**2019년 2학기**

**프로그래밍과 문제해결**

**Assignment #5**

|  |  |
| --- | --- |
| 담당교수 | **윤은영** |
| 학번 | **20190084** |
| 학과 | **무은재학부** |
| 이름 | **권민재** |
| POVIS ID | **mzg00** |

|  |
| --- |
| **명예서약(Honor code)**  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem 1, 2**

**1. 문제의 개요**

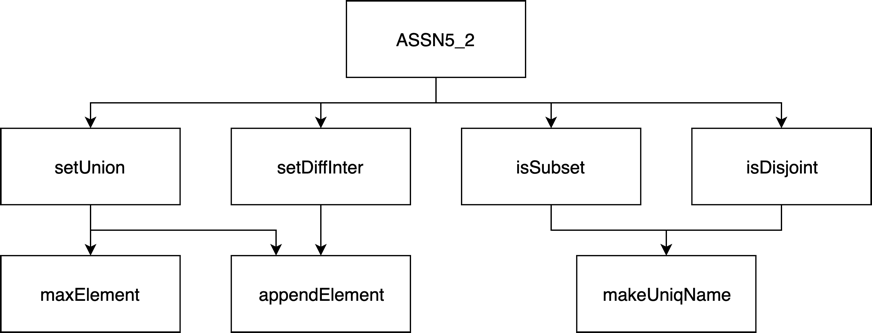
본 문제의 개요는 아래와 같다.

* 본 문제에서는 정수를 원소로 가지는 집합에 대해 여러 기본 기능을 수행하는 프로그램을 작성하고자 한다.
* 이때, 기본 기능으로는 집합 생성, 집합 출력, 멤버 확인, 원소 추가, 원소 삭제, 집합 삭제로 총 6개가 있으며, 명령어는 종료 명령어까지 7개를 포함한다.
* 이때, Problem 1의 내용을 기반으로 추가적인 기능을 구현하여 Problem 2를 해결한다.
* 정수를 원소로 가지는 두 집합 A와 B에 대하여 합집합, 교집합, 차집합을 구하고, 두 집합의 서로소 여부와 두 집합의 포함 여부를 출력하는 기능을 추가적으로 구현한다.

이 때 사용되는 구조 차트는 아래와 같이 표현될 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명



* **입력부:** 입력부는 사용자 또는 파일으로부터 명령을 입력받고 기능을 분배하는 역할을 한다.
* **처리부:** 처리부는 입력받은 명령에 따라 주어진 집합의 관련 기능들을 수행한다.
* **출력부:** 집합을 출력하는 명령에 따라 집합이나 0 또는 1을 적절하게 출력한다.

**2. 알고리즘**

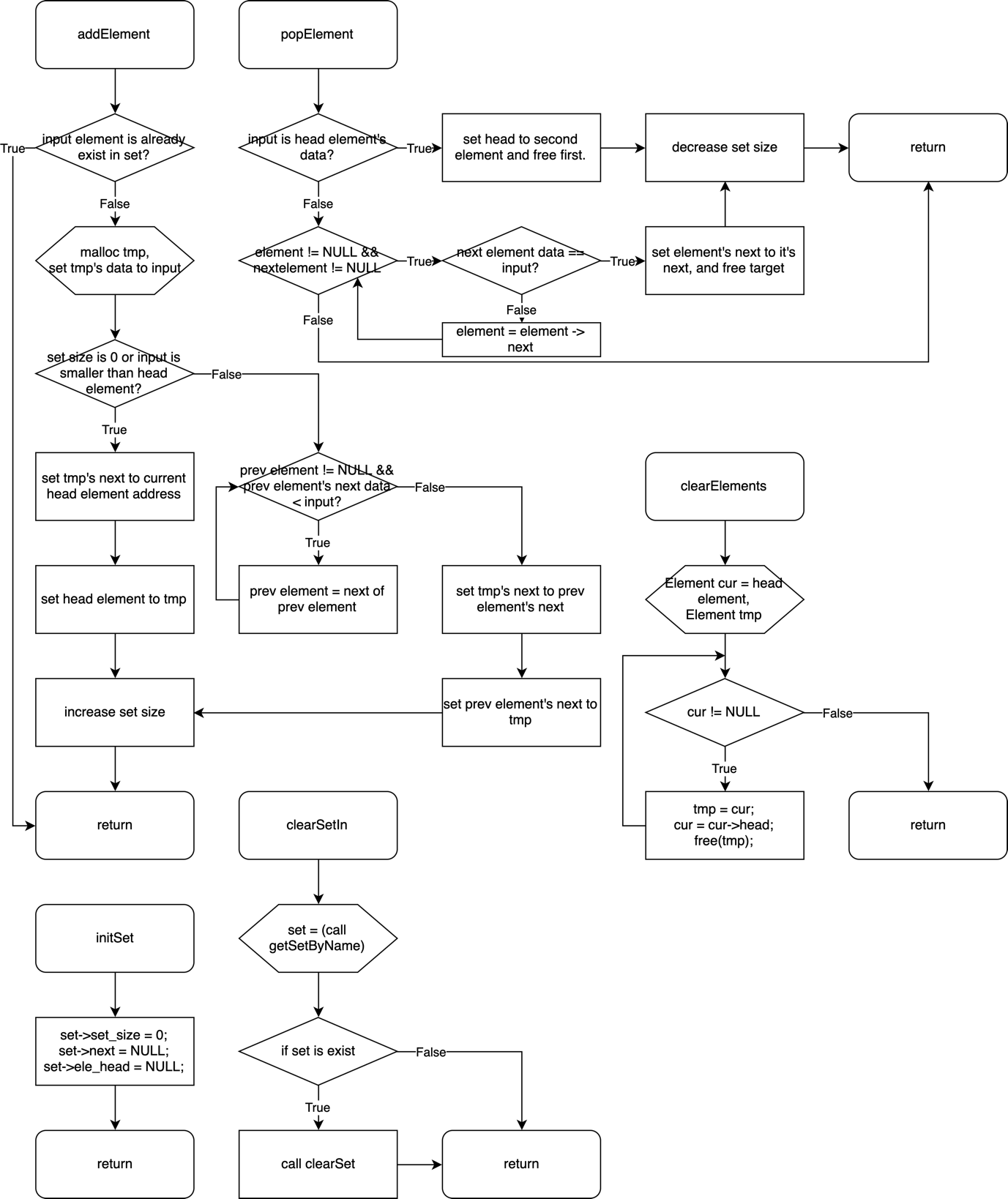
해당 프로그램의 의사 코드는 아래와 같다.

