REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과　　목 : 전자전기 프로그래밍실습
과 제 명 : HW8
담당교수 : 민 형 복
학　　과 : 전자전기공학부
학　　년 : 2
학　　번 : 2017311583
이　　름 : 정해진
제 출 일 : 2018.6.4

1. Introduction

[2]에 의하면, Linked List의 정의는 이렇다.

이것은 각 노드를 유용한 저장 공간에 그 위치에 상관없이 저장시키고, 각 노드의 관련성을 노드에 보관하여 1차원 배열 관계를 유지하도록 함으로써 중간 노드의 삽입, 제거를 손쉽게 할 수 있는 리스트로 각 노드는 링크(또는 포인터) 부분을 가지며, 그 노드와 관련 있는 다음 노드의 주소를 그 값으로 가진다. 다시 말하면 선형 리스트의 노드 배열이 어드레스와 일치하지 않고 기억 공간에 독립적으로 이루어진 리스트를 말한다.

새로운 타입의 리스트 'Linked List"를 활용하여 주어진 조건 속에서 합집합, 교집합 등을 만드는 문제를 해결한다.

2. Problem Statement

① Describe what is the problem.

7개의 함수들 코드 작성.

- int addElement(NODE **set, int i_data);
- int deleteElement(NODE **set, int *i_data);
- int isMember(NODE *set, int i_data);
- int isEmpty(NODE *set);
- NODE *setUnion(NODE *set1, NODE *set2);
- NODE *setIntersection(NODE *set1, NODE *set2);
- void deleteSet(NODE **set);

이 함수는 주어진다.

- printSet(NODE *set);

② Describe how do you solve the problem.

- Main 함수

다음의 함수들을 모두 실행하여 조건에 맞는 결과를 도출한다.

- int addElement(NODE **set, int i_data);
- int deleteElement(NODE **set, int *i_data);
- int isMember(NODE *set, int i_data);
- int isEmpty(NODE *set);
- NODE *setUnion(NODE *set1, NODE *set2);
- NODE *setIntersection(NODE *set1, NODE *set2);
- void deleteSet(NODE **set);

- addElement 함수

주어진 head pointer를 가지는 linked list에 data를 list에 오름차순으로 삽입한다.

- deleteElement 함수

주어진 head pointer를 가지는 linked list에서 해당하는 data를 삭제한다.

- isMember 함수

주어진 head pointer 를 가지는 linked list에 data가 존재하는가를 판별한다.

- isEmpty 함수

주어진 head pointer 를 가지는 linked list가 비어있는 리스트인지 판별한다.

- setUnion 함수

주어진 head pointer를 가지는 두 개의 linked list의 합집합을 구한다.

- setIntersection 함수

주어진 head pointer를 가지는 두 개의 linked list의 교집합을 구한다.

