
        }
    }
    else if (type == 'p') {
        while (SP != FP) {
            strcpy_s(stack_info[SP], "");
            call_stack[SP] = 0;
            SP--;
        }
    }
    else if (type == 's' || type == 'f') {
        strcpy_s(stack_info[SP], "");
        call_stack[SP] = 0;
        FP = call_stack[SP]; // FP 업데이트
        SP--;
    }
}
```

AI edit

- **AI 수정 추천**

- **prologue 함수** : 4가지 push를 모두 1가지 함수에서 처리하기 위해 생성한 NULL_list가 불필요하게 낭비되므로 삭제하고, 대신 반환주소와 SFP만을 push하는 push_single 함수를 따로 작성하여 활용하는 것을 추천

```c
void push_single(int value, char type) {
    SP++;
    call_stack[SP] = value;

    if (type == 's') {
        strcpy_s(stack_info[SP], "Return Address");
    }
    else if (type == 'f') {
        sprintf_s(stack_info[SP], "func%d SFP", FP);
        FP = SP;
    }
}
```

`prologue()` **에서 적용!**

```c
void prologue(int param_list[], int local_list[], int par_len, int loc_len) {
    push(param_list, par_len, 'p');
    push_single(-1, 's');            // 반환 주소 푸시
    push_single(FP, 'f');            // SFP 푸시
    push(local_list, loc_len, 'l');
}
```

AI edit

- **AI 수정 추천**

```
//stack overflow 방지
if (SP >= STACK_SIZE) {
    return;
}
```

- **push 함수 & print_stack 함수** : 주어진 시나리오 상에서는 SP가 STACK_SIZE를 초과할 일이 없지만, 오버플로우의 가능성이 없진 않기에 오버플로우 방어 코드 작성 하는 것을 추천

```
void print_stack()
{
    if (SP == -1)
    {
        printf("======Stack is empty.======₩n");
        return;
    }
    else if (SP >= STACK_SIZE) {
        printf("======Stack Overflow!!!======₩n");
        return;
    }
```