프로그래밍 언어론

Report #5

2017.04.25

20154073

강다현

5장 연습문제

#6. 다음 JavaScript 골격 프로그램을 생각해보자.

//main 프로그램

**var** x;

function sub1( ) {

**var** x;

function sub2( ) {

**. . .**

}

}

function sub3( ) {

**. . .**

}

위의 프로그램 실행이 다음의 순서로 진행된다고 가정하라.

main이 sub1을 호출한다

sub1이 sub2를 호출한다

sub2가 sub3을 호출한다.

1. 정적 영역을 가정하고서, 다음에서, x에 대한 참조에 대해서 x의 어느 선언이 올바른가?  
   1. sub1 🡪 sub1의 x 참조 :: 전역변수가 hidden됨  
   2. sub2 🡪 sub1의 x 참조 :: 정적 부모 영역의 x 변수 참조  
   3. sub3 🡪 전역변수 x 참조 :: 정적 부모 영역의 x 변수 참조
2. 동적 영역을 가정하고서, a의 문제를 다시 해결하라.  
   1. sub1 🡪 sub1의 x 참조 :: sub1 내부에 선언된 변수 참조  
   2. sub2 🡪 sub1의 x 참조 :: sub2를 호출한 sub1에 선언된 변수 참조  
   3. sub3 🡪 sub1의 x 참조 :: sub3를 호출한 sub2, sub2를 호출한 sub1의 x 참조

#10. 다음의 C 프로그램을 생각해보자.  
**void** fun ( **void** ) {

**int**  a, b, c ; /\* 정의 1 \*/

**. . .**

**while** ( **. . .** ) {

**int** b, c, d ; /\* 정의 2 \*/

**. . . 🡨---------------------------------- 1**

**while** ( **. . .** ) {

**int** c, d, e ; /\* 정의 3 \*/

**. . . 🡨------------------------------ 2**

}

**. . . 🡨----------------------------------- 3**

}

**. . . 🡨--------------------------------------- 4**

}

위 함수에서 화살표로 지시된 4개 지점 각각에 대해, 각 가시적인 변수와 이 변수를 정의하는 정의문의 번호를 함께 나열하라.

1. 정의 1의 a, 정의 2의 b, c, d :: 정의 1의 b와 c는 정의 2에 의해 히든

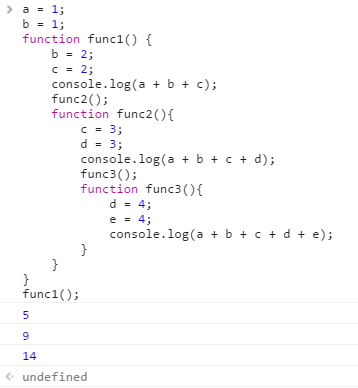
2. 정의 1의 a, 정의 2의 b, 정의 3의 c, d, e :: 정의 2의 c와 d는 정의 3에 의해 히든

3. 정의 1의 a, 정의 2의 b, c, d :: 정의 3의 c, d, e는 참조 영역을 벗어나 정의 2에서   
히든 된 변수들이 참조됨

4. 정의 1의 a, b, c :: 정의 2의 b, c, d는 참조 영역을 벗어나 정의 1에 히든 된 변수들이 참조 가능

5장 프로그래밍 연습문제

# 3. 부프로그램이 3 수준의 깊이만큼 중첩되어 있고, 각 중첩된 부프로그램이 자신을 포괄하는 모든 부프로그램에서 정의되어 있는 변수들을 참조하는 JavaScript 스크립트를 작성하라.



javascript는 동적 변수이므로 호출된 함수는 자신을 호출한 함수의 변수들을 참조할 수 있다. 이 때 각각의 함수에서 참조될 수 있는 변수들은 다음과 같다.

우선 func1( )에서는 외부에 정의된 a와 b를 참조할 수 있지만 변수 b는 func1( ) 내부에도 정의되어 있으므로 외부의 b는 히든되어 전역변수 a와 내부에 정의된 b와 c를 참조할 수 있다.

func2( )에서는 func1( )에서 참조 가능한 변수들을 참조할 수 있고, 내부에 선언된 변수들을 참조할 수 있는데 func1( )에서 참조할 수 있는 변수 c가 func2( )에도 선언되어 있으므로, func1( )에서 참조 가능한 전역변수 a, func1( )의 변수 b, func2( )의 변수 c와 d를 참조할 수 있다.

func3( )에서는 func2( )에서 참조 가능한 모든 변수들을 참조할 수 있고, func3( )내부에 func2( )에서 참조 가능한 변수명과 동일한 변수가 있다면 func2( )의 변수가 히든된다. 그러므로 func3( )에서 참조 가능한 변수는 전역변수 a, func1( )의 변수 b, func2( )의 변수 c, func3( )의 변수 d, e가 있다.