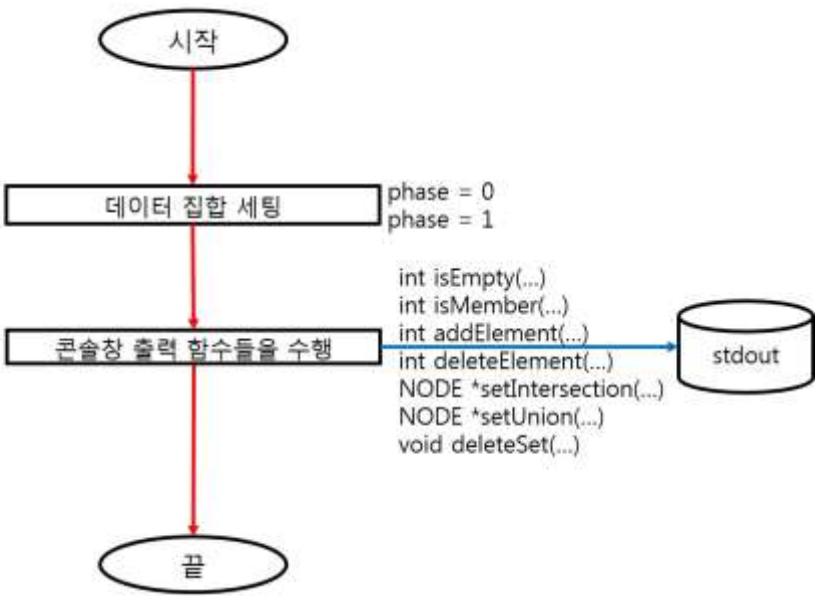
- deleteSet 함수

주어진 head pointer를 가지는 linked list를 삭제한다.

③ Draw a flowchart of your algorithm

1. Main 함수

int main(void)



3. Implementation

- Main 함수

다음의 함수들을 모두 실행하여 조건에 맞는 결과를 도출한다.

- int addElement(NODE **set, int i_data);
- int deleteElement(NODE **set, int *i_data);
- int isMember(NODE *set, int i_data);
- int isEmpty(NODE *set);
- NODE *setUnion(NODE *set1, NODE *set2);
- NODE *setIntersection(NODE *set1, NODE *set2);
- void deleteSet(NODE **set);

- addElement 함수

주어진 head pointer를 가지는 linked list에 i_data를 list에 오름차순으로 삽입한다. 이미 list에 data가 존재하는 경우도 생각하여, memory allocation을 수행한다.

- deleteElement 함수

주어진 head pointer를 가지는 linked list에서 해당하는 *p_i_data를 삭제한다. *p_i_data가 0인 경우, linked list의 맨 앞의 데이터를 삭제한다.

- isMember 함수

주어진 head pointer 를 가지는 linked list에 i_data가 존재하는가를 판별한다. 존재하면 1을, 존재하지 않으면 0을 리턴한다.

- isEmpty 함수

주어진 head pointer 를 가지는 linked list가 비어있는 리스트인지 판별한다. 존재하지 않으면 1, 존재하면 0을 리턴한다.

- setUnion 함수

주어진 head pointer를 가지는 두 개의 linked list의 합집합을 구한다. 합집합을 생성하는 과정에서 addElement2 함수를 이용한다. 알고리즘은 [1]의 슬라이드 6~12에 해당하는 내용을 이용한다.

- setIntersection 함수

주어진 head pointer를 가지는 두 개의 linked list의 교집합을 구한다. 교집합을 생성하는 과정에서 addElement2 함수를 이용한다. 알고리즘은 [1]의 슬라이드 6~12에 해당하는 내용을 이용한다.

- deleteSet 함수
주어진 head pointer를 가지는 linked list를 삭제한다.