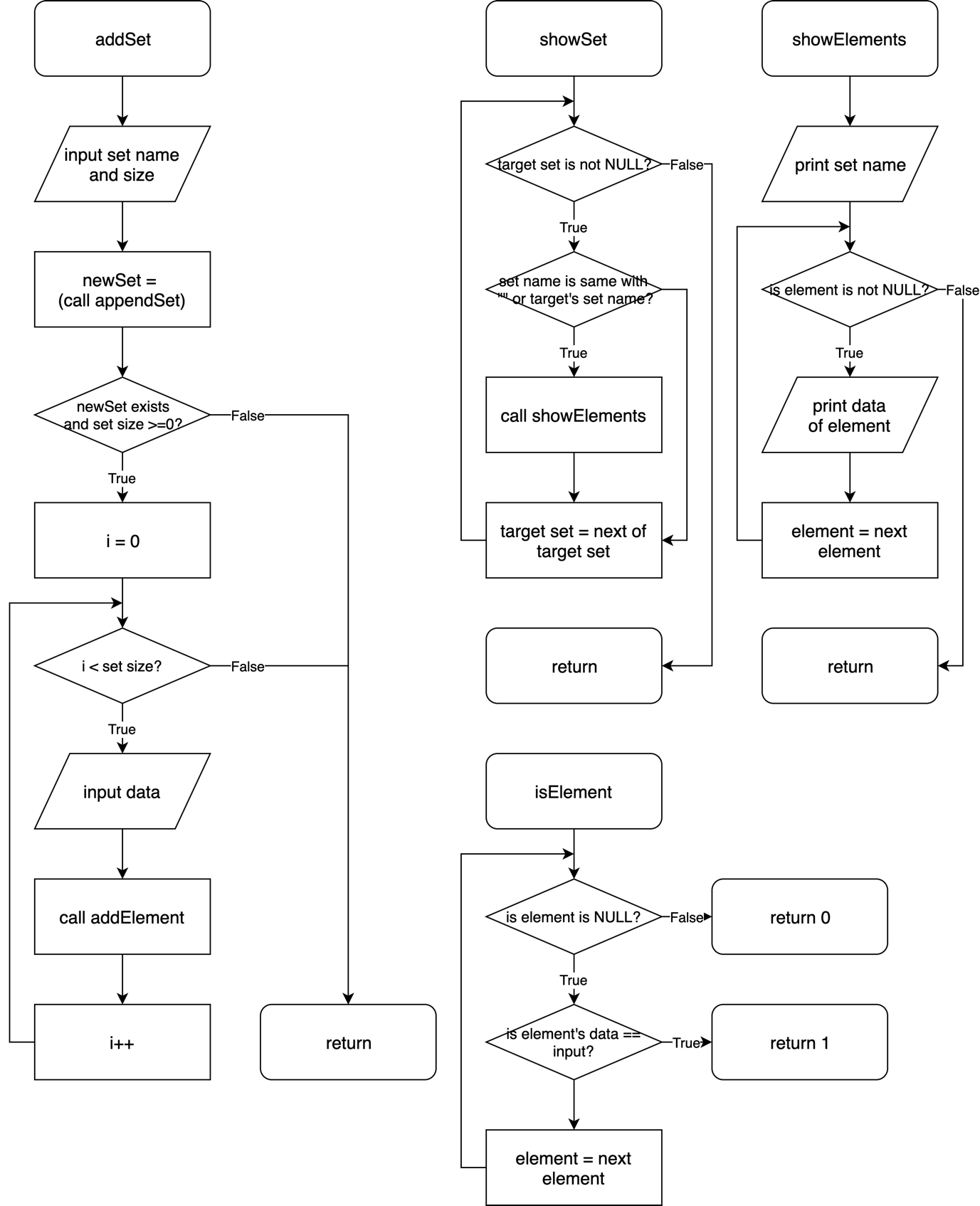
|  |  |
| --- | --- |
| Pseudo-code for ASSN5 <set.c> | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | // 프로그램에 필요한 변수들은 미리 선언한 것으로 가정하자.  include essential headers <stdio.h>, <stdlib.h>, <string.h>  include custom header “set.h”  function *addSet*  input name and size of set  make new set by calling *appendSet*  during size of set, input data from user and add it to set by calling *addElement*  function *showSet*  during set is not null,  if set name is decided, showing that set by calling *showElements*  else, showing all set by calling *showElements*  function *showElements*  during element is not null,  print data of element.  function *isElement*  during element is not null, if there is element same with input, return 1. else, 0.  function *addElement*  check input data is already exist.  if not, make temp element with input data.  if set size is 0, temp element will be head.  else,  if data is smaller than head of set’s element, temp element wil be head of set’s elements.  else, find location to sort in ascending order and insert temp element to that location.  function *popElement*  if data is same with set’s head data, set set’s head to second element and free first element.  else, find element whose data is same, and free it. Link previous to next.  function *clearSet*  if target set is head set, set head set to second set and free head set with calling *clearElements*  else, find target set in set list and free it with calling *clearElements*. Link previous and next set properly.  function *clearElements*  during elements is not NULL, save next element as tmp and free current element. Set next target element to tmp.  function *clearSetIn*  find set with calling *getSetByName*  if exist, call *clearSet* with it.  function *clearAllSet*  during set is not NULL, save next set as tmp and free current set with calling calling *clearElements.* Set next target set to tmp.  function *initSet*  Set target set size to 0, next set address to NULL, element head address to NULL  function *executer*  get function pointer and parameters.  find set by calling *getSetByName* and execute function with it.  function *getSetByName*  during set is not NULL,  if target set name and set’s name is same, return that set’s address.  if there’s not matched set, return 0.  function *appendSet*  check set name is already exist by calling *getSetByName*  if set name is not already exist, make new set at tail of set list  function *appendElement*  // target element is input element same as current set’s last element.  make tmp element with input data  if target element is null, make set’s head tmp element.  else, make next of target element to tmp.  set target element to tmp element. // change current last element.  increase set’s size.  function *setUnion*  get target set 1 and target set 2 by calling *getSetByName* with input set names.  if two target sets don’t exist, return 0.  get element head 1 and 2 from each sets.  make result set by calling *appendSet* with input result set name.  during element 1 or 2 is not NULL,  get smaller element by calling *minElement*  add element to result set by calling *appendElement* with smaller element’s data  if two element’s data is same, make elements to its next.  else make smaller element to its next.  return address of result set  function *setDiffInter*  get target set 1 and target set 2 by calling *getSetByName* with input set names.  if two target sets don’t exist, return 0.  set element 2 to head element of set 2.  get element head 1 and 2 from each sets.  make result set by calling *appendSet* with input result set name.  loop in elements of set 1,  set element 2 to the first element above the element 1’s data.  To get difference, add this element to result set by calling *appendElement*  when current element1 and element 1 is different.  To get intersection, add this element to result set by calling *appendElement* when current element and element 2 is same.  function *isSubset\_Disjoint*  make unique name by calling *makeUniqName*  make result set and save difference set by calling *setDiffInter* with input set names and unique name.  return size of result set is 0 or not, and clear result set.  function *makeUniqName*  Until the same name doesn’t exist in all sets  make set name[0~20] to random character.  function *minElement*  if set a is NULL and set b is not NULL, return set b’s address.  else if set a is not NULL and set b is NULL, return set a’s address.  else if set a is NULL and set b is NULL, return NULL.  else, return set whose data is smaller. |

