**Homework 4**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.03.29.월 |

1. **중복되지 않는 big random number array 생성 및 샘플 출력**

* RAND\_MAX를 초과하는 범위의 중복되지 않는 난수를 발생시킬 수 있는 함수 genBigRandArray(int \*array, int size)를 작성하라

void genBigRandArray(int\* array, int size) {

char\* flag; // 난수가 이미 지정되었는지 확인

unsigned int uint\_32, bigRand;

int count = 0; // size 만큼 실행

srand(time(0));

flag = (int\*)malloc(sizeof(char) \* size); // 동적 할당으로 큰 크기의 배열 생성

while (count < size) {

uint\_32 = ((long)rand() << 15) | rand(); // 15bit 좌로 이동하여 난수 크기를 확장

bigRand = uint\_32 % size;

if (flag[bigRand] == 1) continue;

else {

flag[bigRand] = 1;

array[count++] = bigRand;

}

}

free(flag); // 할당 해제

}

* 100개 이상의 원소를 가지는 큰 정수 배열에서 첫 부분과 마지막 부분의 샘플을 출력하는 함수 printArraySample(int \*array, int size, int line\_size)를 작성하라

void printArraySample(int\* array, int size, int line\_size) {

for (int i = 0, count = 0; i < 3; i++) { // 앞부분 30개 출력

for (int j = 0; j < line\_size; j++) {

printf("%8d", array[count++]);

}

printf("\n");

}

printf(" . . . .\n\n");

for (int i = 0, count = size-31; i < 3; i++) { // 뒷부분 30개 출력

for (int j = 0; j < line\_size; j++) {

printf("%8d", array[count++]);

}

printf("\n");

}

printf("\n");

}

* 위에서 작성한 두 함수를 사용하여 size가 200,000 ~ 1,000,000인 큰 난수배열을 생성하고, 생성된 큰 난수 배열의 샘플을 출력하는 기능을 시험하는 main() 함수를 구현하고, 그 결과를 출력하라

int main(void) {

int\* array; // pointer for dynamin array

for (int size = 200000; size <= 1000000; size += 200000) { // size 를 늘려가며 난수 생성

printf("Testing generation of dynamic array of random numbers ( size : %d )\n", size);

array = (int\*)calloc(size, sizeof(int)); // dynamic array

if (array == NULL) {

printf("Error in dynamic memory allocation for integer array of size (%d)\n", size);

exit(-1); // 바로 프로세스 종료

}

genBigRandArray(array, size); // 확장된 난수 생성

printArraySample(array, size, LINE\_SIZE); // 생성된 난수를 출력

// LINE\_SIZE = 10;

free(array);

}

return 0;

}



1. **power(base, exponent) 함수**

* power(base, exponent) 를 재귀 (recursive) 구조와 반복 (iterative) 구조로 각각 작성하라

double powerIter(double base, int exponent) {

double buf = 1.0;

for (int i = 0; i < exponent; i++) buf \*= base; // 반복문으로 승수만큼 곱한다

return buf;

}

double powerRecur(double base, int exponent) {

if (exponent == 0) return 1.0; // 마지막 승수에서 함수 호출 종료

else return base \* powerRecur(base, exponent - 1); // 재귀함수로 승수를 구현

}

* main() 함수에서는 이 두 함수를 차례로 호출하며 base, exponent 에 인수를 전달하고, exponent 값이 1000, 2000, 3000, 4000 인 경우에 대하여 각각 실행하라

int main(void) {

time\_t t\_before, t\_after;

int t\_diff; // 걸린 sec 의 차이

LONGLONG t\_diff\_pc;

LARGE\_INTEGER freq, t1, t2;

double t\_elapse\_us;

double result\_i, result\_r;

double base = 1.015;

QueryPerformanceFrequency(&freq); // 주파수 값 기록

for (int expo = 1000; expo <= 4000; expo += 1000) {

time(&t\_before); // 함수 호출 이전 시간

QueryPerformanceCounter(&t1); // 이전 performance counter 값을 기록

result\_i = powerIter(base, expo); // 계산 값 도출

QueryPerformanceCounter(&t2);// 이후 performance counter 값을 기록

time(&t\_after); // 함수 호출 후 시간

t\_diff = difftime(t\_after, t\_before); // 경과한 시간

t\_diff\_pc = t2.QuadPart - t1.QuadPart;

t\_elapse\_us = ((double)t\_diff\_pc / (double)freq.QuadPart) \* 1000000; // micro 초 계산

printf("PowerItera(1.015, %d) by iterative = %40.5lf, took(%3d) sec, (%10.2lf) micro-second\n", expo, result\_i, t\_diff, t\_elapse\_us);

time(&t\_before);

QueryPerformanceCounter(&t1);

result\_r = powerRecur(base, expo);

QueryPerformanceCounter(&t2);

time(&t\_after);

t\_diff = difftime(t\_after, t\_before);

t\_diff\_pc = t2.QuadPart - t1.QuadPart;

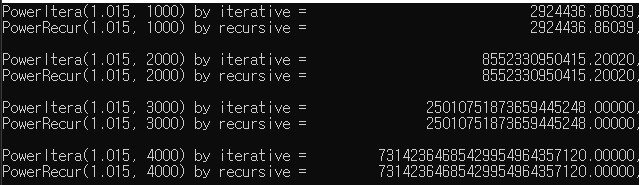
t\_elapse\_us = ((double)t\_diff\_pc / (double)freq.QuadPart) \* 1000000;

printf("PowerRecur(1.015, %d) by recursive = %40.5lf, took(%3d) sec, (%10.2lf) micro-second\n\n", expo, result\_r, t\_diff, t\_elapse\_us);

}

return 0;

}



* 각 함수 실행에서 걸린 시간을 마이크로 초 단위로 측정하고 함수 구현 방법에 따라 시간에 차이가 나는 이유를 설명하라  
  (time(), QueryPerformanceCounter() 함수를 이용)

재귀함수의 프로그램 실행 속도가 반복문으로 만든 프로그램보다 느린 이유는 함수를 호출하고 종료할 때 스택 프레임을 구성하고 해제하는 과정에서 반복문보다 오버헤드가 많이 발생하기 때문에 속도가 느려진다.

스택 프레임 : 함수를 호출할 때 할당되는 정보

오버헤드 : 처리 과정에서 발생하는 간접적인 처리시간과 메모리, ……

재귀함수를 사용해야 한다면, 동적 프로그래밍을 통해 동일한 인수를 사용한 재귀함수 호출이 여러 번 일어나지 않도록 개선한다.

