CyKor 1 week

2020190627 박민용

Call Stack 구현(초기 코드)

```
#include <iostream>
#include <cstring>
using namespace std;
#define STACK_SIZE 50

int call_stack[STACK_SIZE];
char info_stack[STACK_SIZE][20];
int SP=-1;
int FP=-1;

void push_value(int value, const char* info) {
    SP++;
    call_stack[SP]=value;
    strcpy(info_stack[SP], info);
}

void push_SFP(const char* info) {
    SP++;
    call_stack[SP]=FP;
    strcpy(info_stack[SP], info);
}

void push_RA() {
    SP++;
    call_stack[SP]=-1;
    strcpy(info_stack[SP], "Return Address");
}

int pop() {
    int val=call_stack[SP];
    SP--;
    return val;
}

void print_stack() {
```

```
    if(SP==-1) {
        cout<<"Stack is empty."<<"₩n";
        return;
    }

    cout<<"====== Current Call Stack ======"<<"₩n";

    for(int i=SP; i>=0; i--) {
        if(call_stack[i]!=-1)
            cout<<i<<" : "<<info_stack[i]<<" = "<<call_stack[i];

        else
            cout<<i<<" : "<<info_stack[i];

        if(i==SP)
                        cout<<"     <=== [esp]";

        else if(i==FP)
                        cout<<"     <=== [ebp]";

        cout<<"₩n";
    }

    cout<<"================================"<<"₩n₩n";
}

void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);

void func1(int arg1, int arg2, int arg3) {
    push_value(arg3, "arg3");
    push_value(arg2, "arg2");
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func1 SFP");
    FP=SP;
    push_value(100, "var_1");
    print_stack();

    func2(11, 13);
```

```
    pop();
    SP=FP;
    int before_FP=pop();
    FP=before_FP;
    pop();
    pop();
    pop();
    pop();
    print_stack();
}

void func2(int arg1, int arg2) {
    push_value(arg2, "arg2");
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func2 SFP");
    FP=SP;
    push_value(200, "var_2");
    print_stack();

    func3(77);
    pop();
    SP=FP;
    int before_FP=pop();
    FP=before_FP;
    pop();
    pop();
    pop();
    print_stack();
}

void func3(int arg1) {
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func3 SFP");
    FP=SP;
    push_value(300, "var_3");
    push_value(400, "var_4");
    print_stack();

    pop();
```

```
    pop();
    SP=FP;
    int before_FP=pop();
    FP=before_FP;
    pop();
    pop();
    print_stack();
}

int main() {
    func1(1, 2, 3);
    print_stack();
    return 0;
}
```

Call Stack 주요 함수 및 로직 분석

push_value(매개변수/지역변수를 push하는 경우)

```
void push_value(int value, const char* info) {
    SP++;
    call_stack[SP]=value;
    strcpy(info_stack[SP], info);
}
```

SP++;

SP 1 증가

call_stack[SP]=value;

call_stack[SP]에 매개변수/지역변수 저장

strcpy(info_stack[SP], info);

strcpy를 이용해 info_stack[SP]에 매개변수/지역변수의 정보 저장

push_SFP(SFP를 push하는 경우)

```
void push_SFP(const char* info) {
    SP++;
    call_stack[SP]=FP;
    strcpy(info_stack[SP], info);
}
```

SP++;

SP 1 증가

call_stack[SP]=FP;

call_stack[SP]에 FP 저장

strcpy(info_stack[SP], info);

strcpy를 이용해 info_stack[SP]에 FP의 정보 저장

push_RA(Return Address를 push하는 경우)

```
void push_RA() {
    SP++;
    call_stack[SP]=-1;
    strcpy(info_stack[SP], "Return Address");
}
```

