**Pintos Project 0-2 : Pintos Data Structure**

|  |  |
| --- | --- |
| 과목 명 : | [CSE4070] 운영체제 2반 |
| 담당 교수 : | 김영재 교수님 |
| 학번 : | 20141284 |
| 이름 : | 이기현 |

**프로젝트 제목 : pintos project 0-2 Pintos Data Structure**

**제출일 : 2019년 10월 6일**

1. **List**

* void list\_init(struct list \*)

v Parameter : 초기화 할 리스트

v Return : 없음

v Function : list를 head와 tail이 연결된 초기 상태로 만들어준다.

* struct list\_elem \*list\_front(struct list \*)

v Parameter : 항목을 찾을 리스트

v Return : 리스트의 항목 중 첫 번째 list\_elem의 포인터

v Function : 리스트의 항목 중 첫 번째 항목을 찾아준다.

* struct list\_elem \*list\_back(struct list \*)

v Parameter : 항목을 찾을 리스트

v Return : 리스트의 항목 중 마지막 list\_elem의 포인터

v Function : 리스트의 항목 중 마지막 항목을 찾아준다.

* void list\_push\_front(struct list \*, struct list\_elem \*)

v Parameter : 항목을 집어넣을 리스트와 넣을 elem

v Return : 없음

v Function : 리스트의 가장 맨 앞에 인자로 받은 elem을 넣고 연결해준다.

* void list\_push\_back (struct list \*, struct list\_elem \*)

v Parameter : 항목을 집어넣을 리스트와 넣을 elem

v Return : 없음

v Function : 리스트의 가장 맨 앞에 인자로 받은 elem을 넣고 연결해준다.

* struct list\_elem \*list\_pop\_front(struct list \*)

v Parameter : 항목을 뽑아 낼 리스트

v Return : 리스트에서 뽑아 낸 가장 앞 항목의 포인터

v Function : 리스트에서 가장 앞 항목을 뽑아내고 그 뒤에 있던 항목과 head를 연결해준다.

* struct list\_elem \*list\_pop\_back(struct list \*)

v Parameter : 항목을 뽑아 낼 리스트

v Return : 리스트에서 뽑아 낸 가장 마지막 항목의 포인터

v Function : 리스트에서 가장 마지막 항목을 뽑아내고 그 앞에 있던 항목과 tail을 연결해준다.

* struct list\_elem \*list\_begin(struct list \*)

v Parameter : 시작 부분을 찾을 리스트

v Return : 리스트의 항목 중 가장 앞 항목의 포인터

v Function : 리스트의 head 다음, 즉 가장 앞 항목을 찾아준다.

* struct list\_elem \*list\_next (struct list\_elem \*)

v Parameter : 다음 항목을 찾을 앞 항목

v Return : 인자로 받은 항목의 다음 항목의 포인터

v Function : 인자로 받은 항목의 다음 항목을 찾아준다.

* struct list\_elem \*list\_end (struct list \*)

v Parameter : 마지막 부분을 찾을 리스트

v Return : 리스트의 마지막, 즉 tail 의 포인터

v Function : 리스트의 가장 마지막인 tail 부분을 찾아준다.

* void list\_insert(struct list\_elem \*, struct list\_elem \*)

v Parameter : 항목을 넣을 위치에 있는 항목과 새로 넣을 항목

v Return : 없음

v Function : 두 번째로 받은 새로운 항목을 첫 번째 인자로 받은 항목의 앞쪽에 넣고 연결해준다.

* void list\_insert\_ordered(struct list \*, struct list\_elem \*, list\_less\_func \*, void \*aux)

v Parameter : 항목을 넣을 리스트, 넣을 항목, 비교 함수

v Return : 없음

v Function : 비교함수를 이용해 sorting되어 있는 리스트 내부에 들어갈 자리를 찾아서 넣어준다.

* struct list\_elem \*list\_remove(struct list\_elem \*)

v Parameter : 리스트에서 빼낼 항목

v Return : 빼낼 항목의 다음 항목(빼낸 항목이 있던 자리에 오게 되는 항목)

v Function : 인자로 받은 항목의 앞, 뒤 항목을 연결해준다.

