**데이터 구조론3차과제**

<스택설계-원소 삽입부터 최소값 반환까지 상수시간에 해결하기>

15010973 정예지

**프로그램 소스코드**

#include<stdio.h> // 기본함수를이용하기위해포함

#include<stdlib.h> // exit함수를이용하기위해포함

#include<malloc.h> // 동적메모리할당malloc함수를이용하기위해포함

#include<time.h> // 난수의씨앗을변화시키는 time()함수를이용하기위해포함

#include<Windows.h> // 시간측정함수사용

#defineN 1000010 // 스택의크기상수로선언

#pragmawarning(disable:4996)

int push(intpush\_num); // 해당원소를스택에넣기위한함수리턴은그원소값

int pop(); // 해당원소를스택에서제거하기위한함수리턴은그원소값

intfindMin(); // 스택내부최소값을리턴하는함수

voidquit\_Stack(); // 프로그램종료함수

intnIndex = 0; // 메인스택내부의인덱스이자 top이자스택개수내지는크기(nIndex+1)

intnMinIndex = 0; // 최소값스택의인덱스이자 top이자개수내지는크기

int \*findMinStack; // 최소값스택에대한배열을만들기위해포인터선언>동적할당으로공간생성

int \*mainStack; // 메인스택에대한배열을만들기위해포인터선언>동적할당으로공간생성

int main(){

inti; // 반복제어변수

intminofStack; // 스택의최소값을저장하는변수

intePush; // push함수로하여금리턴되는값을저장하는변수

intePop; // pop함수로하여금리턴되는값을저장하는변수

LARGE\_INTEGERtStartTime, tEndTime, Frequency; // 알고리즘구동시간측정을위한변수

doublelfGap; // 시간측정을위해사용되는변수

char command; // push, pop, findMin, quit의명령어를받는변수

intpushNum; // push할수를받는변수

// 모두사용자에게받는다

srand(time(NULL)); // 다양성을위해srand함수로난수의씨앗을매번바꾼다

mainStack = (malloc)(sizeof(int)\*N); // N개만큼메인스택을할당

findMinStack = (malloc)(sizeof(int)\*N); // N개만큼최소값스택을할당

printf("명령을입력하세요>> ");

scanf("%c", &command);

switch (command){

case'p': case'P':

printf("<PUSH>수를입력하세요>> ");

scanf("%d", &pushNum); // 실행와숫자를사용자로부터받는다

QueryPerformanceFrequency(&Frequency);

QueryPerformanceCounter(&tStartTime);

ePush = push(pushNum); // push함수를호출하여리턴받고

QueryPerformanceCounter(&tEndTime); // 시간측정

lfGap = (double)(tEndTime.QuadPart - tStartTime.QuadPart) / Frequency.QuadPart;

printf("%c %d............출력 : push = %d (%d), cpu time = %lf\n", command, pushNum, ePush, nIndex + 1, lfGap);

break;

case'o': case'O': // pop의경우그저 top의것을없애면되므로사용자로부터숫자를받진않는다.

QueryPerformanceFrequency(&Frequency);

QueryPerformanceCounter(&tStartTime);

ePop = pop(); // pop함수를호출하고리턴받아ePop에저장

QueryPerformanceCounter(&tEndTime); // 함수실행시간측정

lfGap = (double)(tEndTime.QuadPart - tStartTime.QuadPart) / Frequency.QuadPart;

printf("%c............출력 : pop = %d (%d), cpu time : %lf\n", command, ePop, nIndex + 1, lfGap);

break;

case'f': case'F': // findMin의경우도MinStack의 top을단순읽어오기만하면되므로사용자로부터숫자를받진않음

QueryPerformanceFrequency(&Frequency);

QueryPerformanceCounter(&tStartTime);

minofStack = findMin(); // findMin함수를호출하고mainStack의최소값을리턴받아저장

QueryPerformanceCounter(&tEndTime); // 시간측정

lfGap = (double)(tEndTime.QuadPart - tStartTime.QuadPart) / Frequency.QuadPart;

printf("%c ............출력 : min = %d (%d), cpu time = %lf\n", command, minofStack, nIndex + 1, lfGap);

break;

case'q': case'Q': // 사용자가종료하기를원할때커맨드 Q값을받아프로그램을종료한다

printf("종료합니다.\n");

quit\_Stack();

break;

default: // 사용자가다른숫자또는커맨드를눌렀을때종료시키는부분

printf("Wrong command!\n");

exit(1);

break;

} // switch-case(1) :처음입력

for (i = 1; i<= 1000000; i++){

nIndex++;

push(rand() % (9999 - 1000 + 1) + 1000);

}//for

// 1000000번난수생성기를이용하여메인스택을채운다. 각실행마다인덱스값과실행횟수를 1씩증가시킨다. 자동적으로이뤄짐

fflush(stdin); // 버퍼클리어(커맨드, 문자값을받으므로)

// 1000001번인덱스이후로입력받는부분

while (TRUE){ // 커맨드값으로 Q를입력받을때프로그램을종료할것이므로무한루프를가동

fflush(stdin);

printf("명령어를입력하세요>> ");

scanf("%c", &command);

switch (command){

case'p': case'P':

printf("<PUSH>수를입력하세요>> ");

scanf("%d", &pushNum);

QueryPerformanceFrequency(&Frequency);

QueryPerformanceCounter(&tStartTime);

nIndex++; // push함수에 index증가하는부분이없고 push함수가호출될때마다인덱스를증가시키는방식으로씀.