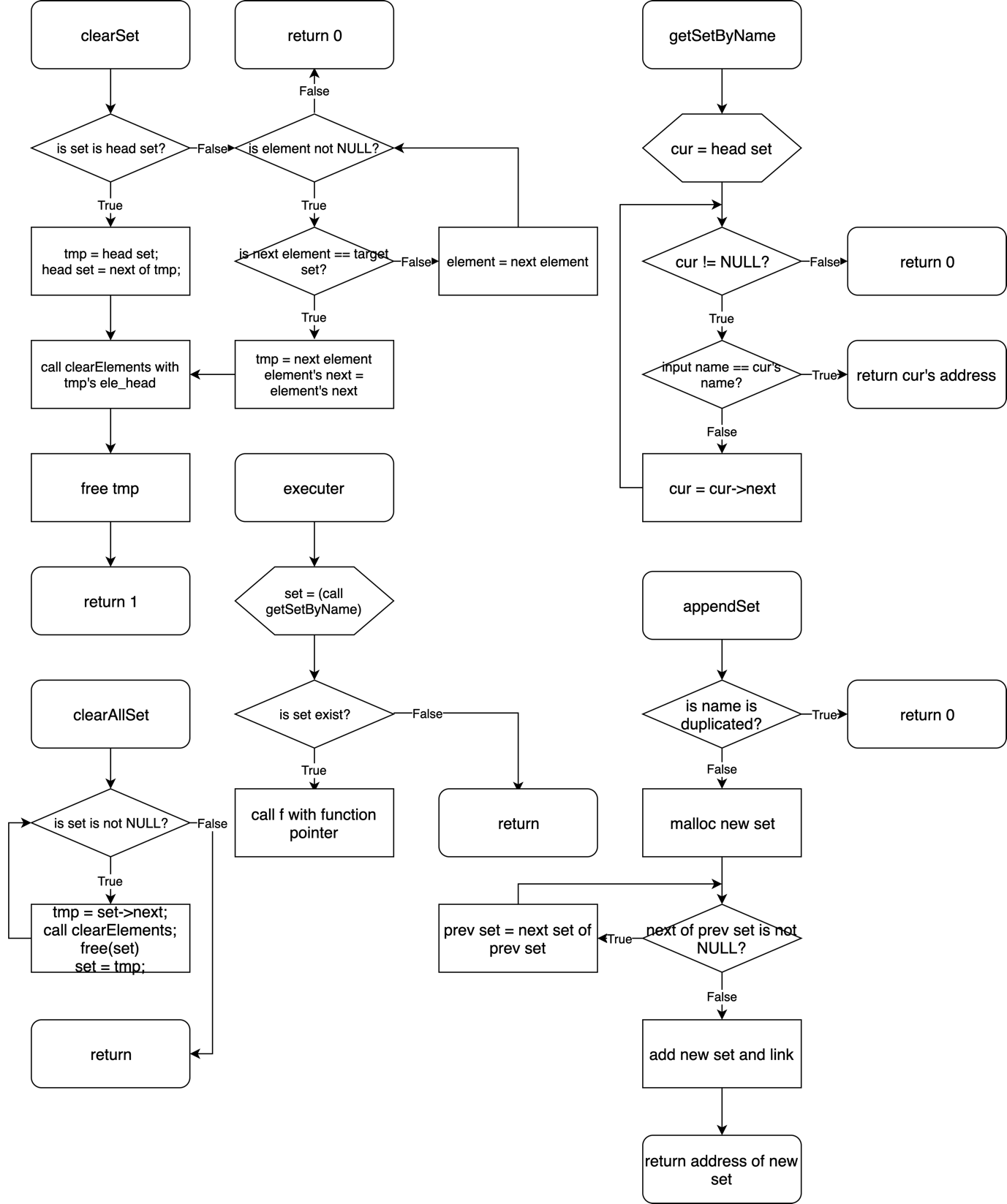
|  |  |
| --- | --- |
| Pseudo-code for ASSN5 <set.h> | |
| 1  2  3  4  5  6  7  8  9 | #pragma once  // #pragma once는 set.h가 여러 곳에서 include 될 때 여러번 빌드되고 포함되어 컴파일 에러가 일어나지 않도록 한번만 빌드하라고 지시하는 것이다.  set custom data type, “Element”, which contain data and next Element’s address.  set custom data type, “Set”, which contain name, size, fisrt element’s address, next set’s address.  write function prototypes. |

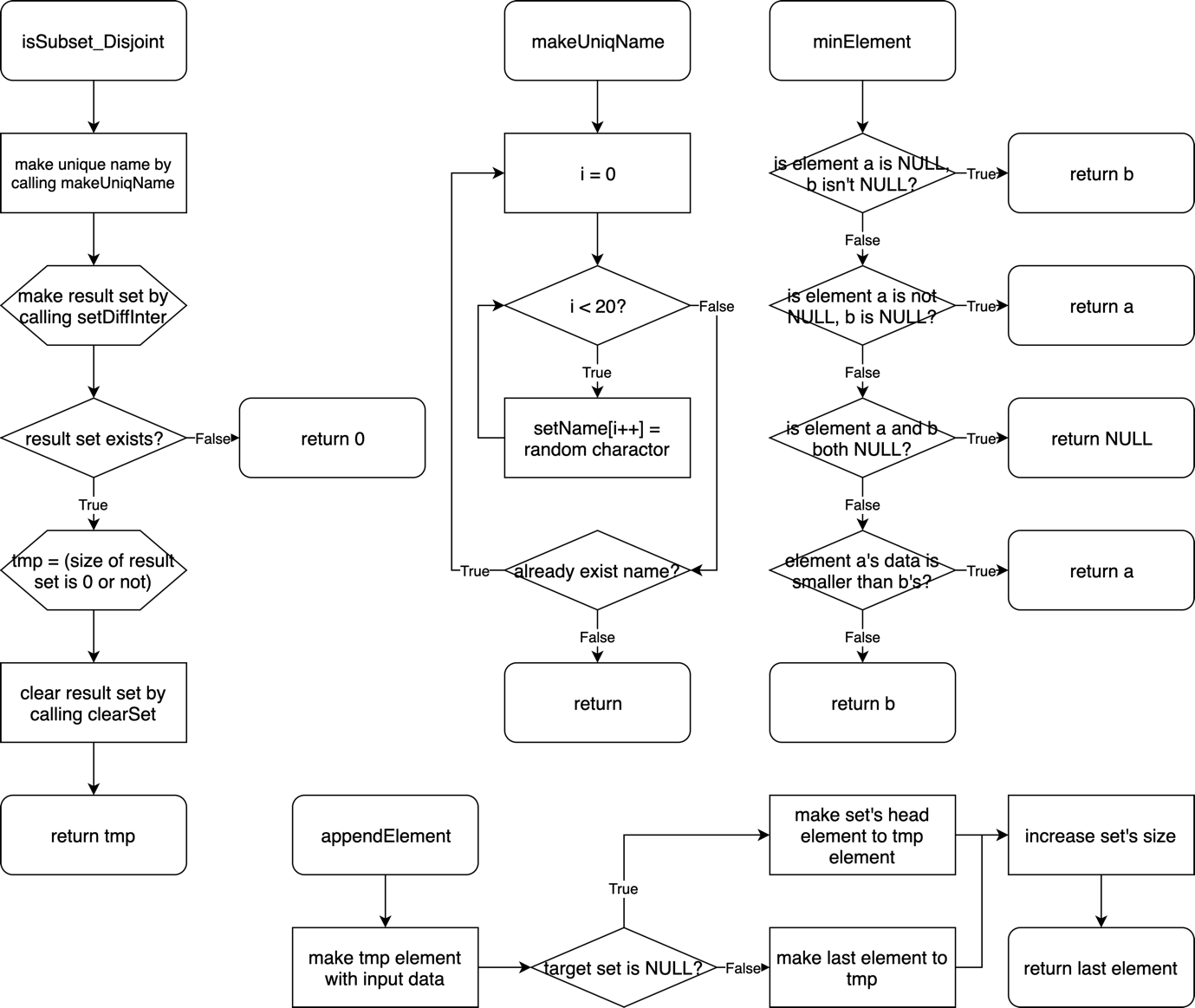
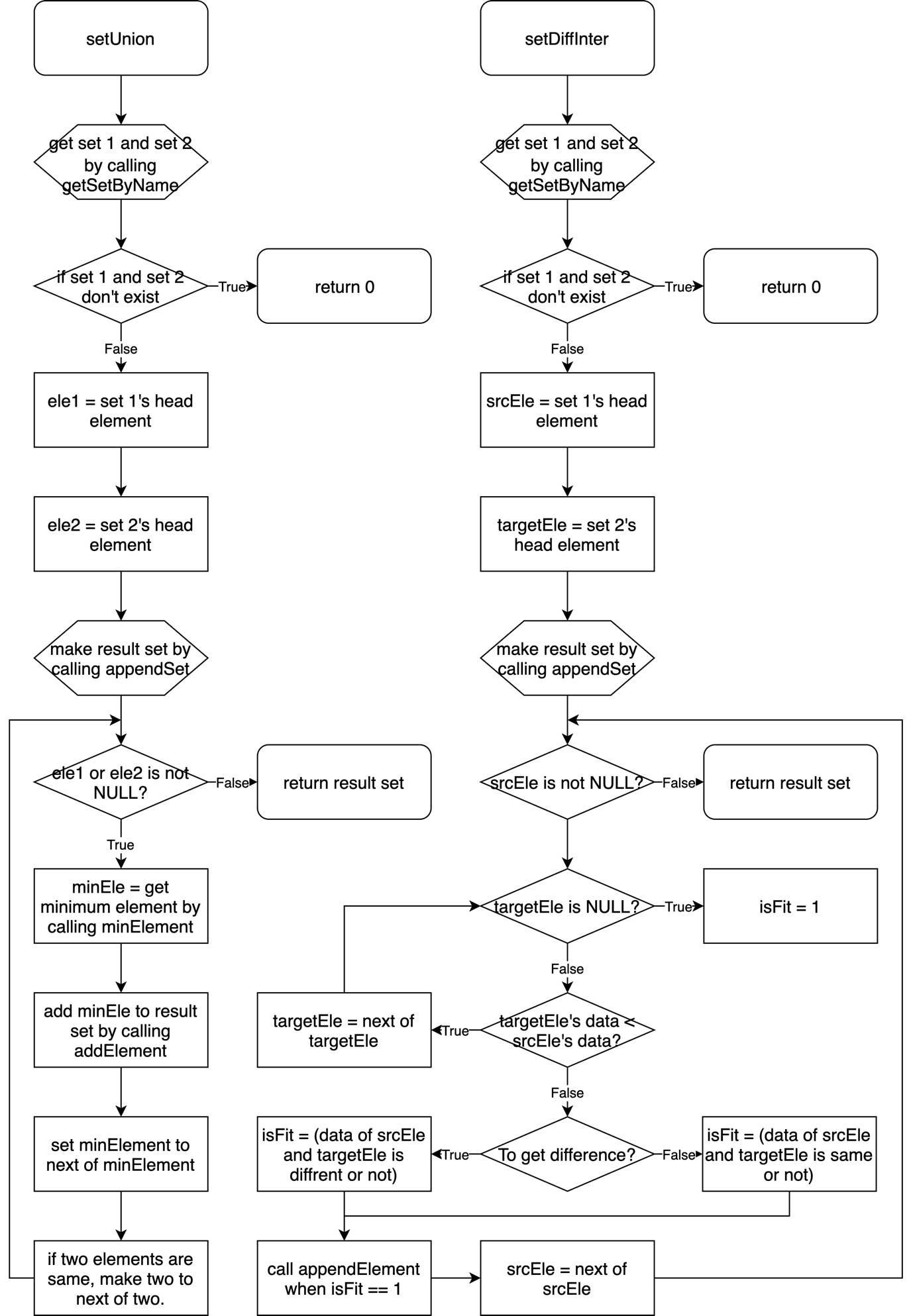
|  |  |
| --- | --- |
| Pseudo-code for ASSN5 <assn5.c> | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | // 프로그램에 필요한 변수들은 미리 선언한 것으로 가정하자.  include essential headers <stdio.h>, <stdlib.h>, <string.h>  include custom header “set.h”  function *main*  open file with **argv**[1]  if file is null, set file pointer to **stdin**  loop  if file pointer is stdin, print “>> “  if input is “set”, call *addSet*  if input is “show”,  if next letter is ‘ ‘ (space), input set name  else copy “” to set name  call *showSet* with set name  if input is “is\_element”, get set name, target and call executer with those and *isElement.*  if input is “add”, get set name and target, call executer with those and *addElement*  if input is “pop”, get set name and target, call executer with those and *popElement*  if input is “clear”, get set name and call *clearSetIn* with it.  if input is “union”, get name of set 1, set 2, result set and call *setUnion* with it.  if input is “intersection”, get name of set 1, set 2, result set and call *setDiffInter* with it.  if input is “difference”, get name of set 1, set 2, result set and call *setDiffInter* with it.  if input is “is\_disjoint”, get name of set 1, set 2 and call *isDisjoint* with it.  if input is “is\_subset”, get name of set 1, set 2 and call *isSubset* with it.  if input is “quit”  call *clearAllSet* and break  clean input buffer (remove EOF, \n) and copy “” to existing input string.  if file is EOF, close file stream and set file pointer to stdin. |