위를 바탕으로 출력 값을 비교하면, func1에서는 1 + 2 + 2의 결과 5가 출력되고, func1( )에서 호출된 func2( )의 결과값은 1 + 2 + 3 + 3으로 9가 된다. 마지막으로 func2( )에서 호출된 func3( )의 결과값은 1 + 2 + 3 + 4 + 4로 14가 출력된다

# 7. C나 C++로 다음 세 개의 함수를 작성하라 : 규모가 큰 배열을 정적으로 선언하는 함수, 동일한 크기의 배열을 스택상에 선언하는 함수, 그리고 동일한 크기의 배열을 힙상에 생성하는 함수. 각 함수를 상당히 많은 횟수(적어도 100,000번)로 호출하고, 각 함수에서 요구되는 시간을 출력하라. 그 결과를 설명하라.

<코드>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define SIZE 100000

int \*p;

void staticDeclaration() { //정적 선언

static int a[SIZE];

}

void stackDeclaration() { //스택상에 선언

int a[SIZE];

}

void heapDeclaration() { //힙상에 선언

p = (int\*)malloc(sizeof(int)\*SIZE);

}

int main() {

float gap = 0;

time\_t startTime = 0, endTime = 0;

////////////////////////정적 선언////////////////////////

startTime = clock();

for (int i = 0; i < SIZE; i++) {

staticDeclaration();

}

endTime = clock();

gap = (float)(endTime - startTime) / (CLOCKS\_PER\_SEC);

printf("정적 선언 측정 시간 : %f 초\n", gap);

////////////////////////스택 선언////////////////////////

startTime = clock();

for (int i = 0; i < SIZE; i++) {

stackDeclaration();

}

endTime = clock();

gap = (float)(endTime - startTime) / (CLOCKS\_PER\_SEC);

printf("스택 선언 측정 시간 : %f 초\n", gap);

////////////////////////힙 선언////////////////////////

startTime = clock();

for (int i = 0; i < SIZE; i++) {

heapDeclaration();

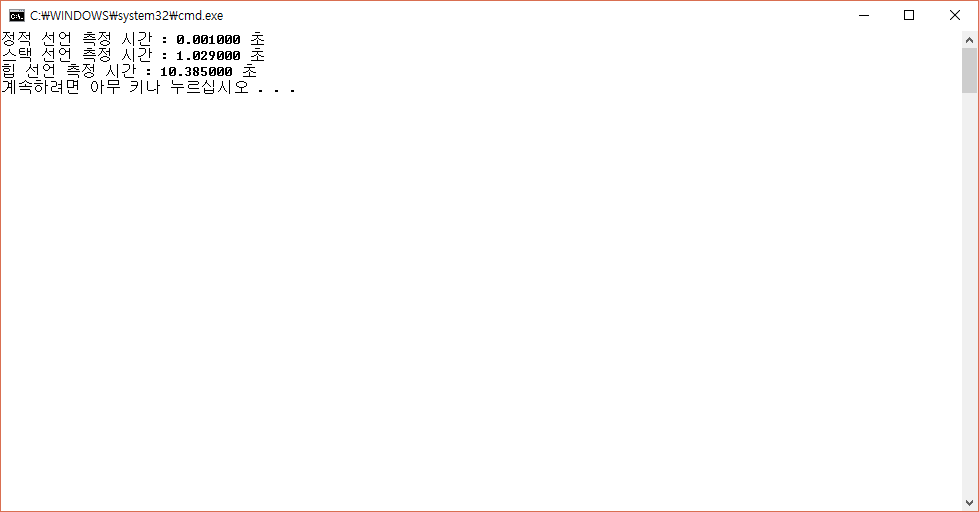
}

endTime = clock();

gap = (float)(endTime - startTime) / (CLOCKS\_PER\_SEC);

printf("힙 선언 측정 시간 : %f 초\n", gap);

}



정적 선언된 배열은 컴파일 시기에 데이터 영역에 바인딩 되어 함수 호출 시 따로 다시 메모리 할당하는 오버헤드가 없어 매우 빠르게 접근 가능하다.

스택 선언된 배열은 부프로그램이 세련화될 때 스택 영역에 바인딩 된다. 부프로그램이 실행완료 되었을 때 해제되며, 호출될 때 마다 반복하여 메모리에서 할당 해제하는 오버헤드가 발생하기 때문에 정적 선언된 배열을 참조하는 것 보다 느리다.

힙 선언된 배열은 명령어가 실행될 때부터, 해제 명령이 있을 때까지 존재하지만, 현재 해제 명령이 존재하지 않기 때문에 프로그램 종료 시까지 존재한다. 또한 할당 작업을 하기 위해 많은 시간이 소요되는데 이러한 속도 저하는 블록이 사용 가능한 목록에 없으므로 런타임 할당자 코드에서 사용 가능한 더 큰 블록을 찾거나 백 엔드로부터 새 블록을 할당하는데 클럭이 소요되기 때문이다.