2018년 4월 12일 목요일

Homework #2 <Singly Linked List 문제> - document

- Name :
- Student ID :
- Program ID : Homework #2 <Singly Linked List 문제>
- Description :
  - 성적 데이터가 주어졌을 때, 데이터를 오름차순으로 리스트에 담고, 성적과 전공에 따른 필터링 기능과 두개의 리스트를 합치는 기능을 가진 프로그램
- Algorithm :
  - hw2_data1.txt와 hw2_data2.txt에 각각 다른 성적 데이터가 들어있다.
  - 각각의 데이터를 리스트에 넣어준다.
  - **데이터를 insert 함과 동시에, 이름순으로 오름차순 정렬을 해준다. insert 를 하는 알고리즘을 아래와 같다.**
    - temp라는 이름의 새로운 노드를 생성해주고, 이 노드의 데이터를 받아온 데이터로 설정해준다.
    - 만약 리스트의 head가 Null 이라면 head에 해당 노드를 그대로 head에 넣어준다.
    - 만약, head->data.name이 현재 넣으려는 temp->data.name 보다 클 경우, temp.next에 head를 넣어주고, temp를 head로 설정해준다. 오름차순으로 정렬하므로, 먼저 오는 알파벳을 앞으로 위치 시키기 위해 위와 같은 작업을 한다.
    - 위의 경우가 모두 아닌 경우. 즉, 들어온 temp->data.name이 현재 head->data.name 보다 클 경우, 들어갈 자리를 while을 통해 찾는다.
      - p와 q에 head를 저장한다.
      - p가 null이 아니고, p->data.name이 temp->data.name보다 작으면 계속해서 탐색을 한다.
        - 찾지 못한 경우 p를 p->next로 바꾸어, 다음 노드를 검색한다.
      - 만약 p가 null이 아니라면 값을 찾은 것이므로, temp->next를 p로 설정해주고, q->next를 temp로 설정해준다.
      - 만약 p가 null이라면 가장 마지막에 새로운 노드가 위치해야 하는 것이므로, q->next에 temp를 저장시킨다.
  - **리스트를 전공이나 성적을 기준으로 필터링 하여 새로운 리스트를 리턴한다. 이때 인자값으로 char 또는 string형태의 필터링 할 값을 받아준다.**
    - 리턴 할 새로운 리스트 rSll과 임시로 head를 가지고 있을 p를 만들어준다.
    - 만약 빈 리스트가 아니라면,
      - p에 head를 넣어주고, p가 null이 아닐때 까지 while문을 돌면서 필터링을 해준다.
        - 만약, p->data->(grade or major) == find 가 참이라면, rSll에 p->data를 insert하여 데이터를 추가해준다.
        - 다음번째 p를 검사하기 위해 p를 p->next로 바꿔준다.
      - rSll를 리턴해준다.
  - **리스트의 병합은 merge정렬의 원리를 그대로 사용하여, 합쳐진 새로운 리스트를 반환한다.**
    - 두개의 리스트의 head를 각각 sllA, sllB에 저장하고, 합쳐진 리스트를 저장 할 rSll를 만들어준다.
    - 만약, sllA가 null이면 sllB의 데이터를 순차적으로 꺼내 rSll에 insert 해준다.
    - 이와 반대로, sllB가 null이면 sllA의 데이터를 순차적으로 꺼내 rSll에 insert 해준다.
    - 이 모두 아니라면, 즉 두 리스트에 모두 데이터가 존재한다면,
      - sllA->data.name과 sllB->data.name을 비교한다.
      - 만약, 둘사이의 대소관계가 sllA->data.name <= sllB->data.name를 만족 한다면, 알파벳 순으로 전자가 더 우선순위에 있는 것이므로,
        - rSll에 sllA->data를 insert 해주고 sllA를 sllA->next로 바꿔준다.
      - 아니라면, 반대로 sllB->data를 rSll에 insert 해주고, sllB를 sllB->next로 바꿔준다.
    - rSll를 리턴해준다.
- Function :
  - main (main.cpp) :
    - hw2_data1.txt와 hw2_data2.txt에서 데이터를 읽어와 각각 리스트 sll[0]과 sll[1]에 insert 해준다.
    - 메소드 체이닝을 사용하여 출력, 필터링 후 출력, 합병과 필터링 후 출력을 실행한다.
  - insert (hw2_sll.cpp) :
    - Student 데이터를 오름차순으로 리스트에 넣어준다.
  - filterScoreList, filterMajorList (hw2_sll.cpp) :
    - 인자로 받은 점수(char) 또는 전공(string)을 기준으로 리스트의 데이터를 필터링하여 새로운 리스트를 만들고, 이를 리턴한다.
  - merge (hw2_sll.cpp) :
    - 인자로 받은 리스트와 해당 리스트(this)를 merge 정렬 알고리즘을 이용해 합병한다. 새롭게 만들어진 리스트를 리턴한다.

