**파일처리론 02분반  
한정규 교수님**

**<Replacement Selection Report>**

2142851 김형준

**동아대학교 컴퓨터공학과**

**목차**

1. 컴파일 방법
2. 프로그램 설계
3. 실행 결과 스크린샷
4. 어려웠던 점과 해결방법
5. 시도한 코드들
6. **컴파일 방법**

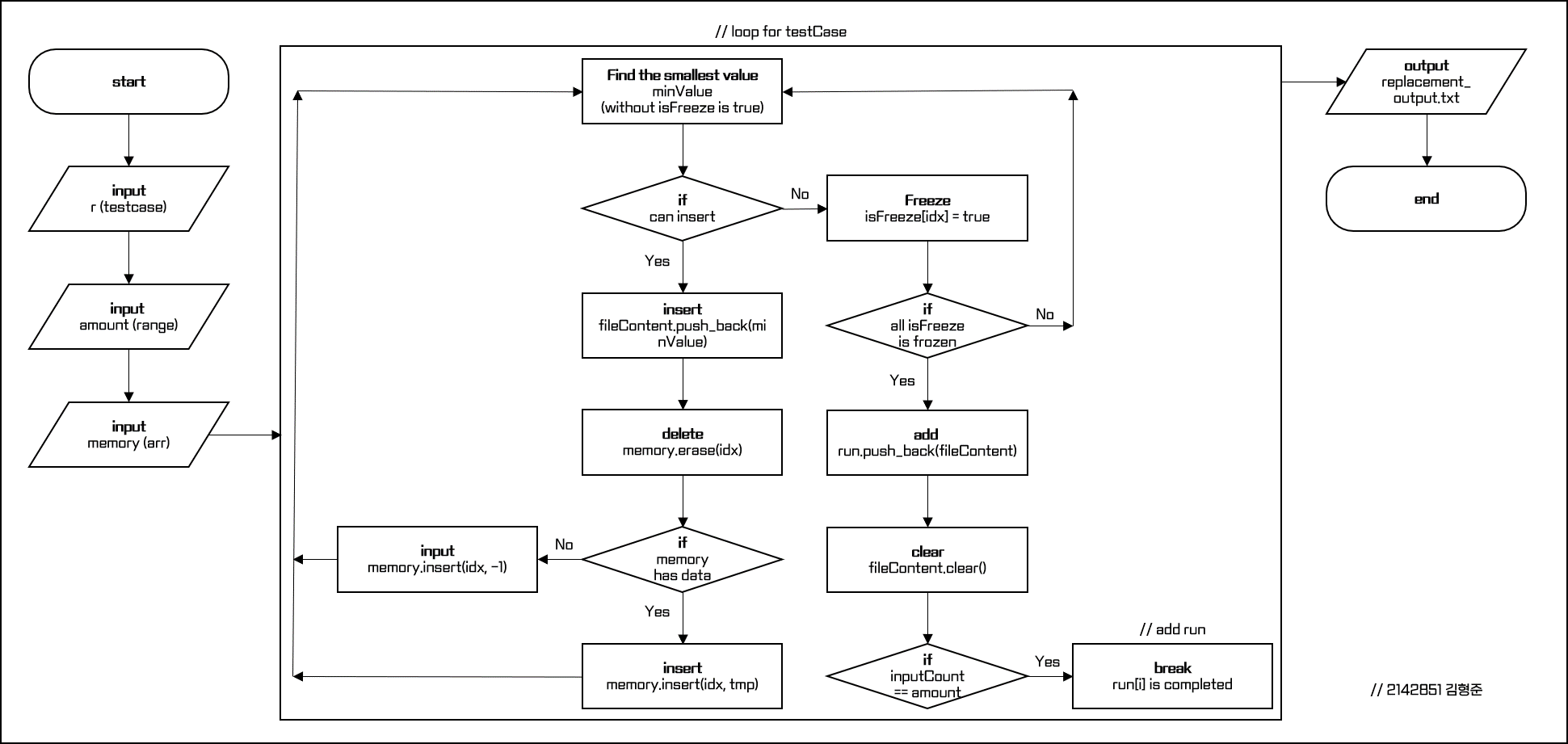
* **Visual Studio 2019를 사용하여 컴파일하였습니다. (컴파일러: Visual C++)**

1. 프로젝트 생성 후 소스파일을 디버깅 시 x64/Debug 폴더 안에 replace.exe 파일이 생성됩니다.
2. 이 Debug 폴더 안에 replacement\_input.txt와 replacement\_output.txt, replacement\_selection\_tester.exe를 넣고 replacement.exe를 실행 후 tester를 실행시키면 PASS가 뜹니다.