4. Result

"2017311583.정해진.HW8.c" 를 컴파일 후 실행한 결과이다.

```
Perform addElement at phase 0
  addElement (Set 1) expected  : 11 18 23 43 51 65 76 89 99
  addElement (Set 1) performed : 11 18 23 43 51 65 76 89 99

Perform deleteElement at phase 0
  deleting an element '51' : 11 18 23 43 51 65 76 89 99
  deleting an element '18' : 11 18 23 43 65 76 89 99
  deleting an element '65' : 11 23 43 65 76 89 99
  deleting an element '43' : 11 23 43 76 89 99
  deleting an element '11' : 11 23 76 89 99
  deleting an element '76' : 23 76 89 99
  deleting an element '23' : 23 89 99
  deleting an element '99' : 89 99
  deleting an element '89' : 89
  deleteElement performed okay.

Perform addElement again at phase 0
  addElement expected  : 11 18 23 43 51 65 76 89 99
  addElement performed : 11 18 23 43 51 65 76 89 99

Perform deleteElement at phase 0
  deleting a smallest element : 11 18 23 43 51 65 76 89 99
  deleting a smallest element : 18 23 43 51 65 76 89 99
  deleting a smallest element : 23 43 51 65 76 89 99
  deleting a smallest element : 43 51 65 76 89 99
  deleting a smallest element : 51 65 76 89 99
  deleting a smallest element : 65 76 89 99
  deleting a smallest element : 76 89 99
  deleting a smallest element : 89 99
  deleting a smallest element : 99
  deleteElement performed okay.

Creating Set 1 for set union/intersection
  Set 1 expected  = { 11 18 23 43 51 65 76 89 99 }
  Set 1 performed = { 11 18 23 43 51 65 76 89 99 }

Creating Set 2 for set union/intersection
  Set 2 expected  = { 17 23 43 55 65 }
  Set 2 performed = { 17 23 43 55 65 }

Union of Set1 and an Empty set at phase 0
  Set1 U Empty-set expected  = { 11 18 23 43 51 65 76 89 99 }
  Set1 U Empty-set performed = { 11 18 23 43 51 65 76 89 99 }

Union of an Empty set and Set2 at phase 0
  Empty-set U Set2 expected  = { 17 23 43 55 65 }
  Empty-set U Set2 performed = { 17 23 43 55 65 }

Union of two sets, (Set1 and Set2) at phase 0
  Set1 U Set2 expected  = { 11 17 18 23 43 51 55 65 76 89 99 }
  Set1 U Set2 performed = { 11 17 18 23 43 51 55 65 76 89 99 }

Intersection of Set1 and an Empty set at phase 0
  Set1 ^ Empty-set expected  = { }
  Set1 ^ Empty-set performed = { }

Intersection of an Empty set and Set2 at phase 0
  Empty-set ^ Set2 expected  = { }
  Empty-set ^ Set2 performed = { }
```

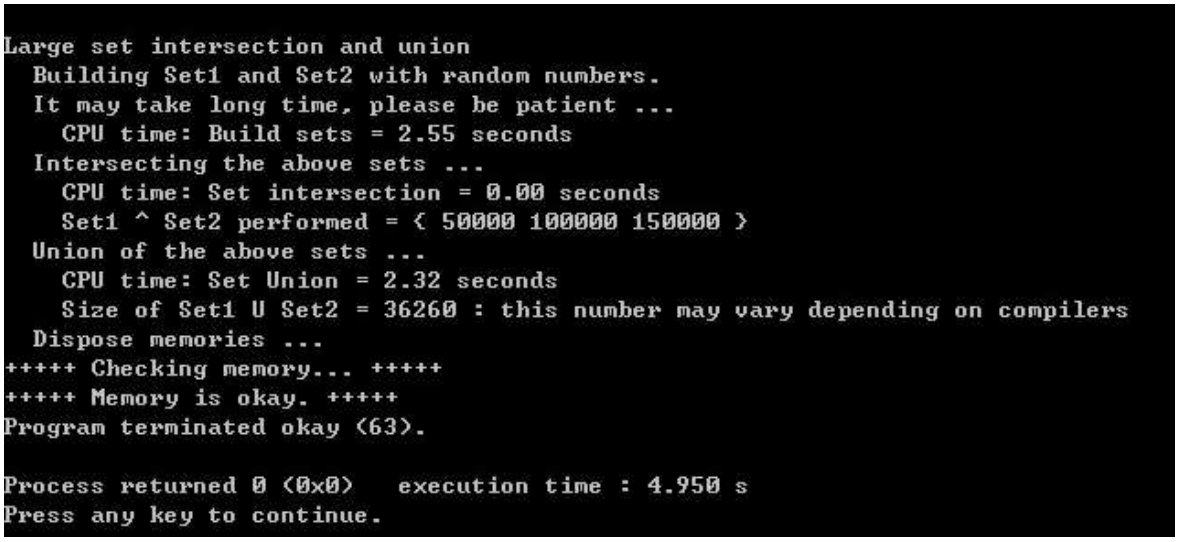
```
Intersection of two sets, (Set1 and Set2) at phase 0
  Set1 ^ Set2 expected  = { 23 43 65 }
  Set1 ^ Set2 performed = { 23 43 65 }

Deleting current sets at phase 0
deleteSet performed okay
```