    - isEmpty (hw2_sll.cpp) :
        - 리스트의 요소가 있는지 없는지 검사한다. 있다면 True, 없다면 False 리턴
    - traverseList (hw2_sll.cpp) :
        - 리스트를 돌면서 모든 요소를 출력한다. 만약, 요소가 없다면 에러 메시지를 띄운다.
- 코드 상세 설명
    - main (main.cpp)
        - description :
            - hw2_data1.txt와 hw2_data2.txt에서 데이터를 읽어와 각각 리스트 sll[0]과 sll[1]에 insert 해준다.
            - 메소드 체이닝을 사용하여 출력, 필터링 후 출력, 합병과 필터링 후 출력을 실행한다.
        - variables
            - hw2_sll sll[2] : hw2_data1.txt와 hw2_data2.txt의 데이터를 저장 할 리스트이다.

```
#include <iostream>
#include <fstream>
#include "hw2_sll.h"                                    // singly linked list 를 구현한 header이다

using namespace std;

int main() {
    ifstream inStream;
    int numTestCases;
    string filename[2] = {                              // 두개의 데이터파일의 이름을 저장한다.
        "hw2_data1.txt", "hw2_data2.txt"
    }

    hw2_sll sll[2];                                     // 두개의 데이터를 각각 담을 리스트 배열을 만든다
    for (int i = 0; i < 2; ++i) {                       // for 문을 돌면서 데이터를 넣어준다.
        inStream.open(filename[i]);
        if (inStream.fail()) {
            cerr << "Input file opening failed.\n";
            exit(1);
        }

        inStream >> numTestCases;
        for (int j = 0; j < numTestCases; j++) {
            Student student;                            // Student 구조체에 정보를 알맞은 위치에 넣어준다.
            inStream >> student.name >> student.id >> student.major >> student.grade;
            sll[i].insert(student);                     // 리스트에 구조체를 insert 해준다.
        }

        inStream.close();
    }

    cout << "1) 학생 이름(Name)의 알파벳 오름차순으로 리스트를 만들어서 출력하라." << endl;
    sll[0].traverseList();                              // sll[0], 첫번째 데이터가 담긴 리스트를 전체 출력한다.

    cout << "2)에서 만든 list에서 Grade \"A\" 성적을 받은 사람만 출력하라." << endl;
    sll[0].filterScoreList('A').traverseList();         // 첫번째 데이터에서 점수가 A인 학생만 필터링한후 출력한다.

    cout << R"(3) "data2" 파일과 "data1"을 합쳐서 하나의 리스트로 만들고, 그 중 CS 전공 학생들만 출력하라.)" <<
endl;
    sll[0].merge(sll[1]).filterMajorList(“CS").traverseList();  // 첫번째 리스트와 두번째 리스트를 병합하고, 전공이
                                                        // CS인 학생을 필터링한다. 그리고 이를 출력한다.

    return 0;
}
```

- hw2_sll() (hw2_sll.cpp)
  - description :
    - hw2_sll 객체 생성시, head를 null로 초기화 해주는 생성자이다.

```
hw2_sll::hw2_sll() {                    // hw2_sll 객체 생성시 아무 인자가 없으면, head=nullptr로 초기화 시켜주는
   head = nullptr;                      // 생성자이다.
}
```