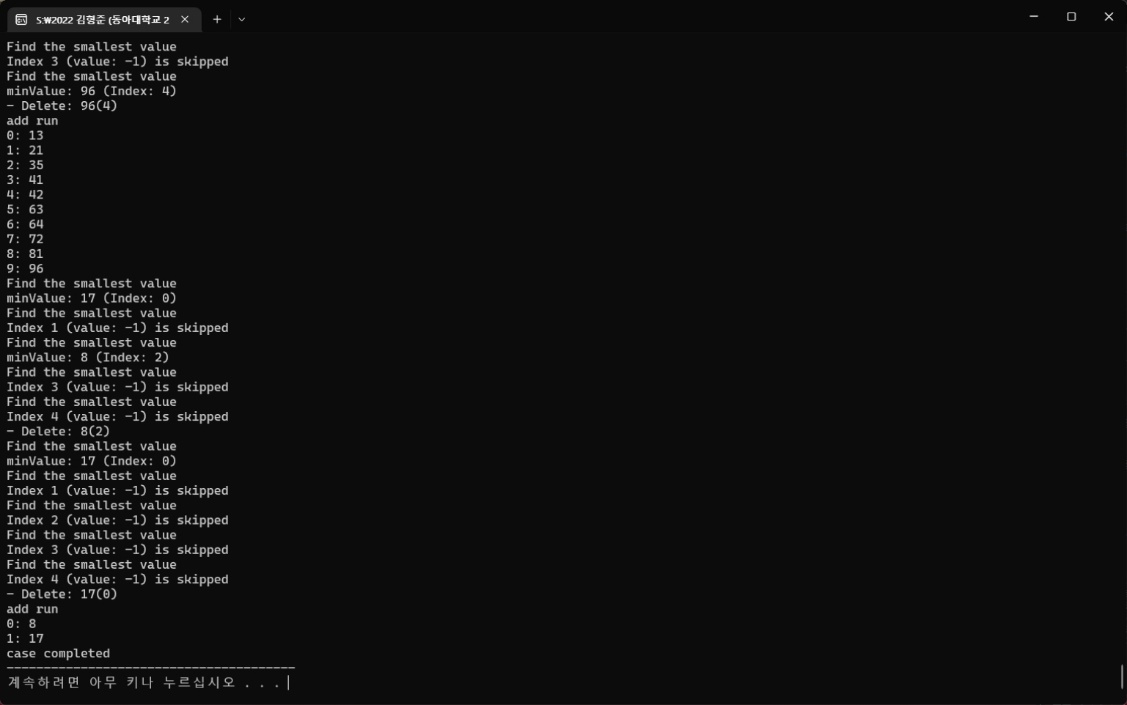
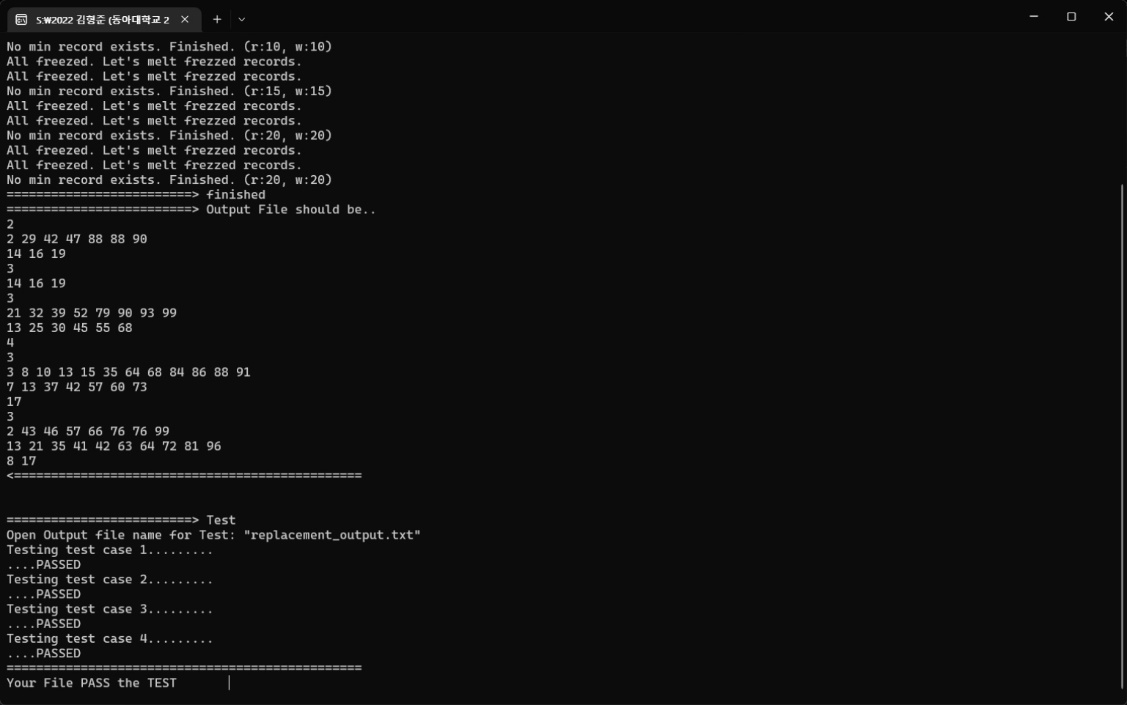
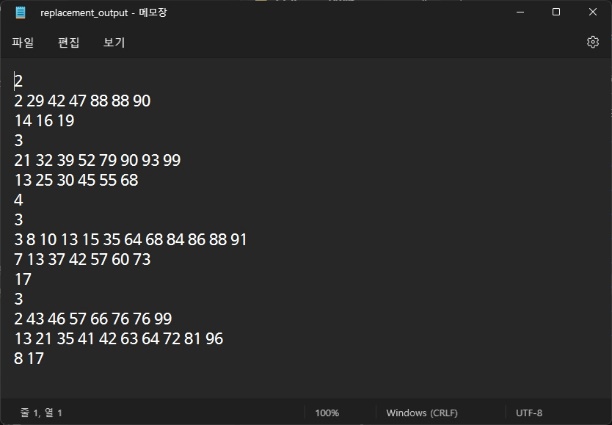
* **안 될 경우 Code Blocks로 해도 됩니다. (컴파일러: GCC)**(<https://www.codeblocks.org/>)