SP++;
SP 1 증가

call_stack[SP]=-1;
call_stack[SP]에 -1 저장 (과제조건)

strcpy(info_stack[SP], "Return Address");
strcpy를 이용w해 info_stack[SP]에 Return Address의 정보 저장(“Return Address”)

pop()

```
int pop() {
    int val=call_stack[SP];
    SP--;
    return val;
}
```

int val=call_stack[SP];
call_stack[SP]를 val에 저장

SP--;
SP 1 감소

return val;
pop함수의 결과로 val 반환

func1

```
void func1(int arg1, int arg2, int arg3) {
    push_value(arg3, "arg3");
    push_value(arg2, "arg2");
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func1 SFP");
    FP=SP;
    push_value(100, "var_1");
    print_stack();

    func2(11, 13);
    pop(); //var_1
    SP=FP;
    int before_FP=pop(); //SFP
    FP=before_FP;
    pop(); //Return Address
    pop(); //arg1
    pop(); //arg2
    pop(); //arg3
    print_stack();
}
```

```
push_value(arg3, "arg3");
push_value(arg2, "arg2");
push_value(arg1, "arg1");
```

push_value를 이용해 매개변수 arg1, arg2, arg3를 call_stack에 insert
info_stack에는 “arg1”, “arg2”, “arg3”로 정보 저장

```
push_RA();
```

push_RA를 이용해 -1을 call_stack에 insert
info_stack에는 "Return Address"로 정보 저장

```
push_SFP("func1 SFP");
```

push_SFP를 이용해 FP를 call_stack에 insert
info_stack에는 "func1 SFP”로 정보 저장

```
FP=SP;
```

function prologue 순서에 의해 SFP 저장 후 FP=SP update

```
push_value(100, "var_1");
```

push_value를 이용해 지역변수 100을 call_stack에 insert
info_stack에는 "var_1"로 정보 저장

print_stack();
print_stack을 이용해 stack 정보 출력

func2(11, 13);
func2(11, 13) call

pop(); //var_1
pop을 이용해 지역변수 삭제

SP=FP;
function epilogue 순서에 의해 지역변수 삭제 후 SP=FP update

int before_FP=pop(); //SFP
FP=before_FP;
pop을 이용해 before_FP에 SFP 저장과 동시에 삭제
FP=before_FP update

pop(); //Return Address
pop을 이용해 Return Address 삭제

pop(); //arg1
pop(); //arg2
pop(); //arg3
pop을 이용해 매개변수 arg1, arg2, arg3 삭제

print_stack();
print_stack을 이용해 stack 정보 출력

func2

```
void func2(int arg1, int arg2) {
    push_value(arg2, "arg2");
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func2 SFP");
    FP=SP;
    push_value(200, "var_2");
    print_stack();

    func3(77);
    pop(); //var_2
    SP=FP;
    int before_FP=pop(); //SFP
    FP=before_FP;
    pop(); //Return Address
    pop(); //arg1
    pop(); //arg2
    print_stack();
}
```

push_value(arg2, "arg2");
push_value(arg1, "arg1");
push_value를 이용해 매개변수 arg1, arg2를 call_stack에 insert
info_stack에는 “arg1”, “arg2”로 정보 저장

push_RA();
push_RA를 이용해 -1을 call_stack에 insert
info_stack에는 "Return Address"로 정보 저장

push_SFP("func2 SFP");
push_SFP를 이용해 FP를 call_stack에 insert
info_stack에는 "func2 SFP“로 정보 저장

FP=SP;
function prologue 순서에 의해 SFP 저장 후 FP=SP update

push_value(200, "var_2");
push_value를 이용해 지역변수 200을 call_stack에 insert
info_stack에는 “var_2”로 정보 저장

print_stack();
print_stack을 이용해 stack 정보 출력

func3(77);
func3(77) call

pop(); //var_2
pop을 이용해 지역변수 삭제

SP=FP;
function epilogue 순서에 의해 지역변수 삭제 후 SP=FP update

int before_FP=pop(); //SFP
FP=before_FP;
pop을 이용해 before_FP에 SFP 저장과 동시에 삭제
FP=before_FP update

pop(); //Return Address
pop을 이용해 Return Address 삭제

pop(); //arg1
pop(); //arg2
pop을 이용해 매개변수 arg1, arg2 삭제

print_stack();
print_stack을 이용해 stack 정보 출력

func3

```
void func3(int arg1) {
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func3 SFP");
    FP=SP;
    push_value(300, "var_3");
    push_value(400, "var_4");
    print_stack();

    pop(); //var_4
    pop(); //var_3
    SP=FP;
    int before_FP=pop(); //SFP
    FP=before_FP;
    pop(); //Return Address
    pop(); //arg1
    print_stack();
}
```

push_value(arg1, "arg1");
push_value를 이용해 매개변수 arg1를 call_stack에 insert
info_stack에는 “arg1”로 정보 저장

push_RA();
push_RA를 이용해 -1을 call_stack에 insert
info_stack에는 "Return Address"로 정보 저장

push_SFP("func3 SFP");
push_SFP를 이용해 FP를 call_stack에 insert
info_stack에는 "func3 SFP”로 정보 저장