- insert(Student newbie) (hw2_sll.cpp)
  - description :
    - Student 구조체의 이름을 기준으로, 데이터가 오름차순이 되게 리스트에 저장한다.
    - 만약 head가 null이면 그대로 넣어주고, head가 temp보다 더 클 경우 head를 temp.next로 넣어준다.
    - p가 null이 아니고, p의 이름이 temp의 이름보다 작을때 까지 위치를 탐색한다.
    - 만약 p가 null이 아닐 경우, q->temp->p 순으로 데이터를 넣어준다.
    - 만약 p가 null일 경우, temp가 마지막 자리에 위치한다는 뜻이므로, q 다음에 temp를 넣어준다.
  - variables
    - temp : 받은 데이터를 임시 저장할 노드
    - p, q : head의 위치를 바꿔가며 데이터를 비교하고, 위치를 찾을 임시 변수
    - head : 리스트가 현재 가르키고 있는 저장소

```
void hw2_sll::insert(Student newbie) {                  // Student 구조체를 인자로 받아 리스트에 넣는다.
   Node *temp = new Node(newbie);                       // 받은 데이터를 임시저장할 temp 노드를 생성한다.
   Node *p, *q;                                         // 이름 오름차순으로 저장하기 위해 임시변수를 선언한다.

   if (head == nullptr) {                               // head가 null이면 그대로 받은 데이터를 넣어준다.
      head = temp;
   } else if (temp->data.name < head->data.name) {      // 만약 현재 head 이름이 temp 이름보다 알파벳
      temp->next = head;                                // 뒤에 존재 할 경우 현재 head를 temp next로 넣어주고
      head = temp;                                      // head를 temp로 바꿔준다.
   } else {
      p = q = head;                                     // 위 모두 아닐 경우 위치를 찾기 위해 p, q를 초기화한다

      while (p != nullptr && p->data.name < temp->data.name) { // p가 null이 아니고, p 이름이 temp 이름
         q = p;                                         // 보다 작을때 까지 계속해서 자리를 찾는다
         p = p->next;                                   // p를 p.next로 바꾸면서 다음을 탐색
      }

      if (p != nullptr) {                               // 만약 p가 null이 아니라면 자리를 찾은 것이므로
         temp->next = p;                                // temp.next를 p로 설정하고, q의 next를 temp로
         q->next = temp;                                // 설정하여 q->temp->p 순으로 오름차순이 되게 한다.
      } else { q->next = temp; }                        // 만약 null일 경우 마지막 자리에 temp를 넣어준다.
   }
}
```

- filterScoreList(char findScore) (hw2_sll.cpp)
  - description :
    - p(head)가 null이 아니면 p의 위치를 다음으로 바꿔가면서 끝까지 탐색한다.
    - 문자를 비교하는 것이므로, == 를 이용하여 p의 grade와 findScore의 값을 비교한다.
    - 만약 같다면 리턴 할 리스트 rSll에 데이터를 insert한다.
  - variables
    - p : 현재 head를 임시저장하고, head를 바꿔가며 데이터를 검사할 임시 노드
    - rSll : 필터링 된 구조체들을 저장 할 리스트
    - findScore : 찾는 기준이 될 성적 인자값 (char)

```
hw2_sll hw2_sll::filterScoreList(char findScore) {                  // 필터링 할 문자 타입의 점수를 인자로 받는다
   Node *p;                                                         // 리스트를 탐색하기 위해 임시변수를 만든다.
   hw2_sll rSll;                                                    // 필터링 할 리스트를 담을 새로운 변수를 만든다

   if (!isEmpty()) {                                                // 찾는 대상이 되는 리스트가 비어있지 않으면 실행
      p = head;                                                     // p를 head로 초기화한다.
      while (p) {                                                   // p가 null이 아니면 반복한다
         if (p->data.grade == findScore) { rSll.insert(p->data); }  // p의 grade 가 찾는 findScore와 같으면
         p = p->next;                                               // 값을 넣어준다. 이후 p를 p->next로 바꿔 다음을
      }                                                             // 찾는다.
   }
   return rSll;                                                     // rSll 리스트를 리턴한다.
}
```

