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

|  |  |
| --- | --- |
| 과목 명 : | 운영체제 |
| 담당 교수 : | 김영재 |
| 학번 : | 20151562 |
| 이름 : | 신은우 |

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

**제출일 : 2019. 10. 06**

본 프로젝트의 경우 OS의 수업에 사용 하는 Pinto에서 사용되는 자료구조 중 List, Hash Table, Bitmap에 대한 이해도를 높이는 것에 목적을 한다. 사용자가 프로그램을 실행시킨 후 적절한 명령어를 입력하면 프로그램은 그에 따라 자료구조를 조작하고 적절한 값을 출력한다.

본 프로그램은 Pintos의 구성 파일에 있는 라이브러리를 사용하여 자료구조를 조작하며 main.c와 main.h 파일에서 사용자의 입력을 받아 라이브러리를 사용하여 적절히 처리해주는 역할을 한다. 해당 자료구조에 대한 추가적인 함수 구현이 필요할 경우 해당 라이브러리 파일에 구현하지 않고 main.h, main.c에 구현을 하였다.

1. **List**

* list\_init
  + Prototype : void list\_init (struct list \*list)
  + Parameter : 초기화 하고 싶은 struct list
  + Return : 반환값 없음.
  + Function : double linked list를 선언한 후 이를 초기화 해주는 역할을 한다. Linked list의 head element와 tail element의 값을 NULL과 서로를 향하게 한다.
* list\_begin
  + prototype : struct list\_elem \*list\_begin (struct list \*list)
  + parameter : head의 다음값을 받고싶은 linked list의 주소값
  + return : 해당 linked list의 head의 다음 element 값을 반환한다.
  + function : 전달 받은 linked list의 head element의 다음 값을 반환한다. 만약 list가 비어있으면 tail element의 주소값을 반환한다.
* List\_end
  + Prototype : struct list\_elem \*list\_end (struct list \*list)
  + Parameter : tail값을 받고 싶은 linked list의 주소값을 전달한다.
  + Return : linked list의 tail 주고 값.
  + Function : linked list를 받아 해당 list의 tail을 반환한다.
* List\_next
  + Prototype : struct list\_elem \*list\_next (struct list\_elem \*elem)
  + Parameter : list element의 주소
  + Return : 전달 받은 element의 다음 주소값
  + Function : list element를 받아 다음 element의 주소값을 반환한다. 만약 전달받은 element가 tail일 경우 에러가 난다.
* list\_entry (매크로 함수)
  + prototype : list\_entry(LIST\_ELEM, STRUCT, MEMBER) STRUCT \*)
  + parameter : head와 tail이 아닌 list element의 주소값, 해당 list\_element를 멤버로 가지고 있는 구조체의 이름, 해당 구조체의 list\_element를 선언한 변수 명
  + return : 전달받은 list\_element를 가지고 있는 구조체 변수의 주소값 반환
  + function : list\_element의 주소값을 전달받아 해당 list\_element를 멤버로 가지고 있는 구조체의 주소값을 반환한다.
* List\_empty
  + Prototype : bool list\_empty (struct list \*list)
  + Parameter : empty인지 확인하고 싶은 linked list의 주소값
  + Return : 해당 list가 비어있으면 true, 아닐시 false 반환
  + Function : 전달받은 linked list가 비었으면 true를 아니면 false를 반환한다.
* List\_pop\_front
  + Prototype : struct list\_elem \* list\_pop\_front (struct list \*list)
  + Parameter : 첫번째 element를 pop해서 반환받을 linked list의 주소값
  + Ruetrn : 전달받은 linked list의 첫번째 element의 주소값
  + Function : 전달받은 linked list 첫번째 element를 해당 linked list에서 제거한 후 이를 반환한다.
* List\_insert
  + Prototype : void list\_insert (struct list\_elem \*before, struct list\_elem \*elem)
  + Parameter : linked list에 element를 삽입할 때 넣을 위치의 다음 element의 주소값과 넣을 element의 주소값.
  + Return : 없음
  + Function : before 매개변수의 바로 전 위치에 elem 매개변수를 삽입한다.
* List\_push\_front
  + Prototype : void list\_push\_front (struct list \*list, struct list\_elem \*elem)
  + Parameter : 새로운 element를 삽입할 linked list의 주소값과 새롭게 삽입할 element의 주소값
  + Return : 없음
  + Function : 주어진 linked list의 첫번째, 즉 head element의 다음에 element를 삽입한다.
* List\_push\_back
  + Prototype : void list\_push\_back (struct list \*list, struct list\_elem \*elem)
  + Parameter : 새로운 element를 삽입할 linked list의 주소값과 새롭게 삽입할 element의 주소값
  + Return : 없음
  + Function : 주어진 linked list의 가장 마지막 즉, tail element의 이전에 element를 삽입한다.
* List\_front
  + Prototype : struct list\_elem \* list\_front (struct list \*list)
  + Parameter : 첫번째 element의 주소값을 반활받을 linke list의 주소값
  + Return : 해당 linked list의 첫번째 element의 주소값
  + Function : linked list를 전달 받아 해당 list의 첫번째 element의 주소값을 반환한다.
* List\_back
  + Prototype : struct list\_elem \* list\_back (struct list \*list)
  + Parameter : 마지막 element의 주소값을 반활받을 linke list의 주소값
  + Return : 해당 linked list의 마지막 element의 주소값