위의 의사 코드를 flowchart를 이용해서 나타내면 아래와 같다. (다음 페이지)

****

****

****

****

**3. 프로그램 구조 및 설명**

**a) 메인 메뉴**

* 메인 메뉴는 기본적으로 무한 루프에 갇힌 구조이다.
* argv[1]로 파일 이름을 받아서 파일을 열고, 해당 파일이 존재하는 파일이 아니면 파일 포인터를 stdin으로 세팅한다.
* 이후, 파일이나 표준 입력을 통해 명령어를 입력 받아 적절한 명령어를 수행한다.
* 단, show 명령어 같은 경우는 특별하게 처리한다. 입력된 명령어가 show일 경우에, 그 뒤의 1글자를 받아서 만약 그것이 ‘ ‘, 즉 띄어쓰기이면 세트 이름을 입력 받아 *showSet*을 호출하여 해당 세트 만을 출력시키고, 아니면 모든 세트가 출력될 수 있도록 한다.
* 명령어 수행이 끝난 후에는 while과 fgetc를 이용해 입력 버퍼에 남아있는 띄어쓰기나 개행 문자를 모두 제거해주고, 입력 받은 명령어 string을 초기화 해준다.
* 만약, 파일 포인터가 stdin가 아닌데 파일의 끝에 다다랐다면, 파일 포인터를 닫고 stdin으로 전환한다.

**b) 집합 생성: set**

* 집합 생성은 함수 *addSet*에 의해 구현된다. *addSet*에서는 우선 세트의 이름과 그 사이즈를 입력 받고, 입력 받은 사이즈만큼 데이터를 입력받는다. 함수 *addElement*를 이 입력 받은 데이터와 함께 호출하여 해당 set에 더한다. 함수 *addElement*는 **e) 원소 추가: add**에서 설명할 것이다.

**c) 집합 원소 출력: show**

* 집합 원소 출력은 함수 *showSet*에 의해 구현된다. *showSet*은 모든 세트를 순환하면서, 입력 받은 출력을 원하는 세트의 이름이 “” 이라면 모든 세트에 대해, 입력 받은 출력을 원하는 세트의 이름이 정해져 있다면 그 해당 세트에 대해, 함수 *showElements*를 호출하여 그 원소들을 출력한다.
* 함수 *showElements*는 입력받는 세트에 대해서, 그 세트의 이름을 출력하고 그 세트의 모든 원소를 출력하는 함수이다.

**d) 멤버 확인: is\_element**

* 멤버 확인은 함수 *isElement*에 의해 실현된다. *isElement*는 주어진 세트의 모든 원소를 순환하면서 만약 주어진 타겟과 같은 값이 있다면 1을, 아니면 0을 반환한다.

**e) 원소 추가: add**

* 원소 추가는 함수 *addElement*에 의해 실현된다. *addElement*는 원소를 탐색하여 오름차순에 부합하도록 주어진 데이터를 삽입하는 목적을 가지고 있다.
* 만약, 주어진 세트가 공집합이거나 세트의 헤드보다 데이터가 작다면, 그 세트의 원소 헤드에 주어진 데이터를 삽입한다. 아니라면, 주어진 데이터 보다는 크고, 그 다음 원소가 주어진 데이터보다는 작은 위치를 찾아서 그 자리에 삽입한다. 그 후, 세트의 크기를 증가시킨다.

**f) 원소 삭제: pop**

* 원소 삭제는 함수 *popElement*에 의해 구현된다. *popElement*는 원소를 탐색하여 주어진 데이터와 같은 원소가 있다면 그를 링크드 리스트에서 삭제하는 목적을 가지고 있다.
* 만약 삭제하고자 하는 데이터가 세트의 헤드 원소라면, 두번째 원소를 헤드 원소로 하고 기존 헤드 원소는 삭제한다. 아니면, 주어진 데이터와 같을 때 까지 원소를 탐색하여, 같은 원소가 발견된다면 그 원소를 삭제하고 그 앞 뒤 원소를 적절하게 연결시킨다.