* struct list\_elem \*list\_max(struct list \*, list\_less\_func \*, void \*aux)

v Parameter : 리스트, 비교함수

v Return : 리스트에서 가장 큰 data를 갖는 elem의 포인터

v Function : 비교함수를 이용해 리스트 내부에서 data를 최대로 갖는 항목을 찾아준다.

* struct list\_elem \*list\_min(struct list \*, list\_less\_func \*, void \*aux)

v Parameter : 리스트, 비교함수

v Return : 리스트에서 가장 큰 data를 갖는 elem의 포인터

v Function : 비교함수를 이용해 리스트 내부에서 data를 최소로 갖는 항목을 찾아준다.

* size\_t list\_size(struct list \*)

v Parameter : 리스트

v Return : 리스트의 사이즈

v Function : 인자로 받은 리스트 내부의 데이터 개수 즉, 사이즈를 찾아준다.

* bool list\_empty(struct list \*)

v Parameter : 리스트

v Return : 리스트가 비었다면 true, 항목이 하나라도 있다면 false

v Function : 리스트에 항목이 있는지 없는지 판별해준다.

* void list\_reverse(struct list \*)

v Parameter : 리스트

v Return : 없음

v Function : 리스트 항목의 순서를 반대로 뒤집어준다.

* void list\_sort(struct list \*, list\_less\_func \*, void \*aux)

v Parameter : 리스트, 비교함수

v Return : 없음

v Function : 비교함수를 이용해 리스트를 오름차순으로 정렬한다.

* void list\_splice(struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last)

v Parameter : 리스트를 이어 붙일 부분의 항목, 이어 붙일 리스트의 시작과 끝 항목

v Return : 없음

v Function : 첫 번째 인자의 앞 부분에 first ~ last 부분을 넣고 연결관계를 연결해준다.

* void list\_unique(struct list \*list, struct list \*duplicates, list\_less\_func \*, void \*aux)

v Parameter : 정리할 리스트, 정리되고 남은 항목을 넣을 리스트, 비교함수

v Return : 없음

v Function : 첫번째로 받은 list에 모든 값을 unique하도록 하나씩만 남기고 뺀 값을 duplicates에 넣어준다.

1. **Hash Table**

* bool hash\_init(struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux)

v Parameter : 초기화할 해시, 해시 함수, 비교 함수

v Return : 초기화 성공하면 true, 못하면 false

v Function : 해시를 초기화 해준다

* void hash\_first(struct hash\_iterator \*, struct hash \*)

v Parameter : 해시의 반복자, 해시

v Return : 없음

v Function : 두 번째 인자로 받은 해시를 순회하기 위해 첫번째 해시 값을 인자로 받은 반복자가 가리키도록 한다.

* struct hash\_elem \*hash\_next(struct hash\_iterator \*)

v Parameter : 해시의 반복자

v Return : 인자로 받은 반복자가 가리키던 값의 다음 항목

v Function : 반복자를 가리키던 항목의 다음 항목을 가리키도록 해주고 그 항목을 리턴해준다.

* struct hash\_elem \*hash\_insert(struct hash \*, struct hash\_elem \*)

v Parameter : 해시, 새로 넣을 elem

v Return : 기존에 elem이 존재했다면 그 항목을, 없었다면 NULL을 리턴

v Function : 해시에 인자로 받은 elem을 새롭게 넣어준다.

* void hash\_apply(struct hash \*, hash\_action\_func \*)

v Parameter : 해시, 수행 함수

v Return : 없음

v Function : 해시내 모든 값에 대해 인자로 받은 action\_func을 수행한다.

* struct hash\_elem \*hash\_delete(struct hash \*, struct hash\_elem \*)

v Parameter : 해시, 지울 elem

v Return : 해시 내부에 elem이 없다면 NULL, 있다면 지운 elem값을 리턴

v Function : 해시 내부에서 elem 을 찾아 지워준다.

* bool hash\_empty(struct hash \*)

v Parameter : 해시

v Return : 해시가 비어있다면 true, 아니라면 false

v Function : 해시가 비었는지 판별해준다.

* struct hash\_elem \*hash\_find(struct hash \*, struct hash\_elem \*)

v Parameter : 해시, 찾을 elem

v Return : 해시 내부에 elem이 있다면 elem을, 없다면 NULL을 리턴

v Function : 해시 내부에 elem 이 존재하는지 찾아준다.

