|  |  |
| --- | --- |
| 이름 | 신동욱 |
| 학번 | 20204101 |
| 학년 | 3 |
| 과목 | 프로그래밍언어론 |



프로그래밍언어론  
보고서#5

1. 연습문제 6, 9, 11
2. 정적 영역, 동적 영역의 차이를 보여주는 Perl 프로그램
3. 다음 각 언어에 대해 더 큰 영역의 이름을 재사용 할 수 있는지를 판단하는 코드, C, C++, Java
4. C나C++로 다음 3개의 함수를 작성하라

1. 규모가 큰 배열을 정적으로 선언하는 함수

2. 동일한 크기의 배열을 스택상에 선언하는 함수

3. 동일한 크기의 배열을 힢 상에 생성하는 함수

각 함수를 상당히 많은 회수로 호출하고 시간을 출력하라

1. 연습문제 6.

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 화이트, 영수증이(가) 표시된 사진

자동 생성된 설명

텍스트, 영수증, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

1. sub1 : sub1 에서 선언된 x
2. sub2 : sub1 에서 선언된 x
3. sub3 : 전역변수 x



1. sub1 : sub1 에서 선언된 x
2. sub2 : sub1 에서 선언된 x
3. sub3 : sub1 에서 선언된 x

텍스트, 스크린샷, 문서, 폰트이(가) 표시된 사진

자동 생성된 설명연습문제 9.

1. sub1 : x(전역, 참조만 가능) , a(sub1), y(sub1), z(sub1)
2. sub2 : x(전역, 할당도 가능), a(sub2), w(sub2)
3. sub3 : a(sub2), b(sub3), z(sub3)

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명

1. a(main), b(f1), c(f2), d(f3), e(f3), f(f3)
2. a(main), b(f1), c(f1), d(f3), e(f3), f(f3)
3. a(main), b(f1), c(f1), d(f1), e(f3), f(f3)
4. a(main), b(f1), c(f1), d(f1), e(f3), f(f3)
5. a(main), b(f1), c(f2), d(f2), e(f2), f(f3)
6. a(main), b(f1), c(f1), d(f1), e(f2), f(f3)

**2. 정적 영역, 동적 영역의 차이를 보여주는 Perl 프로그램**

1. 정적 영역 – perl

sub sub2 {

my ($b, $c) = ("sub2\_b", "sub2\_c");

print "sub2 - (a:$a) (b:$b) (c:$c) (d:$d)\n";

sub sub1 {

my ($a, $b) = ("sub1\_a", "sub1\_b");

print "sub1 - (a:$a) (b:$b) (c:$c) (d:$d) \n";

}

sub1();

}

my ($c, $d) = ("main\_c", "main\_d");

print "Main - (a:$a) (b:$b) (c:$c) (d:$d) \n" ;

sub2();

my 키워드를 사용하여 정적 영역을 구현할 수 있다.

각 함수별로 a b c d 의 값을 출력하도록 했다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명 실행 결과

1. main 블록(함수 밖)에서는 정의된 c, d를 출력한다. a와 b가 정의되어 있지 않기에 출력되지 않는다.
2. sub2 에서는 sub2 내에서 선언된 b, c를 출력한다. 비 지역 변수도 존재하지 않기 때문에 a와 d는 출력되지 않는다.
3. sub1 에서는 sub1 내에서 선언된 a, b를 출력한다. 정적 부모(sub1)에서 선언된 c가 참조되어 sub1의 c가 출력된다.

2. 동적 영역 – perl

sub sub2 {

local ($b, $c) = ("sub2\_b", "sub2\_c");

print "sub2 - (a:$a) (b:$b) (c:$c) (d:$d)\n";

sub sub1 {

local ($a, $b) = ("sub1\_a", "sub1\_b");

print "sub1 - (a:$a) (b:$b) (c:$c) (d:$d) \n";

}

sub1();

}

local ($c, $d) = ("main\_c", "main\_d");

print "Main - (a:$a) (b:$b) (c:$c) (d:$d) \n" ;

sub2();

local 키워드를 사용하여 동적 영역을 구현할 수 있다.