1. Code Blocks 설치 방법과 컴파일러 설정 방법은 별도 파일을 첨부하겠습니다.
2. replacement.cpp 파일을 Code Blocks를 통해 열고 디버깅 시 cpp 파일이 있는 경로에 replacement.exe 파일이 생성됩니다.
3. replacement.exe 파일이 생성된 폴더 안에 위와 같이 replacement\_input.txt와 replacement\_output.txt, replacement\_selection\_tester.exe를 넣고 replacement.exe를 실행 후 tester를 실행시키면 PASS가 뜹니다.
4. **프로그램 설계**

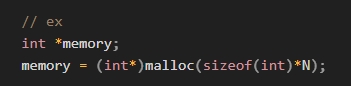
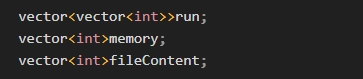
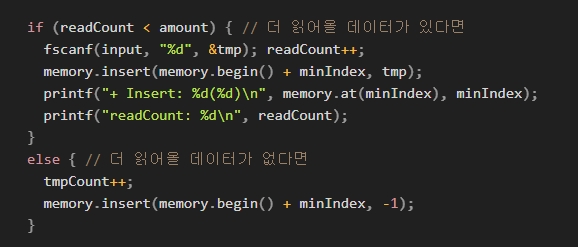
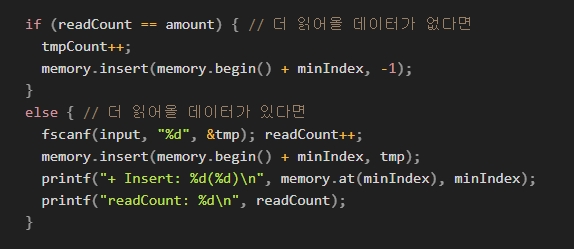
* **프로그램 작성에 앞서 먼저 설계도를 작성하였습니다.**
  + 변수명
  + 간단한 알고리즘
  + 프로그램의 실행 순서
  + 사용할 vector의 함수 정리



1. **실행 결과 스크린샷**

* replacement.exe (console)  
  
* replacement\_selection\_tester.exe  
  
* replacement\_output.txt  
  

1. **어려웠던 점과 해결방법**

* **어려웠던 점 1) 동적 배열의 한계**  
  처음에는 run, memory, fileContent 변수를 vector를 사용하지 않고 동적 할당을 사용하여 코딩하였습니다.   
  ****  
  하지만 이렇게 코드를 작성 시 run에 값을 저장하기 어렵고 복잡해진다는 것을 알게 되었습니다.
  + 이를 해결하기 위해 동적할당 대신 vector를 사용하기로 하였습니다.  
      
    그 결과 memory에 fileContent의 대체선택 값을 저장하고, vector<int> 타입의 vector인 run에 넣어주니 동적 할당보다 더 쉽고 간편하게 코드를 작성할 수 있었습니다.
* **어려웠던 점 2) 출력 및 파일 READ의 문제**  
  두번째 시도 코드에서 아래와 같이 작성하였습니다.  
    
  그 결과 replacement\_output.txt의 값이 매우 이상하게 나왔습니다.  
  2 2 29 42 47 88 88 90 14 16 19 5 13 30 32 45 55 68 79 99 4 10 13 15 20 25 64 68 88 91 7 13 42 60 73 76 99 2 21 41 43 46 57 63 64 81 96 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 17 1 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 1 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
  여러 방법을 시도해보았지만 결과는 여전하였습니다. 이 문제점의 원인을 생각해보니 뭔가 fscanf가 더 읽히는 느낌이 들었습니다. 따라서 fscanf가 실행되는 부분에 console log를 찍어서 확인해보니 fscanf가 실행되어야 할 조건에 만족되지 않아도 실행되는 중이었습니다.  
    
  그래서 아래와 같이 코드를 변경하였습니다. (if, else 문의 순서 변경)  
    
  이후 제대로 작동됨을 확인할 수 있었습니다.
* **어려웠던 점 3) 동결**동결이 발생하는 값의 처리가 어려웠습니다.
  + 동결이 발생하기 위한 조건인 memory의 마지막 값이 옮기려는 값보다 클 경우를 if문을 사용해 골라내었습니다. 그 후 동결되는 값은 freeze 되었는지 확인하기 위한 bool 타입의 isFreeze 변수를 만들고, 그 값이 위치하는 index의 isFreeze를 true로 설정함과 동시에 freezeCount를 1 증가시켰습니다. 그 후 최소값을 찾을 때 isFreeze가 true라면, 그 값은 최소값 찾기에서 제외(continue)시켰습니다.
* **어려웠던 점 4) 빈 메모리와 동결의 처리**  
  코딩을 하다보니 대체 선택 중 피할 수 없는 빈 메모리 생성과 동결에 대해 어려움이 있었습니다. 처음에는 동결된 메모리의 개수가 5일 경우 run을 생성하기로 했습니다. 하지만 이렇게 작성하니 빈 메모리가 생길 경우 run이 아예 생성되지 않는 무한 루프에 빠졌습니다.
  + 이 문제점은 간단한 수식으로 해결하였습니다. 먼저 동결된 메모리가 생길 경우 freezeCount라는 변수를 1 증가시키고(어려웠던 점 3과 동일), 더 읽어올 데이터가 없는 상황에서 데이터가 넘어갔을 경우 빈 메모리가 생기기 때문에 tmpCount라는 변수를 1 증가시키면서 빈 값이 존재할 index의 memory를 -1로 설정하였습니다. 이후 아래의 식을 이용하여 run 생성의 조건을 추가하였습니다.  
    freezeCount + tmpCount == 5
* **추가로) 어려운 점은 대부분 vector의 함수 사용에 있었습니다.**
  + 구글링을 통해 찾아보고 직접 예제를 적용해보며 vector 함수의 이해를 높이니 어려운 부분을 차근차근 해결해 나갈 수 있었습니다.
* **순서도) 위에서 언급했듯이, 순서도를 사용했습니다.**
  + 코딩 중 알고리즘이 생각이 안 날 때, 또는 논리 오류가 발생하였을 때 미리 작성했던 순서도를 보고, 변수의 값을 코드 한 줄 씩 내려가며 생각해보았더니, 문제를 해결할 수 있었습니다.

