자료구조 및 최적화 HW1

B611127 유지혁

Visual studio로 실행할 경우, 가끔씩 dll의 문제로 인해 프로그램이 제대로 동작하지 않는 일이 생겼습니다. 만약 텍스트 파일로 인한 코드가 실행되지 않는경우, B611127 폴더에 있는 .sln파일을 열어서 확인해주시면 될 것 같습니다.

1. **Main 함수의 흐름**
2. 10000개의 record를 임의로 만들어 DB를 생성한다.
3. 실행하고자 하는 명령어를 fgets함수로 입력받으면, 띄어쓰기로 파싱을 진행한다.
4. 첫번째로 파싱된 토큰을 기준으로 각 명령에 맞는 함수를 call한다.

(linked list로 구현하는 경우, head를 가리키는 포인터와 마지막을 가리키는 포인터는 따로 구현하지 않았습니다.)

1. **각 함수의 흐름(배열로 구현한 경우)**

* **Insert 함수**

1. 기존의 배열크기에서 한 구조체의 크기만큼 늘린다. (배열은 각 data들의 주소값이 연속적이어야 하므로 realloc을 이용하여 배열의 크기를 변화시킨다.)
2. 마지막 index 값인 \_msize(students) / sizeof(Student) – 1 에 입력받은 값들을 저장한다.
3. 명령어로 입력받을 땐, 정수 값도 문자열로 받았으므로 num, score의 경우 정수형으로 변환한다.

* **Delete 함수**

1. 배열의 처음 index부터 for문으로 이름을 비교한다. (strcmp함수는 비교할 문자열이 일치하면 0을 반환한다.)
2. 만약, students[i]의 이름이 찾고자 하는 이름이면 i+1 ~ 마지막 배열의 값을 한 칸씩 앞으로 가져온다.
3. 기존 배열의 크기에 하나의 record값을 뺀 크기로 realloc을 실행한다.

* **List 함수**

1. 배열의 처음 index부터 for문으로 이름을 비교한다.
2. 이름이 같은 배열의 index를 찾으면 해당 record를 printf문으로 출력한다.

* **Sort name함수(selection sorting 사용)**

1. 배열의 i번째 index값과 i+1번째부터 마지막 index까지의 값을 비교한다. I는 첫번째 index부터 시작하며, j는 i+1번째부터 마지막 index를 가리킨다.
2. i번째 index의 이름과 j번째 index의 이름을 사전순으로 비교한다.(strcmp함수를 이용). 만약 strcmp(i.name,j.name)이 양수가 나온다면, i번째 이름이 j번째 이름보다 사전순으로 뒤에 있다는 의미이므로 small이라는 index값을 j로 저장한다.
3. J가 배열의 마지막 index까지 비교한 이후, small위치에 있던 값과 I 값을 swap한다.

* **Sort score 함수(selection sorting 사용)**
* Sort name과 같은 방식으로 진행한다. 단, sorting 기준을 score라는 정수값으로 sorting한다.
* **Read DB 함수**

1. 해당 DB가 존재하는지를 알아보는 access함수를 실행한다.(없으면 error)
2. 해당 DB가 존재하면, file 포인터를 사용하여 fileinfo[100]에 한 줄씩 입력받는다.
3. 한 줄씩 입력받아 저장할 때마다, realloc을 사용하여 students의 크기를 늘린다.
4. Fileinfo[100]이 NULL을 받았다면, 더 이상 받을 data가 없다는 의미이므로 함수를 종료한다.

* **Write DB 함수**

1. file 포인터를 이용하여 마지막 index까지 입력한다.
2. i < \_msize(students) / sizeof(Student)를 이용하여 마지막 index를 구한다.
3. **각 함수의 흐름(linked list로 구현한 경우)**

* makerandStudents함수

1. malloc으로 하나의 record를 생성한다.
2. 첫번째 list에 값을 넣고 ptr로 해당 리스트를 가리킨다.
3. 두번째 이후의 리스트에 랜덤으로 값을 넣고 이전 리스트를 가리키는 ptr을 이용하여 list를 연결한다.

* **Insert 함수**

1. Students를 가리키는 ptr를 이용하여 next 링크를 따라 마지막 list의 위치를 찾는다.
2. 새로운 list를 malloc으로 할당하여 입력받은 값을 저장한다.
3. ptr의 next를 새로운 students를 가리키게 하고, 새로운 list의 pre는 ptr을 가리킨다.
4. 명령어로 입력받은 값들은 모두 문자열이므로, 번호와 점수는 정수로 형변환을 한다.

* **Delete 함수**

1. strcmp함수를 이용하여 삭제하고자 하는 이름을 가진 list를 찾는다. Ptr 포인터를 이용하여 원하는 list가 나올 때까지 비교 후 next링크를 따라간다.
2. 만약 지우려는 list가 맨 앞인 경우, 삭제할 list의 다음 list를 맨 처음으로 한다.(pre를 NULL로 함)
3. 지우려는 list가 맨 뒤일 경우, 삭제할 list의 이전 list를 맨 마지막 list로 한다.(next를 NULL로 함)
4. 만약 지우려는 함수가 중간에 있다면, 지우려는 리스트의 이전과 이후의 리스트를 연결시킨다.

* **list함수**

1. 찾고자 하는 학생의 이름이 나올 때까지 next링크를 따라간다.
2. 찾고자 하는 학생의 이름이 나오면, 해당 list의 내용을 출력한다.