- filterMajorList(string findMajor) (hw2_sll.cpp)
  - description :
    - p(head)가 null이 아니면 p의 위치를 다음으로 바꿔가면서 끝까지 탐색한다.
    - 문자열를 비교하는 것이므로, compare 함수를 이용하여 p의 major와 findMajor의 값을 비교한다.
    - 만약 같다면 리턴 할 리스트 rSll에 데이터를 insert한다.
  - variables
    - p : 현재 head를 임시저장하고, head를 바꿔가며 데이터를 검사할 임시 노드
    - rSll : 필터링 된 구조체들을 저장 할 리스트
    - findMajor : 찾는 기준이 될 전공 인자값 (string)

```
hw2_sll hw2_sll::filterMajorList(std::string findMajor) {   // 필터링 할 문자열 타입의 전공을 인자로 받는다
   Node *p;                                                 // 리스트를 탐색하기 위해 임시 변수를 만든다.
   hw2_sll rSll;                                            // 필터링 할 리스트를 담을 새로운 변수를 만든다

   if (!isEmpty()) {                                        // 찾는 대상이 되는 리스트가 비어있지 않으면 실행
      p = head;                                             // p를 head로 초기화한다
      while (p) {                                           // p가 null이 아니면 반복한다
         if (!p->data.major.compare(findMajor)) {           // compare함수는 같으면 0을 리턴한다. 우리는
               rSll.insert(p->data);                        // 같을때 값을 넣어야하므로, not을 취해 True로 만들고,
         }                                                  // 두 값이 같을때 데이터를 rSll에 insert 해준다.
         p = p->next;                                       // p를 p->next로 바꿔 다음을 찾는다.
      }
   }
   return rSll;                                             // rSll 리스트를 리턴한다.
}
```

- merge(hw2_sll sll) (hw2_sll.cpp)
  - description :
    - Merge 정렬 알고리즘을 사용하여 데이터를 병합한다.
    - 병합한후, 병합한 리스트를 리턴해준다.
    - 해당 라인에 대한 자세한 설명은 아래에 주석처리 되어 있습니다.
  - variables
    - sllA, sllB : 자기 자신과 인자로 받은 리스트의 head를 저장할 노드
    - rSll : 병합 된 리스트를 관리하고, 리턴 할 리스트

```
hw2_sll hw2_sll::merge(hw2_sll sll) {                  // merge할 리스트를 인자로 받는다.

    Node *sllA = nullptr, *sllB = nullptr;             // 자기 자신과 인자로 받은 리스트의 head를 각각 저장한다.
    sllA = this->head;
    sllB = sll.head;

    hw2_sll rSll;                                      // 합쳐진 데이터를 리턴 할 리스트를 만든다.
                                                       // Merge 정렬의 원리를 이용하여 병합한다.
    while (true) {                                     // 안에서 break가 될 때까지 무한 루프를 돈다.
        if (sllA == nullptr) {                         // 만약 sllA가 null이라면
            while (sllB != nullptr) {                  // sllB의 데이터를 모두 rSll에 insert 해준다.
                rSll.insert(sllB->data);
                sllB = sllB->next;
            }
            break;                                     // break 로 밖으로 나감
        } else if (sllB == nullptr) {                  // 만약, sllB가 null 이라면
            while (sllA != nullptr) {                  // sllA가 null이 아닐때까지 while문을 돌면서
                rSll.insert(sllA->data);               // sllA의 데이터를 모두 rSll에 insert 해준다.
                sllA = sllA->next;
            }
            break;                                     // break 로 밖으로 나감
        } else {                                       // 위의 케이스가 모두 아니라면
            if (sllA->data.name <= sllB->data.name) {// 만약 sllA의 이름이 sllB의 이름보다 작거나 같다면
                rSll.insert(sllA->data);               // sllA의 데이터를 rSll에 insert하고,
                sllA = sllA->next;                     // sllA를 sllA->next로 바꿔준다.
            } else {                                   // 만약 sllA의 이름이 sllB의 이름보다 크다면
                rSll.insert(sllB->data);               // sllB의 데이터를 rSll에 insert하고,
                sllB = sllB->next;                     // sllB를 sllB->로 바꿔준다.
            }
        }
    }
    return rSll;                                       // rSll를 리턴한다.
}
```