FP=SP;
function prologue 순서에 의해 SFP 저장 후 FP=SP update

push_value(300, "var_3");
push_value(400, "var_4");
push_value를 이용해 지역변수 300, 400을 call_stack에 insert
info_stack에는 “var_3”, “var_4”로 정보 저장

print_stack();

print_stack을 이용해 stack 정보 출력

```
pop(); //var_4
pop(); //var_3
```

pop을 이용해 지역변수 삭제

```
SP=FP;
```

function epilogue 순서에 의해 지역변수 삭제 후 SP=FP update

```
int before_FP=pop(); //SFP
FP=before_FP;
```

pop을 이용해 before_FP에 SFP 저장과 동시에 삭제
FP=before_FP update

```
pop(); //Return Address
```

pop을 이용해 Return Address 삭제

```
pop(); //arg1
```

pop을 이용해 매개변수 arg1 삭제

```
print_stack();
```

print_stack을 이용해 stack 정보 출력

output

```
====== Current Call Stack ======
5 : var_1 = 100     <=== [esp]
4 : func1 SFP     <=== [ebp]
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 ======
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.
Stack is empty.
```

추가 기능 구현

1. Stack 초기값 변경

```
#define INF 987654321
SP=-INF;
FP=-INF;
```

이와 같이 #define INF 987654321을 사용하여 변수를 초기화하면, -1과 달리 -INF와 같은 극단적인 값을 통해 해당 변수가 정상적인 스택 인덱스가 아닌 초기값임을 쉽게 식별할 수 있다.

또한 이후 Underflow 방지 기능을 구현할 때 자세히 설명하겠지만 프로그램 실행 중 문제가 발생했을 경우 987654321이라는 숫자는 모든 자릿수가 다르기에 출력된 값이 잘못된 경우 어디에서 오류가 발생했는지를 보다 빠르게 식별할 수 있다. 예를 들어, 999999999와 같은 숫자는 반복되는 패턴으로 인해 디버깅 시 가독성이 떨어질 수 있지만, 987654321은 특정한 형태를 가지고 있어 디버깅 과정에서 오류 발생 지점을 명확하게 파악하는 데 도움이 된다.

2. Overflow 방지 + Stack 초기값 처리

STACK_SIZE가 50을 넘어 Overflow가 일어날 가능성 존재

```
void push_value(int value, const char* info) {
    if(SP+1>=STACK_SIZE) {
        cout<<"Stack Overflow : "<<info<<"₩n";
        return;
    }

    if(SP==-INF)
        SP=0;

        else
        SP++;

    call_stack[SP]=value;
    strcpy(info_stack[SP], info);
}

void push_SFP(const char* info) {
    if(SP+1>=STACK_SIZE) {
        cout<<"Stack Overflow : "<<info<<"₩n";
        return;
    }

    if(SP==-INF)
        SP=0;

    else
        SP++;

    call_stack[SP]=FP;
    strcpy(info_stack[SP], info);
}

void push_RA() {
    if(SP+1>=STACK_SIZE) {
        cout<<"Stack Overflow"<<"₩n";
        return;
    }
```

```
    if(SP==-INF)
        SP=0;

    else
        SP++;

    call_stack[SP]=-1;
    strcpy(info_stack[SP], "Return Address");
}
```

```
if(SP==-INF)
    SP=0;

else
    SP++;
```

SP를 -INF로 초기화했으므로 이 경우 예외처리를 새로이 추가했다.

```
if(SP>STACK_SIZE) {
    cout<<"Stack Overflow'"<<info<<"'₩n";
    return;
}
```

이와 같이 SP+1>STACK_SIZE일 경우 Error를 전달하고 return하는 if문을 구현하여 Overflow를 방지할 수 있다. 동시에 매개변수로 전달된 info를 “Stack Overflow : ”후에 출력하여 무엇을 push할 때 Overflow가 발생했는지 확인할 수 있다. Return Address를 push하는 경우는 info를 전달하지 않기에 "Stack Overflow"만을 출력한다.