**g) 집합 삭제: clear**

* 집합 삭제는 함수 *clearSet*에 의해 구현된다. *clearSet*는 주어진 세트의 원소들을 삭제하고 세트를 프리 시키는데 목적이 있다.
* 만약, 삭제하고자 하는 세트가 헤드 세트라면, 두번째 세트를 헤드 세트로 삼고, 기존 세트는 함수 *clearElements*를 호출하여 원소들를 삭제 시킨 후 세트를 free 시킨다. 아니라면, 세트를 찾은 후, 그 세트를 *clearElements*로 원소들을 삭제 시킨 후 세트를 free 시킨다. 그 전 후의 세트를 적절하게 연결시킨다.
* 함수 *clearElements*에서는 주어진 함수의 모든 원소를 free 시킨다.

**h) 합집합: union**

* 합집합은 함수 *setUnion*에 의해 실현된다. 함수 *setUnion*은 입력 받은 두 세트의 합집합을 입력 받은 이름의 세트로 생성하는 역할을 한다.
* *setUnion*의 특징은 오름차순을 이용하여 합집합을 구한다는 것이다. 이 함수에서, 우선 입력 받은 두 세트의 이름의 세트를 찾은 후, 요소 1과 2에 각 세트의 헤드 요소를 저장한다. 이때, 요소 1과 2가 둘 다 NULL이 될 때 까지, 요소 1과 2 중 작은 요소를 함수 *appendElement*를 통해 결과 세트에 추가하고, 작았던 요소를 그 다음 요소로 세팅 한다.
* 위와 같은 방법을 이용하면 오름차순을 이용하여 요소들을 정렬하면서 합집합을 생성할 수 있다.
* 작은 요소를 찾는데에는 함수 *minElement*를 이용한다. 이 함수에서는 둘 중 하나의 값이 NULL일 때, NULL이 아닌 요소를 리턴 하고, 둘 다 NULL일 경우 NULL을 리턴한다. 이 경우에 모두 해당하지 않을 경우, 데이터를 비교하여 작은 데이터를 가지고 있는 요소를 리턴한다.
* 함수 *appendElement*는 현재 가장 마지막 요소와 데이터를 입력받아서, 가장 마지막 요소의 다음에 입력받은 데이터를 가지는 요소를 추가하는 역할을 수행한다.

**i) 차집합, 교집합: difference, intersection**

* 차집합과 교집합을 구하는 알고리즘을 하나의 함수로 구현할 수 있을 정도로 거의 동일하게 구현하였기 때문에, 함수 *setDiffInter*로 이 둘을 구현하였다.
* *setDiffInter*의 특징은 오름차순을 이용하여 차집합이나 교집합을 구할 수 있다는 것이다. 이 함수에서, 우선 입력 받은 두 문자열을 이름으로 가지는 세트들을 구하고, 요소 1과 2에 각 세트의 헤드 요소를 저장한다. 이때, 첫번째 세트를 기준으로 생각할 것이다. 첫번째 세트의 요소를 읽어 나가는 동안, 지금 첫번째 세트의 요소보다 첫번째로 같거나 큰 두번째 세트의 요소를 찾는다. 현재 구하고자 하는 것이 차집합이라면 이 두 요소가 다를 때, 현재 구하고자 하는 것이 교집합이라면, 이 두 요소가 같을 때에 결과 세트에 이 요소를 추가한다.
* 첫번째 세트의 요소를 읽어나가는 하나의 반복 안에서, 두번째 세트를 요소를 읽을 때 헤드 요소부터 읽지 않고, 마지막으로 읽었던 요소부터 시작하므로, 오름차순의 특징을 이용한 것을 볼 수 있다.

**j) 포함, 서로소 여부: is\_subset, is\_disjoint**

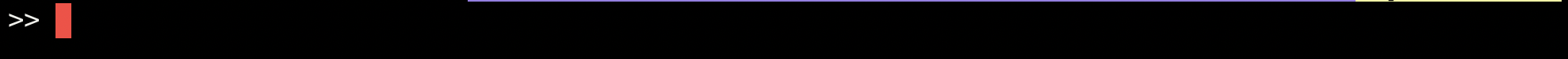
* **i) 차집합, 교집합**과 마찬가지로, 포함, 서로소 여부를 구하는 알고리즘이 거의 동일하기에 함수 *isSubset\_Disjoint*로 이 둘을 구현하였다.
* 이 함수에서는 우선 세트에 존재하지 않는 유일한 세트 이름을 생성한다. 이 유일한 세트 이름으로, 결과 세트에 함수 *setDiffInter*의 결과를 저장한다. 포함 여부를 알아야 할 때는 함수 *setDiffInter*를 차집합을 구하는 모드로, 서로소를 알아야 할 때에는 함수 *setDiffInter*를 교집합을 구하는 모드로 실행시킨다. 이 결과 세트의 원소 크기가 0이라면, *isSubset\_Disjoint*는 1을, 아니면 0을 반환한다.
* 유일한 세트 이름을 생성하는 것은 함수 *makeUniqName*으로 실현된다. 이 함수는 지금까지 존재하지 않는 세트 이름이 될 때 까지 21자리의 문자 배열을 생성한다.

**h) 종료: quit**

* 프로그램을 종료하기 전에 동적으로 할당한 메모리들을 모두 free 해 주며, 이는 함수 *clearAllSet*을 통해 실현된다.
* 함수 *clearAllSet*에서는 모든 세트를 순회하며 함수 *clearElements*를 호출하여 요소들을 free한 후, 그 세트도 free 한다.

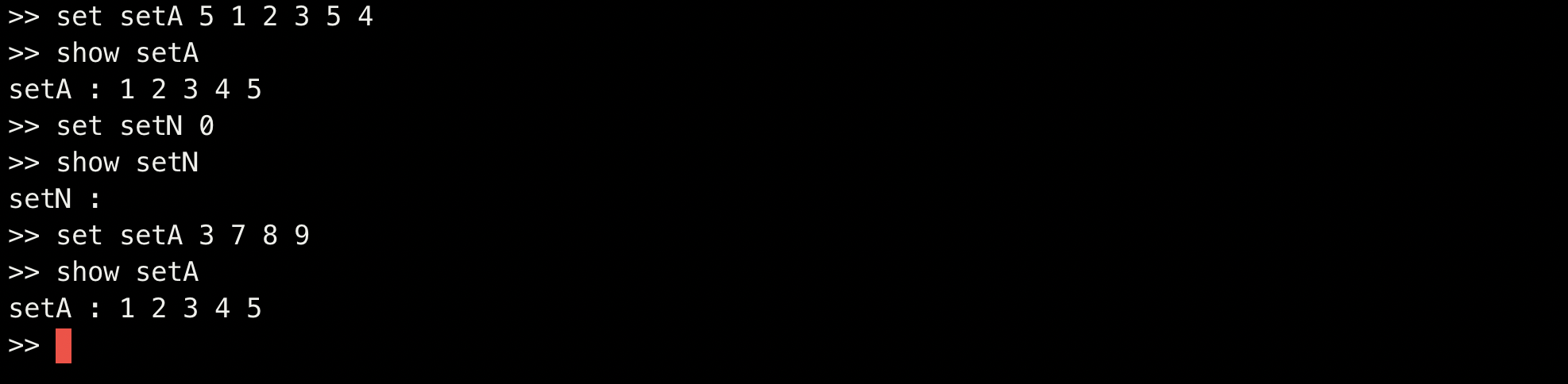
**4. 프로그램 실행 방법 및 예제**

**a) 초기화면**



프로그램을 실행하면 위와 같이 사용자의 명령어 입력을 대기하는 “>> “ 표시가 출력된다.