1. **시도한 코드들**

* 총 3번의 코드 시도를 저장해놨습니다.
* 실행을 위한 코드는 과제 제출 페이지에 제출한 replacement.cpp를 사용해주세요.
* **첫번째 시도 코드**  
  #pragma warning (disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <vector>

#define nl(x) for(int i = 0; i < x; i++) printf("\n");

#define BUFFER\_SIZE 5

using namespace std;

int main() {

vector<vector<int>>run; // run

vector<int>memory;

vector<int>fileContent;

int r; // Test Case

int amount;

int tmp;

FILE\* input = NULL;

FILE\* output = NULL;

input = fopen("replacement\_input.txt", "r"); // input file open with "read"

output = fopen("replacement\_output.txt", "w"); // output file open with "write"

if (input == NULL) { // open fail

printf("FILE(input.txt) Reading Fail.\n");

return 1;

}

if (output == NULL) { // open fail

printf("FILE(output.txt) Writting Fail.\n");

return 1;

}

printf("FILE Reading and Writting are ready.\n");

fscanf(input, "%d", &r); // input testCase

printf("---------------------------------------\n");

printf("testCase: %d\n", r); // check testCase

printf("---------------------------------------\n");

for (int testCase = 0; testCase < r; testCase++) {

fscanf(input, "%d", &amount); // input amount

printf("amount: %d\n", amount); // check amount

//arr = (int\*)malloc(sizeof(int) \* amount); // malloc

int fileCount = 0; // file count

int minValue; // minimum value

int minIndex = 0; // minimum index

bool isFreeze[BUFFER\_SIZE] = { false, false, false, false, false }; // 동결된 버퍼 T/F

int freezeCount = 0;

int n = 0; // 더 읽어올 데이터가 있는지 찾기 위한 count

// 이후 Replacement Selection

int i = 0;

while (1) {

if (i == 0) { // 처음 5개 먼저 넣기

for (int j = 0; j < BUFFER\_SIZE; j++) {

printf("Input 1~5 Buffer: %d\n", j);

fscanf(input, "%d", &tmp);

memory.push\_back(tmp);

printf("Value: %d\n", tmp);

}

}

bool flag = true; // 동결되지 않은 가장 첫 Value, Index를 minValue, minIndex로 설정하기 위함

// Buffer 5 안의 가장 작은 값 찾기

for (int j = 0; j < BUFFER\_SIZE; j++) { // find minimum in memory(buffer)

printf("Find the smallest value\n");

if (isFreeze[j] || memory.at(j) == -1) { // 동결되어 있거나 빈(-1) 데이터를 가지고 있다면 건너뜀

printf("Index %d is skipped\n", j);

continue;

}

if (flag) {

flag = false;

minValue = memory.at(j);

minIndex = j;

}

if (memory.at(j) < minValue) {

minValue = memory.at(j);

minIndex = j;

}

printf("minValue: %d (Index: %d)\n", minValue, minIndex);

