Programming Methodology
Instructor: Kyuseok Shim

Project #4: external sort with template
Due Date: 0:0 a.m. between 2007-12-2 & 2007-12-3

Introduction

이 프로젝트는 C++의 template을 이용한 sorting algorithm과 정렬해야 할 데이터의 크기가 main memory의 크기를 넘어갈 때 데이터를 정렬하는 external sorting algorithm을 구현하는 것이다.

Implementation

1. in memory sorting algorithm. memory안에 들어갈 수 있는 크기의 데이터를 정렬하는 heap-sort을 구현한다. 데이터의 type은 정해져 있는 것이 아니라 template을 사용하고, 정렬은 오름차순으로 한다.
2. main memory의 크기보다 큰 사이즈의 데이터를 정렬하는 external sorting algorithm을 구현한다.
   - 입력 파일과 결과 파일은 text형태이다
   - 중간 생성 파일들은 text 형태가 아닌 binary 형태로 저장한다
   - page size는 1024 byte로 고정한다

Input & Output

input file은 숫자들의 집합이다. 첫 줄의 첫번째 숫자는 현재 파일에 들어 있는 데이터의 개수를 나타내고 그 다음은 S/I/L/F/D 중 한 문자이다 – short, int, long, float, double. 다음 줄은 빈칸이나 탭키로 구별되는 숫자들이 나열된다.

```
100 I  // 앞으로 integer 100개가 나옴
722 329 62 1958 2 2950 6291 58 942 ... 2984
```

output file은 정렬된 숫자들의 집합이다. 각 줄마다 숫자를 하나씩 출력한다.

```
2
66
245
...
```

프로그램 실행 예시

">(실행파일) (입력) (메모리사이즈-byte) (출력)"의 형식으로 실행한다

```
> external_sort.exe  input.txt  102400  output.txt
>
```

참고

1. external sort와 heap의 pseudo-code는 게시된 pdf 파일을 참고하도록 한다.

2. heap

다음 skeleton code를 참고한다. 기능만 동일하다면 아래 코드와 전혀 다르게 구현하더라도 무방하다.

※클래스 이름, 함수 이름, 함수 인자 종류 등은 임의로 변경가능

```
template<class T> class HeapEntry {
public :
	T value;		// data
	int run;		// 어느 run에서 온 data인가를 기록
};

template<class T> class Heap {
private:
	HeapEntry<T> *heap; // heap으로 사용할 공간
	Int maxsize;		// allocate 받은 heap size
	Int cursize;		// 현재 heap size, insert나 extract있을 때 +/-
	...			// 추가로 필요한 항목 추가
public :
	Heap();			// Constructor, 필요하면 인자 추가
	~Heap();			// Destructor
	int insert(const HeapEntry<T>& h);	// Heap에 data를 삽입
	HeapEntry<T>& extract();	// top node에 있는 값을 반환, 삭제
	void build_heap();		// heap을 구성하는 함수
	void heapify();		// 주어진 data로 heap을 만듬
	HeapEntry<T>& lookTop();	// heap의 젤 윗부분 반환, 삭제X
	...			// 추가로 필요한 항목 추가
};
```

3. binary 형태로 읽고 쓰기

C의 함수를 이용

#메모리의 내용을 파일을 쓰는 함수

size_t fwrite(const void* ptr, size_t size, size_t count, FILE* stream);

ptr : Pointer to the array of elements to be written

size : Size in bytes of each element to be written

count : Number of elements, each one with a size of *size* bytes

stream : Pointer to a FILE object that specifies an output stream

---

example

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
   FILE* pFile;
   int buffer[5] = {1,2,3,4,5};
   pFile = fopen("myfile.bin","wb");
   if ( pFile == NULL ) {               // 파일을 열지 못했으면 오류출력
      cerr<<"File error"<<endl;
      return 1;
   }
   fwrite(buffer,sizeof(int),5,pFile);  // buffer가 가리키는 주소부터
   fclose(pFile);                       // sizeof(int)*5 byte를
   return 0;                            // pFile에 씀
}
```

#파일의 내용을 메모리에 읽어들이는 함수

size_t fread (void *ptr, size_t size, size_t count, FILE* stream);

ptr : Pointer to a block of memory with a minimum size of (*size***count*) bytes

size : Size in bytes of each element to be read

count : Number of elements, each one with a size of *size* bytes

stream : Pointer to a FILE object that specifies an input stream

---

```
example
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
   FILE* pFile;
   int buffer[5];
   pFile = fopen("myfile.bin","rb");
   if ( pFile == NULL ) {
      cerr<<"File error"<<endl;
      return 1;
   }
   fread(buffer, sizeof(int), 5, pFile);// buffer가 가리키는 주소로부터
   fclose(pFile);                       // sizeof(int)*5 byte 읽음
   for ( int i = 0; i < 5; i++ )
       cout<<"buffer["<<i<<"]="<<buffer[i]<<endl;
   return 0;
}
```

#파일에서 원하는 위치로 포인터를 움직이는 함수

int fseek (FILE * stream, long int offset, int origin);

stream : Pointer to a FILE object that identifies the stream

offset : Number of bytes to offset from origin

origin : Position from where offset is added. It is specified by one of the following constants defined in <cstdio>

SEEK_SET - Beginning of file

SEEK_CUR - Current position of the file pointer

SEEK_END - End of file

example

```
#include <iostream>
#include <cstdio>
using namespace std;
int main () {
   FILE* pFile;
   int buf;
   pFile = fopen("myfile.bin","rb");
   if ( pFile == NULL ) {
      cerr<<"File error"<<endl;
      return 1;
   }
   fseek(pFile,-sizeof(int),SEEK_END);  // 파일의 끝에서 –sizeof(int)이동
   fread(&buf,sizeof(int),1,pFile);     // 현재 위치에서 sizeof(int) byte
   cout<<"buf : "<<buf<<endl;           // 를 buf에 카피(읽음)

   fseek(pFile,2*sizeof(int),SEEK_SET); //파일의시작부분에서 2*sizeof(int)
   fread(&buf,sizeof(int),1,pFile);     //byte 이동해서 sizeof(int) byte
   cout<<"buf : "<<buf<<endl;           //읽어서 출력
   return 0;
}
```

제출사항

1. 제출 방법 및 내용

   1) 프로젝트 구현 파일 제출

      - 파일은 ee 홈페이지에 과제 제출을 이용하여 제출한다.
      - 소스 파일과 컴파일을 할 수 있는 파일과 실행파일을 압축하여 제출한다.
      - **각 제출 파일 이름은 다음의 형식을 따른다.**

        pro_4_학번.zip

        예) pro_4_2007-12345.zip

      - 각 소스 파일 상단에 **E-mail주소와 작성자 학번 이름**을 적는다.

   2) 보고서 제출

      간단한 설명과 discussion을 하드카피로 제출한다.

2. 제출 일시

   파일 제출 마감일 : 2007년 12월 2일과 3일 사이 새벽 0:00

   - 구현 파일 제출 시각은 홈페이지 과제 제출 시각에 준한다

   보고서 제출 마감일 : 2007년 12월 3일 오전 12:00(정오)

   302동 516-2호 앞 과제제출함

채점 규칙

몇 가지 다양한 대용량 데이터들에 대해서 정렬이 정확히 되는지, 중간생성 파일이 생성이 되는지 확인하여 채점

딜레이는 하루에 10%씩 감점

Copy 발견 시 F 처리