**b) 집합 생성: set**



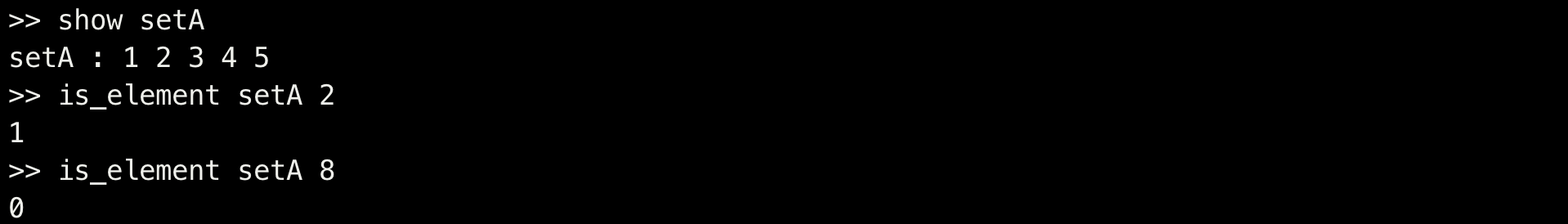
set 함수를 이용하여 오름차순으로 원소를 저장하며, 중복된 경우는 무시하고, 공집합도 정상적으로 저장되는 모습을 볼 수 있다.

**c) 집합 원소 출력: show**



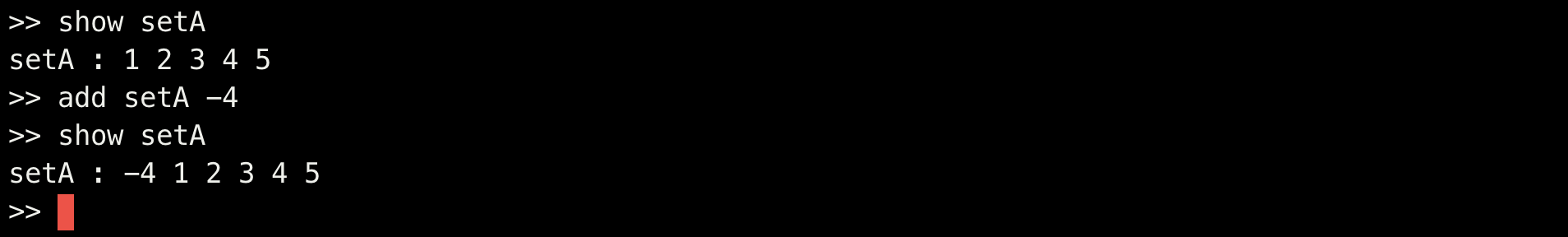
show를 입력하면 모든 세트를 보여주고, 없는 세트에 대해서는 아무것도 보여주지 않는 모습을 볼 수 있다.

**d) 멤버 확인: is\_element**



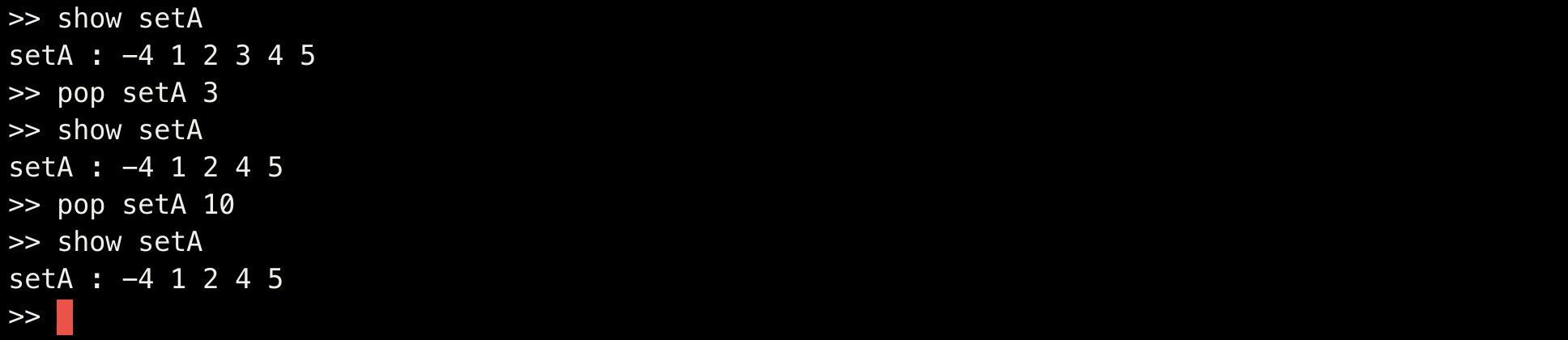
setA에 있는 원소인 2에 대해서는 1을, 없는 원소인 8에 대해서는 0을 정상적으로 출력하는 모습을 볼 수 있다.

**e) 원소 추가: add**



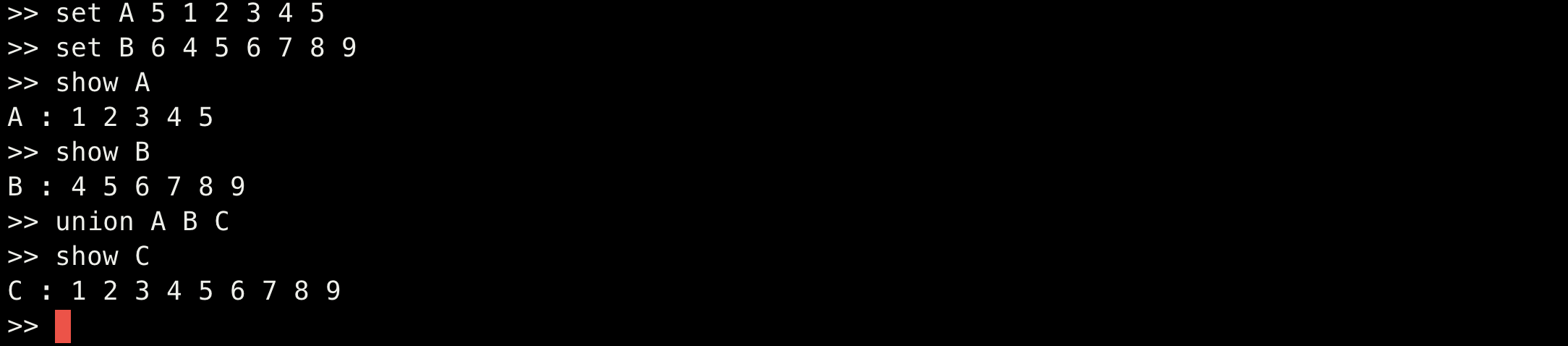
특정 집합에 오름차순으로 원소가 잘 추가되는 모습을 볼 수 있다.

**f) 원소 삭제: pop**



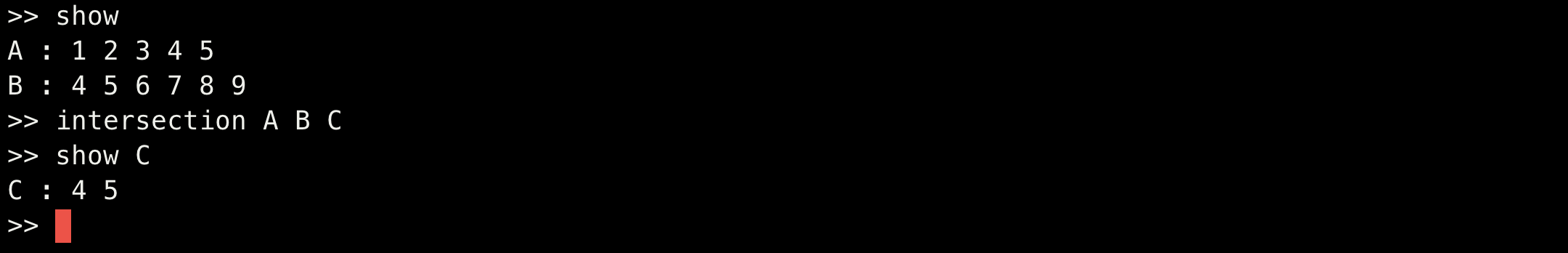
특정 집합에 삭제하고자 하는 원소가 있을 때에 그 원소를 삭제하는 모습을 볼 수 있다.