}

// 정상적으로 들어갈 수 있다면

if (i == 0 || (fileContent.back() <= minValue)) {

// run.at(n).push\_back(minValue); // save data

fileContent.push\_back(minValue);

memory.erase(memory.begin() + minIndex);

n++;

if (n < (amount - BUFFER\_SIZE \* 2 + 1)) { // 더 읽어올 데이터가 있다면

fscanf(input, "%d", &tmp);

memory.insert(memory.begin() + minIndex, tmp);

}

else { // 더 읽어올 데이터가 없다면

memory.insert(memory.begin() + minIndex, -1);

}

}

// 동결

else {

isFreeze[minIndex] = true;

printf("%d Index is frozen\n", minIndex);

freezeCount++;

printf("freezeCount: %d\n", freezeCount);

}

i++;

// 모두 동결된 상태라면

if (freezeCount == 4) {

printf("add run\n");

run.push\_back(fileContent);

fileContent.clear(); // 저장된 fileContent의 모든 요소 삭제

freezeCount = 0;

i = 0;

for (int cnt = 0; cnt < 5; cnt++) isFreeze[cnt] = false; // isFreeze 전부 false로 초기화

}

if (n == amount - BUFFER\_SIZE) {

printf("case completed\n");

break;

}

//fscanf(input, "%d", &memory[minIndex]);

}

// Text File에 출력하기

fprintf(output, "%d\n", run.size());

for (int i = 0; i < run.size(); i++) {

for (int j = 0; j < run.at(i).size(); j++) {

fprintf(output, "%d ", run.at(i).at(j));

}

fprintf(output, "\n");

}

printf("---------------------------------------\n");

fprintf(output, "---------------------------------------\n");

}

fclose(input);

fclose(output);

system("pause");

return 0;

}

* **두번째 시도 코드**

#pragma warning (disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <vector>

#define nl(x) for(int i = 0; i < x; i++) printf("\n");

#define BUFFER\_SIZE 5

using namespace std;

int main() {

//COORD bufferSize = { 100, 10000 }; // 가로, 세로

//SetConsoleScreenBufferSize(GetStdHandle(STD\_OUTPUT\_HANDLE), bufferSize);

vector<vector<int>>run; // run

vector<int>memory;

vector<int>fileContent;

int r; // Test Case

int amount;

int tmp;

FILE\* input = NULL;

FILE\* output = NULL;

input = fopen("replacement\_input.txt", "r"); // input file open with "read"

output = fopen("replacement\_output.txt", "w"); // output file open with "write"

if (input == NULL) { // open fail

printf("FILE(input.txt) Reading Fail.\n");

return 1;

}

if (output == NULL) { // open fail

printf("FILE(output.txt) Writting Fail.\n");

return 1;

}

printf("FILE Reading and Writting are ready.\n");

fscanf(input, "%d", &r); // input testCase

printf("---------------------------------------\n");

printf("testCase: %d\n", r); // check testCase

printf("---------------------------------------\n");

for (int testCase = 0; testCase < r; testCase++) {

fscanf(input, "%d", &amount); // input amount

printf("amount: %d\n", amount); // check amount

//arr = (int\*)malloc(sizeof(int) \* amount); // malloc

int fileCount = 0; // file count

int minValue; // minimum value

int minIndex = 0; // minimum index

bool isFreeze[BUFFER\_SIZE] = { false, false, false, false, false }; // 동결된 버퍼 T/F

int freezeCount = 0;

int inputCount = 0; // 더 읽어올 데이터가 있는지 찾기 위한 count

int tmpCount = 0;

int readCount = 0;

// 이후 Replacement Selection

int i = 0;

while (1) {

if (i == 0) { // 처음 5개 먼저 넣기

for (int j = 0; j < BUFFER\_SIZE; j++) {

fscanf(input, "%d", &tmp); readCount++;

memory.push\_back(tmp);

printf("Value(%d): %d\n", j, memory.at(j));