각 함수별로 a b c d 의 값을 출력하도록 했다.

실행 결과

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. main 블록(함수 밖)에서는 정의된 c, d를 출력한다. a와 b가 정의되어 있지 않기에 출력되지 않는다.
2. sub2 에서는 sub2 내에서 선언된 b, c를 출력하고, 정적 영역이기에 호출자의 변수들도 참조된다. 호출된 영역(main – 함수 밖)의 d가 출력된다.
3. sub1 에서는 sub1 내에서 선언된 a, b를 출력하고, 정적 영역이기에 호출자의 변수들도 참조된다. sub1을 호출한 sub2의 c가 참조된다. sub2를 호출한 main – 함수 밖의 d가 출력된다

**3. C, C++, Java에 대해서 더 큰 영역의 이름을 재사용 할 수 있는지 판단하는 코드**

1. C

int a = 1;

{

int a = 2;

printf("%d\n", a);

}

영역 밖에서 선언한 a를 재사용 할 수 있다.

2. C++

int a = 1;

{

int a = 2;

cout << a << endl;

}

영역 밖에서 선언한 a를 재사용 할 수 있다.

3. Java

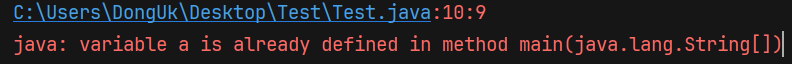
int a = 1;

{

int a = 2;

System.out.println(a);

}



Java는 영역 밖에서 선언한 a에 대해 영역 내부에서 정의하면 컴파일 오류가   
 발생한다

**4. 규모가 큰 배열 - 정적, 스택, 힢 상에 선언하는 함수를 여러 번 호출하여 시간 측정하기**

#define SIZE 100000

void staticArr() {

static int staticArr[SIZE];

}

void stackArr() {

int stackArr[SIZE];

}

int\* heapArr() {

int\* heapArr = new int[SIZE];

return heapArr;

}

int main() {

for (int i = 0; i < 100000; i++) {

staticArr();

}

for (int i = 0; i < 100000; i++) {

stackArr();

}

for (int i = 0; i < 100000; i++) {

heapArr();

}

}

(코드와 보고서의 가독성을 위해 시간 측정 부분은 생략)

실행 결과

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

C++ 환경에서 작성되었다.

배열의 사이즈는 상수로 100000을 부여했다.

정적 배열을 만드는 함수 staticArr() 내부에서는 배열을 static 키워드로 배열을 생성하였다.

배열을 스택 상에 선언하는 함수 stackArr() 내부에서는 배열의 크기를 미리 지정하여 스택 상에 배열이 만들어지도록 했다.

배열을 힢 상에 선언하는 함수 heapArr() 내부에서는 배열을 new 키워드로 생성하여 힢 상에 배열이 만들어지도록 했다.

메인 함수에서는 각 함수를 100000씩 반복을 돌려 함수를 호출하였다.

정적 배열은 함수가 종료되어도 만들어진 배열은 사라지지 않고 남아있다. 따라서 함수를 여러 번 호출하여도 기존 배열이 그대로 남아 있기에 매우 짧은 시간이 걸린 것을 볼 수 있다.

스택 배열은 0.95초로, 함수가 종료되면 만들어진 배열은 사라지기에, 함수가 호출 될 때마다 스택 영역에 배열이 생성된다. 따라서 정적으로 선언한 경우보다 느린 것을 볼 수 있다.

힢 배열은 35.9초로, 마찬가지로 함수가 종료되면 만들어진 배열은 사라진다. 함수가 호출될 때 마다 힢 영역에 배열이 생성된다. 힢 영역에 배열을 생성하는 것은 스택 영역에 만드는 것 보다 (힢 구조 특성상) 상당히 느린 구조를 가진다. 따라서 매우 오랜 시간이 걸리는 것을 볼 수 있다.