3. Underflow 방지 + Stack 초기값 처리

SP가 -1일 때, 즉 아무것도 stack에 없을 때 pop();을 실행시켜 Underflow가 일어날 가능성 존재

```
int pop() {
    if(SP==-INF) {
        cout<<"Stack Underflow";
        return 0;
    }

    int val=call_stack[SP];

    if(SP==-1)
        SP=-INF;

    else
        SP--;

    return val;
}
```

```
if(SP==-1)
    SP=-INF;

else
    SP--;
```

SP를 -INF로 초기화했으므로 이 경우 예외처리를 새로이 추가했다.

```
if(SP==-INF) {
    cout<<"Stack Underflow";
    return 0;
}
```

SP는 초기에 -INF로 초기화되어 있으므로 스택이 한 번도 사용된 적이 없는 상태에서 pop 연산이 실행될 경우 즉시 "Stack Underflow"를 출력하고 연산을 중단하여 Underflow를 방지할 수 있다.

만약 모종의 이유로 Underflow가 일어난다고 해도 초기 SP를 -INF, 즉 -987654321과 같이 모든 자릿수가 다른 수를 사용해 어디서 오류가 발생했는지 확인하기 용이하다. 이는 BaekJoon을 계속 풀어오면서 사용하던 알고리즘의 기본 Error 탐지 방법을 활용하였다.

4. Stack Size 확인

```
bool isEmpty() {
    return (SP==-INF);
}
```

현재 코드에서 SP가 -INF면 Stack이 비어있다는 의미로 판단할 수 있다. 이에 isEmpty()를 통해 SP==-INF라는 bool값을 return하여 Stack이 비어있는지 아닌지 확인할 수 있다.

```
bool isFull() {
    return (SP+1==STACK_SIZE);
}
```

현재 코드에서 SP가 49, 즉 STACK_SIZE보다 하나 적다면 끝까지 차 있다는 의미로 판단할 수 있다. 이에 isFull()을 통해 SP+1==STACK_SIZE라는 bool값을 return하여 Stack이 가득 차있는지 아닌지 확인할 수 있다.

```
int size() {
    return SP+1;
}
```

또는, 위의 isFull() 함수에서 언급했듯이, SP는 하나 채워질 때 0으로 초기화된다. 따라서 SP+1이 현재 Stack에 가지고 있는 함수들의 수이며 SP+1을 그대로 return하여 Stack의 size를 확인할 수 있다.

5. top()

```
int top() {
    if(isEmpty()) {
        cout<<"Stack is empty"<<"₩n";
        return 0;
    }

    return call_stack[SP];
}
```

C++의 Stack에서는 일반적으로 top()이라는 함수가 존재한다. pop()은 삭제와 함께 Stack의 가장 위에 있는 값을 확인하는 반면 top()은 삭제하지 않은 채로 Stack의 가장 위에 있는 값을 확인하는 함수이다. 과제 코드의 func1, fun2, fun3를 보면 int before_FP=pop(); FP=before_FP;와 같이 구현한 부분이 있다. SFP를 삭제하면서 before_FP라는 변수를 SFP값으로 초기화하여 이를 FP에 할당하고 있는데, pop()을 통해 삭제하지 않으면서 할당만 하는 경우 또한 필요하다 생각하였다. 이에 isEmpty()를 사용해 Stack이 비어 있을 경우 Error를 출력하고 그렇지 않을 경우 맨 위에 있는 값만 return하는 top()함수를 추가하였다.

6. clear()

```
void clear() {
    SP=-INF;
    FP=-INF;

    for(int i=0; i<STACK_SIZE; i++) {
        call_stack[i]=0;
        strcpy(info_stack[i], "");
    }
}
```

일반적으로 C++의 <stack> library에서는 clear()와 같은 함수는 없지만 <vector>와 같은 배열에서 clear()함수가 전체 초기화를 위해 제공된다. 과제 코드와 같이 직접 Stack을 구현하는 경우 Stack 내부 상태를 초기 상태로 되돌려야 하는 상황이 종종 발생할 것이라 생각해 이와 같은 함수를 구현하게 되었다. SP와 FP를 -INF와 같이 초기 상태로 되돌리고 STACK_SIZE까지 index에 대해 call_stack과 info_stack을 각각 0, ""로 초기화하였다.