}

}

bool flag = true; // 동결되지 않은 가장 첫 Value, Index를 minValue, minIndex로 설정하기 위함

// Buffer 5 안의 가장 작은 값 찾기

for (int j = 0; j < BUFFER\_SIZE; j++) { // find minimum in memory(buffer)

printf("Find the smallest value\n");

if (isFreeze[j] || memory.at(j) == -1) { // 동결되어 있거나 빈 데이터(-1)를 가지고 있다면 건너뜀

printf("Index %d (value: %d) is skipped\n", j, memory.at(j));

continue;

}

if (flag) { // min에 동결되어 있거나 빈 데이터(-1)가 아닌 가장 첫 값 대입

flag = false;

minValue = memory.at(j);

minIndex = j;

}

if (memory.at(j) < minValue) {

minValue = memory.at(j);

minIndex = j;

}

printf("minValue: %d (Index: %d)\n", minValue, minIndex);

}

// 정상적으로 들어갈 수 있다면

if (i == 0 || (fileContent.back() <= minValue)) {

// run.at(n).push\_back(minValue); // save data

fileContent.push\_back(minValue);

printf("- Delete: %d(%d)\n", memory.at(minIndex), minIndex);

memory.erase(memory.begin() + minIndex);

inputCount++;

if (readCount < amount) { // 더 읽어올 데이터가 있다면

fscanf(input, "%d", &tmp); readCount++;

memory.insert(memory.begin() + minIndex, tmp);

printf("+ Insert: %d(%d)\n", memory.at(minIndex), minIndex);

printf("readCount: %d\n", readCount);

}

else { // 더 읽어올 데이터가 없다면

tmpCount++;

memory.insert(memory.begin() + minIndex, -1);

}

}

// 동결

else {

isFreeze[minIndex] = true;

printf("%d Index is frozen\n", minIndex);

freezeCount++;

printf("freezeCount: %d\n", freezeCount);

}

i++;

// 모두 동결된 상태라면

if (freezeCount + tmpCount == 5) {

printf("add run\n");

run.push\_back(fileContent);

for (int cnt = 0; cnt < fileContent.size(); cnt++) {

printf("%d: %d\n", cnt, fileContent.at(cnt));

}

fileContent.clear(); // 저장된 fileContent의 모든 요소 삭제

freezeCount = 0;

i = 0;

for (int cnt = 0; cnt < 5; cnt++) isFreeze[cnt] = false; // isFreeze 전부 false로 초기화

}

if (inputCount == amount) {

printf("case completed\n");

break;

}

//fscanf(input, "%d", &memory[minIndex]);

}

// Text File에 출력하기

fprintf(output, "%d\n", run.size());

for (int i = 0; i < run.size(); i++) {

for (int j = 0; j < run.at(i).size(); j++) {

fprintf(output, "%d ", run.at(i).at(j));

}

fprintf(output, "\n");

}

// 모든 vector 초기화

run.clear();

fileContent.clear();

memory.clear();

printf("---------------------------------------\n");

//fprintf(output, "---------------------------------------\n");

}

fclose(input);

fclose(output);

system("pause");

return 0;

}

* **세번째 시도 코드**

#pragma warning (disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <vector>

#define nl(x) for(int i = 0; i < x; i++) printf("\n");

#define BUFFER\_SIZE 5

using namespace std;

int main() {

//COORD bufferSize = { 100, 10000 }; // 가로, 세로

//SetConsoleScreenBufferSize(GetStdHandle(STD\_OUTPUT\_HANDLE), bufferSize);

vector<vector<int>>run; // run

vector<int>memory;

vector<int>fileContent;

int r; // Test Case

int amount;

int tmp;

FILE\* input = NULL;

FILE\* output = NULL;

input = fopen("replacement\_input.txt", "r"); // input file open with "read"

output = fopen("replacement\_output.txt", "w"); // output file open with "write"

if (input == NULL) { // open fail

printf("FILE(input.txt) Reading Fail.\n");

return 1;

}

if (output == NULL) { // open fail

printf("FILE(output.txt) Writting Fail.\n");

return 1;