**g) 합집합: union**

****

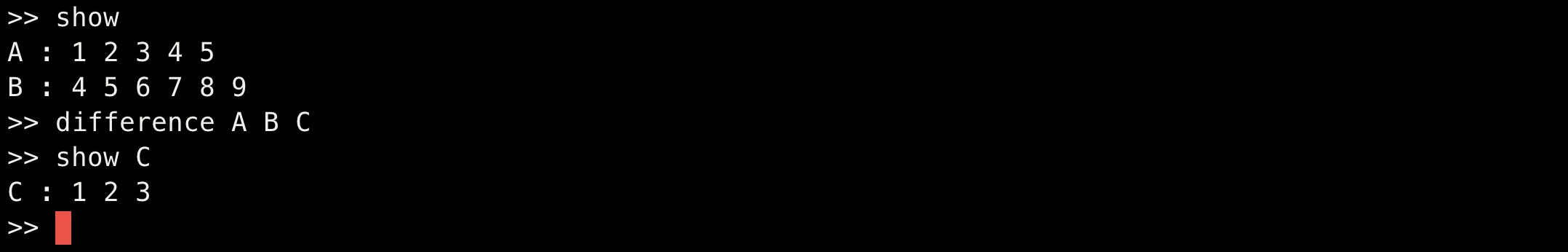
집합 A와 B의 합집합을 구하여 새로운 집합 C를 만드는 모습을 볼 수 있다.

**h) 교집합: intersection**

****

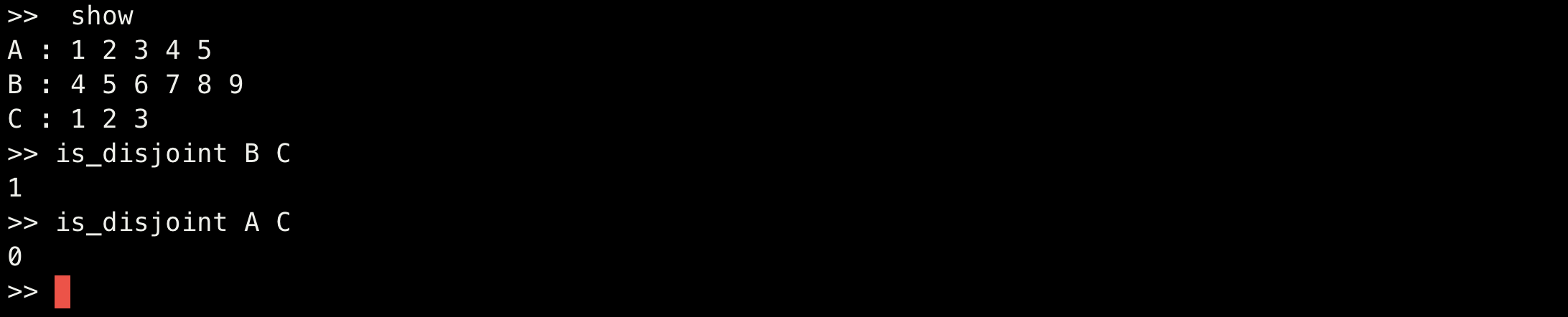
집합 A와 B의 교집합을 구하여 새로운 집합 C를 만드는 모습을 볼 수 있다.

**i) 차집합: difference**

****

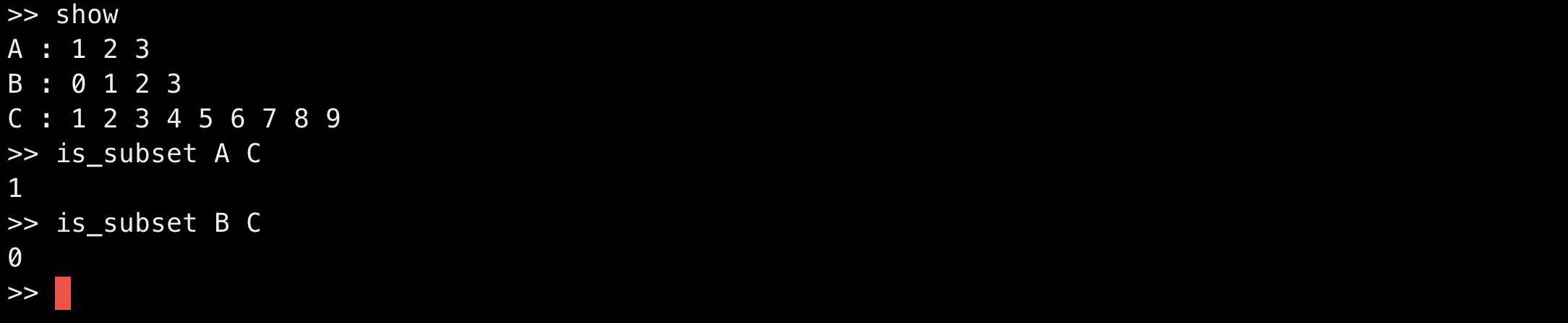
집합 A에 속하지만 B에 속하지 않는 원소들로 이루어진 새로운 집합 C를 만드는 모습을 볼 수 있다.

**j) 서로소 여부: is\_disjoint**

****

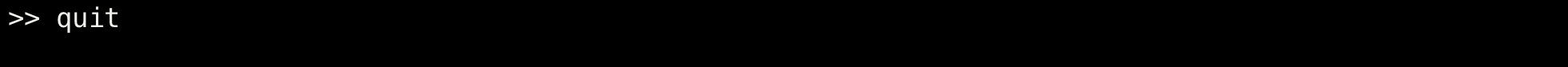
두 집합이 서로소이면 1, 아니면 0을 출력하는 모습을 볼 수 있다.

**k) 포함 여부: is\_subset**

****

두 입력된 집합 중, 앞에 있는 집합이 뒤의 집합에 포함되는 관계이면 1, 아니면 0을 출력하는 모습을 볼 수 있다.