추가 기능 구현 코드

```
#include <iostream>
#include <cstring>
using namespace std;
#define INF 987654231
#define STACK_SIZE 50

int call_stack[STACK_SIZE];
char info_stack[STACK_SIZE][20];
int SP=-INF;
int FP=-INF;

bool isEmpty() {
	return (SP==-INF);
}

bool isFull() {
	return (SP+1==STACK_SIZE);
}

int size() {
	return SP+1;
}

int top() {
	if(isEmpty()) {
		cout<<"Stack is empty"<<"₩n";
		return 0;
	}

	return call_stack[SP];
}

void clear() {
	SP=-INF;
	FP=-INF;

	for(int i=0; i<STACK_SIZE; i++) {
		call_stack[i]=0;
		strcpy(info_stack[i], "");
	}
```

```
}

void push_value(int value, const char* info) {
    if(SP+1>=STACK_SIZE) {
        cout<<"Stack Overflow : "<<info<<"₩n";
        return;
    }

    if(SP==-INF)
        SP=0;

        else
        SP++;

    call_stack[SP]=value;
    strcpy(info_stack[SP], info);
}

void push_SFP(const char* info) {
    if(SP+1>=STACK_SIZE) {
        cout<<"Stack Overflow : "<<info<<"₩n";
        return;
    }

    if(SP==-INF)
        SP=0;

    else
        SP++;

    call_stack[SP]=FP;
    strcpy(info_stack[SP], info);
}

void push_RA() {
    if(SP+1>=STACK_SIZE) {
        cout<<"Stack Overflow"<<"₩n";
        return;
    }

    if(SP==-INF)
```

```
            SP=0;

        else
            SP++;

        call_stack[SP]=-1;
        strcpy(info_stack[SP], "Return Address");
}

int pop() {
        if(SP==-INF) {
            cout<<"Stack Underflow";
            return 0;
        }

        int val=call_stack[SP];

        if(SP==-1)
            SP=-INF;

        else
            SP--;

        return val;
}

void print_stack() {
        if(SP==-1) {
            cout<<"Stack is empty."<<"₩n";
            return;
        }

        cout<<"====== Current Call Stack ======"<<"₩n";

        for(int i=SP; i>=0; i--) {
            if(call_stack[i]!=-1)
                cout<<i<<" : "<<info_stack[i]<<" = "<<call_stack[i];

            else
                cout<<i<<" : "<<info_stack[i];
```

```
        if(i==SP)
                        cout<<"     <=== [esp]";

        else if(i==FP)
                        cout<<"     <=== [ebp]";

        cout<<"₩n";
    }

    cout<<"================================"<<"₩n₩n";
}

void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);

void func1(int arg1, int arg2, int arg3) {
    push_value(arg3, "arg3");
    push_value(arg2, "arg2");
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func1 SFP");
    FP=SP;
    push_value(100, "var_1");
    print_stack();

    func2(11, 13);
    pop();
    SP=FP;
    int before_FP=pop();
    FP=before_FP;
    pop();
    pop();
    pop();
    pop();
    print_stack();
}

void func2(int arg1, int arg2) {
    push_value(arg2, "arg2");
    push_value(arg1, "arg1");
```

```
    push_RA();
    push_SFP("func2 SFP");
    FP=SP;
    push_value(200, "var_2");
    print_stack();

    func3(77);
    pop();
    SP=FP;
    int before_FP=pop();
    FP=before_FP;
    pop();
    pop();
    pop();
    print_stack();
}

void func3(int arg1) {
    push_value(arg1, "arg1");
    push_RA();
    push_SFP("func3 SFP");
    FP=SP;
    push_value(300, "var_3");
    push_value(400, "var_4");
    print_stack();

    pop();
    pop();
    SP=FP;
    int before_FP=pop();
    FP=before_FP;
    pop();
    pop();
    print_stack();
}

int main() {
    func1(1, 2, 3);
    print_stack();
    return 0;
}
```