}

printf("FILE Reading and Writting are ready.\n");

fscanf(input, "%d", &r); // input testCase

printf("---------------------------------------\n");

printf("testCase: %d\n", r); // check testCase

printf("---------------------------------------\n");

for (int testCase = 0; testCase < r; testCase++) {

fscanf(input, "%d", &amount); // input amount

printf("amount: %d\n", amount); // check amount

//arr = (int\*)malloc(sizeof(int) \* amount); // malloc

int fileCount = 0; // file count

int minValue; // minimum value

int minIndex = 0; // minimum index

bool isFreeze[BUFFER\_SIZE] = { false, false, false, false, false }; // 동결된 버퍼 T/F

int freezeCount = 0;

int inputCount = 0; // 더 읽어올 데이터가 있는지 찾기 위한 count

int tmpCount = 0;

int readCount = 0;

// 이후 Replacement Selection

int i = 0;

for (int j = 0; j < BUFFER\_SIZE; j++) {

fscanf(input, "%d", &tmp); readCount++;

memory.push\_back(tmp);

printf("Value(%d): %d\n", j, memory.at(j));

}

while (1) {

bool flag = true; // 동결되지 않은 가장 첫 Value, Index를 minValue, minIndex로 설정하기 위함

// Buffer 5 안의 가장 작은 값 찾기

for (int j = 0; j < BUFFER\_SIZE; j++) { // find minimum in memory(buffer)

printf("Find the smallest value\n");

if (isFreeze[j] || memory.at(j) == -1) { // 동결되어 있거나 빈 데이터(-1)를 가지고 있다면 건너뜀

printf("Index %d (value: %d) is skipped\n", j, memory.at(j));

continue;

}

if (flag) { // min에 동결되어 있거나 빈 데이터(-1)가 아닌 가장 첫 값 대입

flag = false;

minValue = memory.at(j);

minIndex = j;

}

if (memory.at(j) < minValue) {

minValue = memory.at(j);

minIndex = j;

}

printf("minValue: %d (Index: %d)\n", minValue, minIndex);

}

// 정상적으로 들어갈 수 있다면

if (i == 0 || (fileContent.back() <= minValue)) {

// run.at(n).push\_back(minValue); // save data

fileContent.push\_back(minValue);

printf("- Delete: %d(%d)\n", memory.at(minIndex), minIndex);

memory.erase(memory.begin() + minIndex);

inputCount++;

if (readCount == amount) { // 더 읽어올 데이터가 없다면

tmpCount++;

memory.insert(memory.begin() + minIndex, -1);

}

else { // 더 읽어올 데이터가 있다면

fscanf(input, "%d", &tmp); readCount++;

memory.insert(memory.begin() + minIndex, tmp);

printf("+ Insert: %d(%d)\n", memory.at(minIndex), minIndex);

printf("readCount: %d\n", readCount);

}

}

// 동결

else {

isFreeze[minIndex] = true;

printf("%d Index is frozen\n", minIndex);

freezeCount++;

printf("freezeCount: %d\n", freezeCount);

}

i++;

// 모두 동결된 상태라면

if (freezeCount + tmpCount == 5) {

printf("add run\n");

run.push\_back(fileContent);

for (int cnt = 0; cnt < fileContent.size(); cnt++) {

printf("%d: %d\n", cnt, fileContent.at(cnt));

}

fileContent.clear(); // 저장된 fileContent의 모든 요소 삭제

freezeCount = 0;

i = 0;

for (int cnt = 0; cnt < 5; cnt++) isFreeze[cnt] = false; // isFreeze 전부 false로 초기화

}

if (inputCount == amount) {

printf("case completed\n");

break;

}

//fscanf(input, "%d", &memory[minIndex]);

}

// Text File에 출력하기

fprintf(output, "%d\n", run.size());

for (int i = 0; i < run.size(); i++) {

for (int j = 0; j < run.at(i).size(); j++) {

fprintf(output, "%d ", run.at(i).at(j));

}

fprintf(output, "\n");

}

// 모든 vector 초기화

run.clear();

fileContent.clear();

memory.clear();

printf("---------------------------------------\n");

//fprintf(output, "---------------------------------------\n");

}

fclose(input);

fclose(output);

system("pause");

return 0;

}-