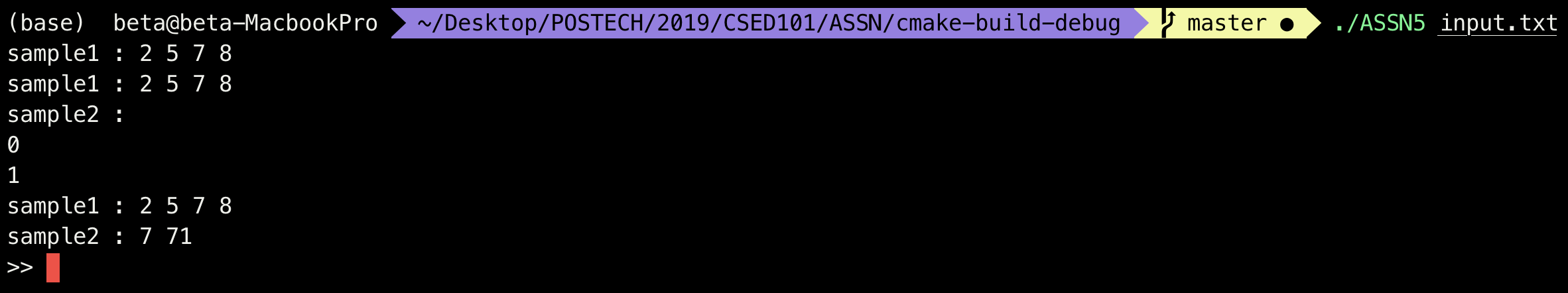
**l) 프로그램 종료: quit**



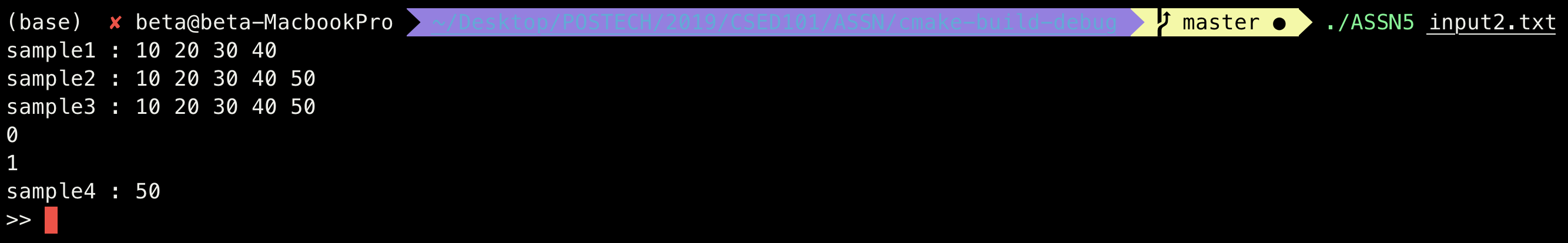
정상적으로 종료되는 모습을 볼 수 있다.

**m) 파일명과 함께 실행**

|  |  |
| --- | --- |
| input.txt | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | set sample1 3 5 3 2  add sample1 7  add sample1 8  pop sample1 3  show sample1  set sample2 0  show  add sample2 7  is\_element sample2 71  add sample2 7  add sample2 71  is\_element sample2 71  show |



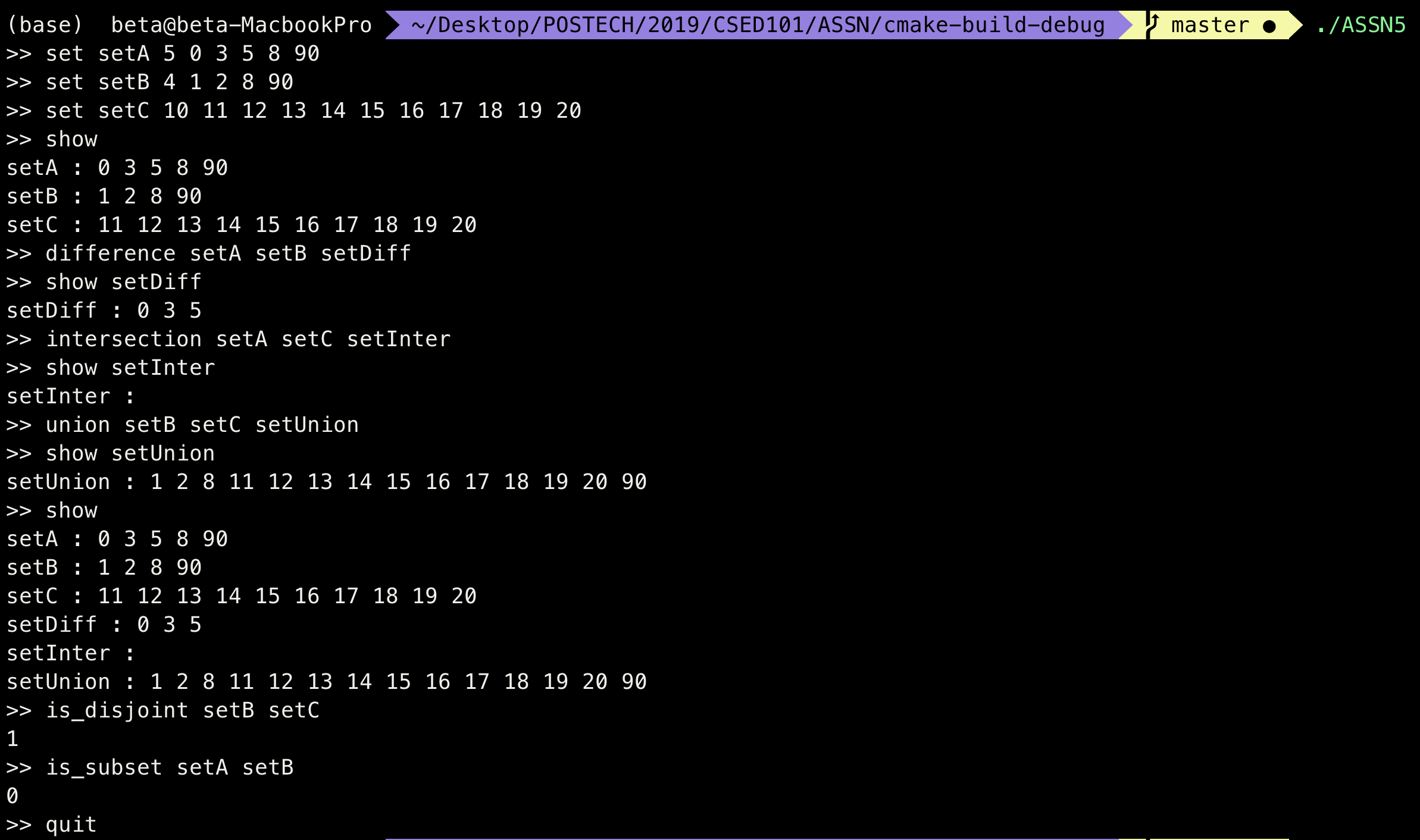
|  |  |
| --- | --- |
| input2.txt | |
| 1  2  3  4  5  6  7  8  9 | set sample1 4 10 20 30 40  set sample2 5 10 20 30 40 50  show  union sample2 sample1 sample3  show sample3  is\_subset sample2 sample1  is\_subset sample1 sample2  difference sample3 sample1 sample4  show sample4 |



input.txt, input2.txt에 대해 정상적으로 명령을 수행한 후, 사용자의 입력을 대기하는 모습을 볼 수 있다.

**n) 파일명 없이 실행**





파일명 없이 실행한 경우, 사용자의 입력을 정상적으로 수행하는 모습을 볼 수 있다.

**5. 토론**

* 헤더 파일 <set.h>를 작성할 때, “**#pragma once**” 혹은 “**#ifndef USER\_H #define USER\_H ~ #endif**”를 이용해야 하는데, 이는 헤더 파일이 중복으로 로드 되고 컴파일 되어 생기는 오류를 막기 위해 해당 헤더 파일을 한번만 로드하고 컴파일 하도록 위와 같은 문구로 지시하는 것이다.
* 함수 *getSetByName*을 호출하여 세트의 주소를 구하고, 그 세트를 매개변수로 다른 함수를 호출하는 구조가 여러번 쓰이는 점을 이용하여, 함수 포인터를 이용해 해당 구조를 여러 함수에 걸쳐 사용할 수 있는 함수는 *executer*를 만들어서 사용하였다.

**6. 결론**

* 본 과제를 통해 구조체를 포인터와 함께 사용할 수 있는 능력을 키울 수 있었고, 이를 바탕으로 연결 리스트의 사용법을 익혔다.
* 본 과제를 통해 헤더 파일을 이용하여 여러 파일로 분할하여 코드를 작성하는 방법을 알 수 있었다.
* 연결 리스트가 탐색하는데 배열보다 비교적 비효율적임을 알았다.

**7. 개선방향**

* 주어진 문제에서 주어진 구조의 연결 리스트는 각 노드가 다음 주소 값만을 가지고 있는 단순 연결 리스트였는데, 이전 주소 값도 가지는 이중 연결 리스트로 구현했다면 탐색할 때 더 효율적이었을 것이다.
* 현재 코드의 구성에서, 가장 첫번째 세트를 리스트의 정보를 담는 역할로 사용하였는데, 여기에 연결 리스트의 가장 마지막 원소도 저장하였었다면 더 편리하게 코드를 작성할 수 있었을 것이다.