* struct hash\_elem \*hash\_replace(struct hash \*, struct hash\_elem \*)

v Parameter : 해시, hash\_elem

v Return : 해시 내부에 elem이 있었다면 elem을, 없었다면 NULL을 리턴

v Function : 해시 내부에 elem을 새로 넣는다 기존에 있었다면 새로 교체해준다.

* size\_t hash\_size(struct hash \*)

v Parameter : 해시

v Return : 해시의 사이즈

v Function : 해시의 사이즈를 구해 리턴한다.

* void hash\_clear(struct hash \*, hash\_action\_func \*)

v Parameter : 해시, 수행함수

v Return : 없음

v Function : 해시 내부의 모든 값을 지워준다.

1. **Bitmap**

* struct bitmap \*bitmap\_create(size\_t bit\_cnt)

v Parameter : 비트 개수

v Return : 생성된 비트맵

v Function : 비트 개수를 인자로 받아 그 개수만큼 크기의 비트맵을 생성해준다.

* void bitmap\_mark(struct bitmap \*, size\_t idx)

v Parameter : 비트맵, 인덱스

v Return : 없음.

v Function : 인자로 받은 인덱스 위치의 비트 값을 1로 set해준다.

* bool bitmap\_all(const struct bitmap \*, size\_t start, size\_t cnt)

v Parameter : 비트맵, 시작 인덱스, 개수

v Return : start부터 cnt개수의 비트가 모두 1이라면 true, 아니라면 false

v Function : 비트맵 내부의 start ~ start+cnt -1의 비트가 모두 1인지 체크해준다.

* bool bitmap\_any(const struct bitmap \*, size\_t start, size\_t cnt)

v Parameter : 비트맵, 시작 인덱스, 개수

v Return : start부터 cnt개수의 비트 중에 1개라도 1이라면 true, 아니라면 false

v Function : 비트맵 내부의 start ~ start+cnt -1의 비트 중에 1이 하나라도 있는지 체크해준다.

* bool bitmap\_contains(const struct bitmap \*, size\_t start, size\_t cnt, bool)

v Parameter : 비트맵, 시작 인덱스, 개수, Boolean flag

v Return : start부터 cnt개수의 비트 중에 flag가 true일 땐 1이, flag가 false일 땐 0이 존재하면 true, 없으면 false리턴

v Function : 비트맵 내부의 start ~ start + cnt- 1의 비트 중에 flag에 해당하는 값이 있는지 체크해준다.

* size\_t bitmap\_count(const struct bitmap \*, size\_t start, size\_t cnt, bool)

v Parameter : 비트맵, 시작 인덱스, 개수, Boolean flag

v Return : start부터 cnt개수의 비트 중에 flag가 true일 땐 1의 개수를, flag가 false일 땐, 0의 개수를 리턴

v Function : 비트맵 내부의 start ~ start + cnt- 1의 비트 중에 flag에 해당하는 값의 개수를 파악해준다.

* void bitmap\_dump(const struct bitmap \*)

v Parameter : 비트맵

v Return : 없음

v Function : 비트맵을 인자로 받아서 비트맵의 값을 16진수로 표현해준다.

* void bitmap\_flip(struct bitmap \*, size\_t idx)

v Parameter : 비트맵, 인덱스

v Return : 없음.

v Function : 인자로 받은 비트맵의 인덱스 번째 값을 true에서 false로, false에서 true로 바꿔준다.

* bool bitmap\_none(const struct bitmap \*, size\_t start, size\_t cnt)

v Parameter : 비트맵, 시작 인덱스, 개수

v Return : start부터 cnt개수의 비트 중에 1개라도 1이라면 false, 아니라면 true

v Function : 비트맵 내부의 start ~ start+cnt -1의 비트 중에 1이 하나라도 있는지 체크해준다.

* void bitmap\_reset(struct bitmap \*, size\_t idx)

v Parameter : 비트맵, 인덱스

v Return : 없음.

v Function : 인자로 받은 인덱스 위치의 비트 값을 0으로 reset해준다.