Phase 0 에 대한 콘솔 결과 창이다.

```
Perform addElement at phase 1
  addElement (Set 1) expected  : 128 256 328 415 512
  addElement (Set 1) performed : 128 256 328 415 512

Perform deleteElement at phase 1
  deleting an element '512' : 128 256 328 415 512
  deleting an element '328' : 128 256 328 415
  deleting an element '128' : 128 256 415
  deleting an element '415' : 256 415
  deleting an element '256' : 256
  deleteElement performed okay.

Perform addElement again at phase 1
  addElement expected  : 128 256 328 415 512
  addElement performed : 128 256 328 415 512

Perform deleteElement at phase 1
  deleting a smallest element : 128 256 328 415 512
  deleting a smallest element : 256 328 415 512
  deleting a smallest element : 328 415 512
  deleting a smallest element : 415 512
  deleting a smallest element : 512
  deleteElement performed okay.

Creating Set 1 for set union/intersection
  Set 1 expected  = { 128 256 328 415 512 }
  Set 1 performed = { 128 256 328 415 512 }

Creating Set 2 for set union/intersection
  Set 2 expected  = { 202 303 404 }
  Set 2 performed = { 202 303 404 }

Union of Set1 and an Empty set at phase 1
  Set1 U Empty-set expected  = { 128 256 328 415 512 }
  Set1 U Empty-set performed = { 128 256 328 415 512 }

Union of an Empty set and Set2 at phase 1
  Empty-set U Set2 expected  = { 202 303 404 }
  Empty-set U Set2 performed = { 202 303 404 }

Union of two sets, (Set1 and Set2) at phase 1
  Set1 U Set2 expected  = { 128 202 256 303 328 404 415 512 }
  Set1 U Set2 performed = { 128 202 256 303 328 404 415 512 }

Intersection of Set1 and an Empty set at phase 1
  Set1 ^ Empty-set expected  = { }
  Set1 ^ Empty-set performed = { }

Intersection of an Empty set and Set2 at phase 1
  Empty-set ^ Set2 expected  = { }
  Empty-set ^ Set2 performed = { }
```

```
Intersection of two sets, (Set1 and Set2) at phase 1
  Set1 ^ Set2 expected  = { }
  Set1 ^ Set2 performed = { }

Deleting current sets at phase 1
deleteSet performed okay
```

Phase 1 에 대한 콘솔 결과 창이다.



데이터가 방대한 두 집합에 대하여 실행한 콘솔 결과 창이다. 계산 시간이 모범 답안과 비교하여 길게 도출되었다. 이 이유로, "setIntersection", "setUnion" 함수가 각각 교집합과 합집합에 들어가는 데이터들을 저장할 때 사용하는 함수 "addElement2"에서 비효율적인 과정이 들어가기 때문이라고 추정한다. addElement2 함수에서 사용하는 방법은 주어진 링크 리스트의 맨 마지막에 데이터를 추가하는 방식이다. 그런데 위처럼 데이터가 방대할 경우에는 주어진 링크 리스트의 맨 끝을 찾는데 걸리는 시간이 오래 걸릴 수밖에 없다. 따라서 위 함수를 이용하는 방법이 아닌, 데이터가 들어올 때마다 바로바로 이어 붙이는 알고리즘을 생각했다. 그러나 포인터 처리 중에 memory allocation을 수행할 포인터와 링크 리스트의 맨 끝으로 포인터를 이동하는 그 두 가지 일을 수행할 수 없어 결국 위와 같은 방식을 택하였다.

5. Conclusion & Evaluation

실습을 통해 저번 실습에서 다뤘던 linked list를 활용하여 집합 형태로 다루는 방법에 대해 잘 알게 되었다.

6. 참고 문헌

[1] Min, H. B. and SKKU, “set_slide.pdf”
[2]컴퓨터인터넷 IT 용어대사전, “연결된 리스트”, 2011. 1. 20.