#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

int \*A;

int candleArrayVer1(int n, int k); //첫번째 방식 : 배열로 구하는 함수

int candleArrayVer2(int n, int k); //두번째 방식 : 배열로 구하는 함수

void buildListArray(int A[], int n); //배열 빌드

int runSimulationVer1(int A[], int n, int k); //첫번째 방식으로 실행

int runSimulationVer2(int A[], int n, int k); //두번째 방식으로 실행

int removeAtRank(int rank, int n, int A[]); //삭제함수

LARGE\_INTEGER startTimeLinked, endTimeLinked; //연결리스트 함수 시간구하기 위한 스타트와 엔드타임, simulation함수를 잰다

LARGE\_INTEGER startTimeArray1, endTimeArray1; //배열함수 첫번째 방식 시간구하기 위한 스타트함수와 엔드타임, simulation 함수를 잰다

LARGE\_INTEGER startTimeArray2, endTimeArray2; //배열함수 두번째 방식 시간구하기 위한 스타트와 엔드타임, simulation함수를 잰다.

LARGE\_INTEGER Frequency; //ms초로 시간을 구하기 위한

typedef struct node //연결리스트

{

int elem;

struct node\* next;

}NODE;

NODE \*L; //연결리스트 L

NODE\* getnode(); //노드함수

int candleLinked(int n, int k); //연결리스트로 구하는 함수

NODE\* buildListLinked(int n); //연결리스트 빌드

int runSimulationLinked(NODE\* L, int n, int k); // 연결리스트로 실행

int removeLinked(NODE\* L);

int candleArrayVer1(int n, int k) {

int \*A;

A = (int\*)malloc(sizeof(int)\*n); //포인터 A 동적할당

buildListArray(A, n); //build함수 실행

return runSimulationVer1(A, n, k);

}

int candleArrayVer2(int n, int k) {

int \*A;

A = (int\*)malloc(sizeof(int)\*n); //포인터 A 동적할당

buildListArray(A, n);

return runSimulationVer2(A, n, k);

}

void buildListArray(int A[], int n) {

//build함수의 시작시간을 잰다.

int r;

for (r = 0; r < n; r++) {

A[r] = r + 1;

}

//build함수의 끝나는 시간을 잰다.

}

int removeAtRank(int r, int n, int A[]) {

int i;

int data = 0;

data = A[r]; //data값은 A[r]

for (int i = r + 1; i < n; i++) //r+!부터 n-1까지 반복

A[i - 1] = A[i];

n -= 1; //n은 1씩 감소

return n;

}

int runSimulationVer1(int A[], int n, int k) {

QueryPerformanceFrequency(&Frequency); //frequency 설정

QueryPerformanceCounter(&startTimeArray1); //첫번째 배열 방식 스타트타임 재기

int rank = 0, i, N;

N = n;

rank = 0;

while (n >= 1) { //while문 n이 1보다 크거다 같을 때

i = 0;

while (i < k) { //i가 k보다 작을동안

rank = (rank + 1) % N; //rank는 rank+1을 배열의 크기로 나눈 나머지값

if (A[rank] != 0) { //A[rank]가 0이 아니라면

i += 1; //i는 1씩 증가

}

}

if (n > 1) { //n이 1보다 클경우

A[rank] = 0; //A[rank]는 0

do

rank = (rank + 1) % N; //rank는 rank+1을 배열의 크기로 나눈 나머지값

while (A[rank] == 0); //A[rank]가 0일동안

}

n -= 1;

}

QueryPerformanceCounter(&endTimeArray1); //첫번째 배열방식 엔드 타임 재기

return A[rank];

}

int runSimulationVer2(int A[], int n, int k) {

QueryPerformanceCounter(&startTimeArray2); //두번째 배열방식 스타트타임 재기

int r = 0;

r = 0;

while (n > 1) { //n이 1보다 클동안

r = (r + k) % n; //r은 r+k를 배열의 크기로 나눈 나머지값

n = removeAtRank(r, n, A);

}

QueryPerformanceCounter(&endTimeArray2); //두번째 배열방식 엔드타임 재기

return A[0];

}

NODE \*getnode() { //node 만들기

NODE\* node;

node = (NODE\*)malloc(sizeof(NODE));

node->elem = NULL;

node->next = NULL;

return node;

}

int removeLinked(NODE\* L) { //Node 삭제

NODE\* temp;

int nTemp;

temp = L; //연결리스트 temp와 L같다.

nTemp = L->elem; //ntemp값은 L의elem과 같음.

while (temp->next != L) {

temp = temp->next; //temp 옆으로 이동

}

temp->next = L->next; //이동한 것이 L의 이동과 동일

free(L); //L을 놔주자

return nTemp;

}

int candleLinked(int n, int k) {

NODE\* L = buildListLinked(n);

return runSimulationLinked(L, n, k);

}

NODE\* buildListLinked(int n) { //build함수

int i;

NODE\* p;

NODE\* L;

p = getnode();

L = p;

p->elem = 1;

for (i = 2; i <= n; i++) {

p->next = getnode(); //p->next는 getnode()로

p = p->next; //p는 이동

p->elem = i; //p의 구성요소에 i

}

p->next = L;

return L;

}

int runSimulationLinked(NODE\* L, int n, int k) {

QueryPerformanceCounter(&startTimeLinked); //연결리스트 함수 스타트타임 재기

int i;

NODE\* p;

p = L;

while (p != p->next) { //p가 p의 다음노드가 아닐때

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

p = p->next; //p의 이동

}

removeLinked(p->next);

p = p->next; //p의 이동

}

QueryPerformanceCounter(&endTimeLinked); //연결리스트 함수 엔드타임 재기

return p->elem;

}

void main() {

int n[] = { 31, 33, 10000 };

int k[] = { 5, 59, 100 };

int i;

QueryPerformanceFrequency(&Frequency); //frequency 설정

for (i = 0; i < 2; i++) {

printf("배열버전1: (n = %d, k = %d) 생존 위치 = %d\n", n[i], k[i], candleArrayVer1(n[i], k[i]));

printf("배열버전2: (n = %d, k = %d) 생존 위치 = %d\n", n[i], k[i], candleArrayVer2(n[i], k[i]));

printf("연결버전: (n = %d, k = %d) 생존 위치 = %d\n", n[i], k[i], candleLinked(n[i], k[i]));

printf("\n");

}

if (i = 2) {

printf("배열버전1: (n = %d, k = %d) 생존 위치 = %d 실행시간 = %lf\n", n[2], k[2], candleArrayVer1(n[2], k[2]), (double)(endTimeArray1.QuadPart - startTimeArray1.QuadPart) / Frequency.QuadPart);

printf("배열버전2: (n = %d, k = %d) 생존 위치 = %d 실행시간 = %lf\n", n[2], k[2], candleArrayVer2(n[2], k[2]), (double)(endTimeArray2.QuadPart - startTimeArray2.QuadPart) / Frequency.QuadPart);

printf("연결버전: (n = %d, k = %d) 생존 위치 = %d 실행시간 = %lf\n", n[2], k[2], candleLinked(n[2], k[2]), (double)(endTimeLinked.QuadPart - startTimeLinked.QuadPart) / Frequency.QuadPart);

}

return 0;

}