- isEmpty() (hw2_sll.cpp)
  - description :
    - 리스트가 비어있는지 확인하는 함수이다.
    - 비어있다면 True를, 데이터가 존재한다면 False를 리턴한다.

```
bool hw2_sll::isEmpty() {
    return head == nullptr;                            // head 가 null과 같다면 True를 반환 할 것이므로 리스트 내에
}                                                      // 데이터 유무를 쉽게 확인 할 수 있다.
```

- traverseList() (hw2_sll.cpp)
  - description :
    - 리스트 내의 모든 데이터를 출력한다.
    - 만약, 데이터가 존재하지 않을 경우 리스트가 비어있다는 에러 메시지를 출력한다.
  - variables
    - p : 현재 리스트의 head를 담고, head를 이동하며 출력 할 노드

```
void hw2_sll::traverseList() {
   Node *p;

   if (!isEmpty()) {                                  // 현재 리스트가 비어있지 않으면 해당 코드를 실행 시킨다.
      p = head;                                       // p를 head로 초기화한다.

      while (p) {                                     // p가 null이 아닐때 까지 반복한다.
         cout << p->data.name << "\t";                // 데이터의 이름, 아이디, 전공, 성적을 순서대로 출력한다.
         cout << p->data.id << "\t";
         cout << p->data.major << "\t";
         cout << p->data.grade << endl;
         p = p->next;                                 // 다음번째 데이터를 출력하기 위해, p를 p->next로 바꾼다.
      }
      cout << endl;                                   // 마지막번째에 개행을 해준다.
   } else { cout << "List is Empty!" << endl; }       // 만약, 리스트가 비어있다면, 리스트가 비어있다는 에러를 띄운다.
}
```

- hw2_sll.h
  - description :
    - hw2_sll.cpp의 header 파일이다.
    - Student 구조체와 Node 클래스와, hw2_sll 클래스가 정의 되어있다.

```
#include <iostream>

struct Student {                                      // Student 구조체를 정의한다
   char name;                                         // 학생의 이름 ex) A, B etc..
   int id;                                            // 학생의 아이디 ex) 100, 200, 600 etc..
   std::string major;                                 // 학생의 전공 ex) CS, EE etc..
   char grade;                                        // 학생의 성적 ex) A, B, C etc..
};

class Node {                                          // Node를 클래스로 정의한다.
private:
   Student data;                                      // Node의 데이터는 Student를 받으며,
   Node *next;                                        // 다음에 어떤 노드가 올지 저장하는 next가 있다.

   friend class hw2_sll;

   Node() {                                           // 기본 생성자에서는 next를 null로 초기화해준다.
      this->next = nullptr;
   }

   Node(Student data) {                               // 오버로딩 되어 Student가 들어오면, 이 값을 data로 넣고
      this->data = data;                              // next를 null로 초기화해준다.
      this->next = nullptr;
   }
};                                                    // 다음장에 계속
```

```
class hw2_sll {                                  // 리스트를 hw2_sll이라는 이름의 클래스로 정의한다.
private:
    Node *head;                                  // 현재 노드를 저장할 head를 만든다.

public:                                          // 클래스의 함수를 선언한 부분
    hw2_sll();

    void insert(Student);

    hw2_sll merge(hw2_sll);

    bool isEmpty();

    void traverseList();

    hw2_sll filterScoreList(char);

    hw2_sll filterMajorList(std::string);

};
```

- 실행 화면 캡쳐