* size\_t bitmap\_scan\_and\_flip(struct bitmap \*, size\_t start, size\_t cnt, bool)

v Parameter : 비트맵, start인덱스, 개수, bool값

v Return : 아래에서 설명한 인덱스

v Function : 비트맵 내부의 start이후 연속으로 cnt개수만큼 bool값을 갖는 인덱스를 찾고 찾는데 성공하면 비트의 값을 뒤집어준다.

* size\_t bitmap\_scan(struct bitmap \*, size\_t start, size\_t cnt, bool)

v Parameter : 비트맵, start인덱스, 개수, bool값

v Return : 아래에서 설명한 인덱스

v Function : 비트맵 내부의 start이후 연속으로 cnt개수만큼 bool값을 갖는 인덱스를 찾는다

* void bitmap\_set\_all(struct bitmap \*, bool)

v Parameter : 비트맵, bool값

v Return : 없음

v Function : 비트맵의 모든 값을 bool값으로 설정해준다.

* void bitmap\_set(struct bitmap \*, size\_t idx, bool)

v Parameter : 비트맵, 인덱스, bool값

v Return : 없음.

v Function : 비트맵의 인덱스 위치에 있는 값을 bool값으로 변경해준다.

* void bitmap\_set\_multiple(struct bitmap \*, size\_t start, size\_t cnt, bool)

v Parameter : 비트맵, start 인덱스, 개수, bool값

v Return : 없음.

v Function : 비트맵의 start ~ start + cnt – 1 의 값들을 모두 bool값으로 바꿔준다.

* size\_t bitmap\_size(const struct bitmap \*)

v Parameter : 비트맵

v Return : 비트맵의 사이즈

v Function : 비트맵의 사이즈를 파악해준다.

* bool bitmap\_test(const struct bitmap \*, size\_t idx)

v Parameter : 비트맵, 인덱스

v Return : 비트맵의 인덱스 위치의 값이 1이면 true, 0이면 false

v Function : 비트맵의 특정 인덱스 값이 0인지 1인지 파악해준다.

1. **User defined Function**

* bool l\_less(const struct list\_elem\* a, const struct list\_elem\* b, void\* aux)

v Parameter : list\_elem 인자 2개

v Return : a의 데이터가 b의 데이터보다 작으면 true, 아니면 false

v Function : 정렬 등에 사용하기 위해 list 의 두 항목을 비교해준다.

* unsigned h\_func(const struct hash\_elem\* e, void\* aux)

v Parameter : hash\_elem 인자 1개

v Return : e에 대응되는 value값

v Function : 인자로 받은 e의 값을 key로 생각했을 때 대응될 value값을 찾아준다.

* bool h\_less(const struct hash\_elem\* a, const struct hash\_elem\* b, void\* aux)

v Parameter : hash\_elem 인자 2개

v Return : a의 데이터가 b의 데이터보다 작으면 true, 아니면 false

v Function : 해시테이블에서 값들을 관리하고 찾기 위한 해시 비교함수

* void h\_square(struct hash\_elem\* e, void\* aux)

v Parameter : hash\_elem 인자 1개

v Return : 없음.

v Function : hash\_elem의 데이터를 3제곱 해준다.

* void h\_triple(struct hash\_elem\* e, void\* aux)

v Parameter : hash\_elem 인자 1개

v Return : 없음.

v Function : hash\_elem의 데이터를 3제곱 해준다.

* void list\_swap(struct list\_elem\* a, struct list\_elem\* b)

v Parameter : list\_elem 인자 2개

v Return : 없음.

v Function : 리스트의 두 항목 a, b위치를 바꿔준다.

* void list\_shuffle(struct list\* list)

v Parameter : 리스트

v Return : 없음.

v Function : 리스트를 임의의 순서로 변경해준다.

* unsigned hash\_int\_2(int i)

v Parameter : 숫자

v Return : 인자로 받은 값에 대응되는 value

v Function : 인자에 대응될 value를 찾아준다.

* struct bitmap\* bitmap\_expand(struct bitmap\* bitmap, int size)

v Parameter : 비트맵, 확장할 사이즈

v Return : 확장된 비트맵

v Function : 비트맵을 size만큼 확장시키고 확장된 부분을 0으로 만들어 준다.