// 첫번째 switch문과다른것은첫번째는바로 0번인덱스에넣어줄용도이므로인덱스값을증가시키지않았으나이번엔증가시킴

ePush = push(pushNum);

QueryPerformanceCounter(&tEndTime);

lfGap = (double)(tEndTime.QuadPart - tStartTime.QuadPart) / Frequency.QuadPart;

printf("%c %d............출력 : push = %d (%d), cpu time = %lf\n", command, pushNum, ePush, nIndex + 1, lfGap);

// 함수호출과실행시간측정

break;

case'o': case'O':

QueryPerformanceFrequency(&Frequency);

QueryPerformanceCounter(&tStartTime);

ePop = pop();

QueryPerformanceCounter(&tEndTime);

lfGap = (double)(tEndTime.QuadPart - tStartTime.QuadPart) / Frequency.QuadPart;

printf("%c............출력 : pop = %d (%d), cpu time : %lf\n", command, ePop, nIndex + 1, lfGap);

break;

case'f': case'F':

QueryPerformanceFrequency(&Frequency);

QueryPerformanceCounter(&tStartTime);

minofStack = findMin();

QueryPerformanceCounter(&tEndTime);

lfGap = (double)(tEndTime.QuadPart - tStartTime.QuadPart) / Frequency.QuadPart;

printf("%c ............출력 : min = %d (%d), cpu time = %lf\n", command, minofStack, nIndex+1, lfGap);

break;

case'q': case'Q':

printf("종료합니다.\n");

quit\_Stack();

break;

default:

printf("Wrong command!\n");

exit(1);

break;

}

}

return 0;

}

int push(intpush\_num){

if (nIndex == N - 1) { // top이배열의크기보다 1뺀만큼이면

printf("FULL!\n");

exit(1); // 예외처리수행

}

mainStack[nIndex] = push\_num; // 넣으려는인덱스값에넣고자하는원소대입

if (nIndex == 0){ // 0번째원소이면최솟값판별스택에그냥들어가짐.

findMinStack[0] = push\_num; // 최소스택에넣고

nMinIndex = 0; // 최소스택의현인덱스이자 top을 0으로만듦

}

elseif (nIndex != 0 &&mainStack[nIndex] <= findMinStack[nMinIndex]){ // 만약에인덱스값이 0이아니고 (이미한개씩은각스텍이들어가있음) 최소스택보다작거나같으면

if (nMinIndex == N - 1){ // 이때에도최소스택에예외상황이있을수있으니처리해주고

printf("FULL!\n");

exit(1);

}

nMinIndex++; // 인덱스하나증가시킨뒤에

findMinStack[nMinIndex] = mainStack[nIndex]; // 현재최소스택의top에있는애보다작은메인스택의것을 top에대입한다

}

/\*

최소스택의 top엔항상집어넣을때기준으로최신의최소값이들어있음. 집어넣을때마다계속비교를하기때문에.

\*/

returnpush\_num; // 방금집어넣은원소대입. 반드시대입이된후에리턴이되게끔설계되어있으므로메인함수에서받은것을그대로리턴해줌

}

int pop(){

intpop\_return;

if (nIndex == -1) {

printf("Empty!\n");

exit(1); // pop해줄때마다인덱스값이자 top값이 1씩빠지게된다. 스택의마지노선인 0에서다시호출하면 -1이되므로이는예외처리신호로사용됨

}

pop\_return = mainStack[nIndex];

if (mainStack[nIndex] == findMinStack[nMinIndex]){ // 지우고자하는메인스택의값이최소스택의값과같아질때가있다. 이때는둘다삭제해주어야함.

/\*만약 0번째에 119가있고 200000번째이자 top에 119가있고이는최소값이라고가정하자. 우리는 119를지우고싶다. 하지만그러려면

최소스택의 top에있는 119와 main스택의 top에있는 119가지워져야한다. 그럼메인스택의 0번째에도 119가있는데, 문제가되는거아닌가? 라고반문할수도있다.

하지만우리는앞에서 push를해줄때작거나같은것도다넣었다. top의 119가 200000번째메인스택의원소로인한것이라면바로밑의 119가있고

이는 0번째원소로기인한것이므로상관없다.\*/

mainStack[nIndex] = findMinStack[nMinIndex] = 0; // 해당인덱스의공간은초기화하고

nIndex -= 1;

nMinIndex -= 1; // 각각인덱스값은빼준다

}

else{

mainStack[nIndex] = 0; // 최소값과같지않다면그냥초기화하고

nIndex -= 1; // 메인의인덱스만줄여주면된다

}

returnpop\_return; // pop\_num을리턴한다.

}

intfindMin(){

returnfindMinStack[nMinIndex]; // 최소값은그냥스택맨윗부분에있는것을리턴하기만하면된다.

}

voidquit\_Stack(){

exit(1); // 종료시키는함수

}

**프로그램 실행 화면**