* **Sort name함수**

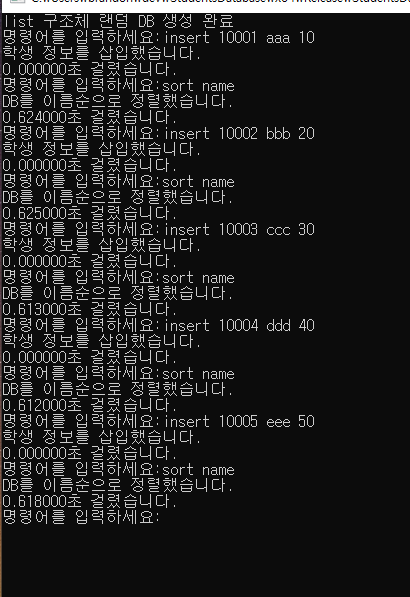
1. List의 i번째 이름과 i+1번째부터 마지막 리스트까지의 이름 중 사전순으로 가장 이전의 값을 비교한다.
2. 만약 i번째 이름보다 그 이후의 이름이 사전순으로 이전이면, i번째 list와 해당 list의 값을 swap한다.
3. 위의 방식으로 i를 처음부터 마지막 list-1까지 실행한다.

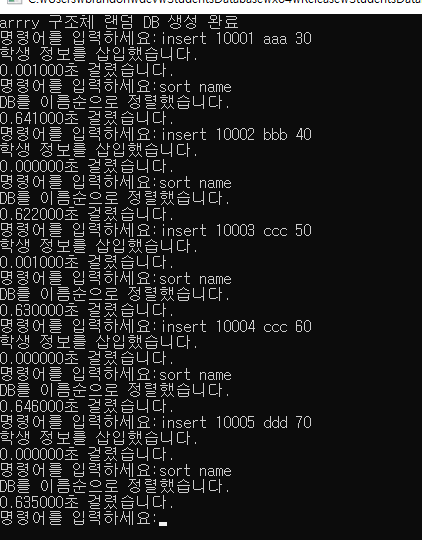
* **Sort score함수**
* Sort name과 같은 방식으로 진행한다. 단, sorting 기준을 score라는 정수값으로 sorting한다.
* **Read DB 함수**

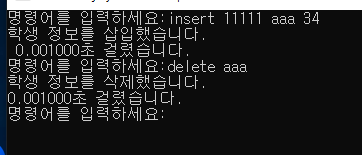
1. 해당 DB가 존재하는지를 알아보는 access함수를 실행한다.(없으면 error)
2. 해당 DB가 존재하면, file 포인터를 사용하여 fileinfo[100]에 한 줄씩 입력받는다.
3. 한 줄씩 입력받아 저장할 때마다, realloc을 사용하여 students의 크기를 늘린다.
4. Fileinfo[100]이 NULL을 받았다면, 더 이상 받을 data가 없다는 의미이므로 함수를 종료한다.

* **Write DB 함수**

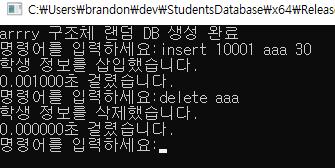
1. file 포인터를 이용하여 마지막 index까지 입력한다.
2. i < \_msize(students) / sizeof(Student)를 이용하여 마지막 index를 구한다.
3. **DB의 크기를 늘리면서 크기 N에 따른 정렬(sorting)에 소요되는 시간**
4. **list방식으로 정렬할 경우**

****

1. **array로 구현한 경우**
2. ****
3. **data를 insert/delete 할 때, 배열로 한 경우와 리스트로 처리한 경우의 시간을 비교**
4. list로 data를 insert/delete한 경우



1. 배열로 data를 insert, delete한 경우



1. **데이터 검색에 소요되는 시간을 줄이는 여러 가지 방법**

-이번 과제의 경우, selection sort로 진행하였으나, sorting 알고리즘 중에서는 selection sort보다 시간 복잡도가 좋은 알고리즘이 존재한다. 시간 복잡도를 Worst Case에 맞추어 비교하면 다음과 같다. Selection sort는 한번 함수가 실행 될 때마다 for문이 중첩되어 있으므로 시간복잡도는 O(n^2)이다.

1) 삽입정렬

: index i를 기준으로 i와 i 이후의 키값을 비교하여 i보다 I 이후의 값이 더 작다면, 작은 값을 원래 자리로 옮긴다. 하나의 배열에 n개가 들어있다고 하면, 시간복잡도는 O(n^2)이 될 것이다.

1. 버블정렬

:index i와 i+1번째 값을 비교하여 i+1 값이 더 작다면 둘의 위치를 swap한다. 이 방식으로 i가 n-1에 올때까지 진행한다. 하나의 배열에 n개가 들어있다고 하면, 시간복잡도는 O(n^2)이다.

1. 병합 정렬

:divide and conquer를 이용한 방법으로, n개의 배열을 반으로 계속 나누면서 하나의 서브 배열안에 1개의 키값 또는 2개의 값을 가지고 있을 때까지 나눈다. 만약 배열이 1개 또는 2개의 값을 가지면, 그 안에서 sorting 후 다시 병합하는 과정을 진행한다. 시간복잡도는 O(nlogn)으로 만약 시간을 줄이려 한다면 병합 정렬이 가장 가능성이 높은 알고리즘이 된다.