  + Function : linked list를 전달 받아 해당 list의 마지막 element의 주소값을 반환한다.
* List\_pop\_back
  + Prototype : struct list\_elem \* list\_pop\_back (struct list \*list)
  + Parameter : 마지막 element를 pop하고 싶은 linked list의 주소값
  + Return : 해당 linked list의 마지막 element의 주소값
  + Function : linked list를 전달받아 해당 list의 마지막 element를 제거한 뒤 이를 반환한다.
* List\_insert\_ordered
  + Prototype : void list\_insert\_ordered (struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux)
  + Parameter : element가 정렬되어 있는 linked listm이 주소값, 삽입할 elment의 주소값, 두 element가 주어졌을 때 어떤 것이 더 작은 element인지 확인할 수 있는 less 함수의 주소값, less함수에게 추가적으로 넘겨줄 값
  + Return : 없음
  + Function : linked list가 정렬되어 있을 때 새로운 값을 넣었을 때에도 정렬을 유지하기 위해 적절한 위치에 이를 넣어준다.
* List\_size
  + Prototype : size\_t list\_size (struct list \*list)
  + Parameter : size를 알고싶은 linked list의 주소값
  + Return : linked list의 주소값
  + Function : linked list를 전달받아 해당 list의 크기를 반환한다.
* List\_max
  + Prototype : struct list\_elem \*list\_max (struct list \*list, list\_less\_func \*less, void \*aux)
  + Parameter : element의 값을 탐색할 linked list의 주소값, 두 element가 주어졌을 때 어떤 element가 더 작은 값인지 반환하는 less함수, less 함수에 추가적으로 전다할 값
  + Return : linked list에서 가장 큰 값을 가진 element의 주소값
  + Function : 전달받은 linked list에서 가장 큰 값의 element의 주소값을 반환한다.
* List\_min
  + Prototype : struct list\_elem \*list\_min (struct list \*list, list\_less\_func \*less, void \*aux)
  + Parameter : element의 값을 탐색할 linked list의 주소값, 두 element가 주어졌을 때 어떤 element가 더 작은 값인지 반환하는 less함수, less 함수에 추가적으로 전달할 값
  + Return : linked list에서 가장 작은 값을 가진 element의 주소값
  + Function : 전달받은 linked list에서 가장 작은 값의 element의 주소값을 반환한다.
* Iist\_tail
  + Prototype : struct list\_elem \* list\_tail (struct list \*list)
  + Parameter : tail값을 반환받을 linked list의 주소값
  + Return : linked list의 tail의 주소값
  + Function : linked list가 주어지면 해당 list의 tail의 주소값을 반환한다.
* List\_remove
  + Prototype : struct list\_elem \* list\_remove (struct list\_elem \*elem)
  + Parameter : 제거할 list의 element의 주소값
  + Return : 제거한 element의 다음 element의 주소값을 반환한다.
  + Function : 해당 element를 linked list에서 제거하고 해당 element의 다음 element를 반환한다.
* List\_reverse
  + Prototype : void list\_reverse (struct list \*list)
  + Parameter : 순서를 뒤집을 linke list의 주소값
  + Return : 없음
  + Function : 주어진 linked list의 순서를 뒤집는다.
* List\_sort
  + Prototype : void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux)
  + Parameter : 정렬할 linked list의 주소값, 두 element의 받았을 때 어떤 값이 더 작은 값인지 반환해주는 less 함수, less 함수에 추가적으로 전달해줄 값
  + Return : 없음
  + Function : 전달받은 linked list 정렬한다.
* List\_splice
  + Prototype : void list\_splice (struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last)
  + Parameter : linked list를 잘라 삽입할 위치의 다음 element의 주소값과 linked list중 자를 element의 시작 element 주소 값과 마지막 element의 다음 주소값
  + Return : 없음.
  + Function : linked list에서 first부터 last이전까지 element를 잘러 before이전의 위치에 삽입한다.
* List\_swap
  + Prototype : void list\_swap(struct list\_elem \*a, struct list\_elem \*b)
  + Parameter : 값을 바꿀 두개의 element 주소값
  + Return : 없음
  + Function : 두개의 element의 값을 서로 바꾼다.
* List\_unique
  + Prototype : void list\_unique (struct list \*list, struct list \*duplicates, list\_less\_func \*less, void \*aux)
  + Parameter : 중복된 값을 지울 linked list와 지운 중복된 값을 받을 linked list의 주소값, 두개의 element중 어떤 값이 작은 값인지 판별하는 less함수와 less함수에 추가적으로 전달할 값.
  + Return : 없음
  + Function : linked list에서 중복된 값을 제거한 후 이를 다른 linked list에 저장한다. 만약 duplicate에 NULL이 전달되면 중복값 제거만 수행한다.
* List\_shuffle
  + Prototype : void list\_shuffle(struct list \*list)
  + Parameter : 값을 섞을 linked list의 주소값
  + Return : 없음
  + Function : 전달받은 linked list 값들을 뒤섞는다.
* ListLessFunction
  + Prototype : bool ListLessFunction(const struct list\_elem \*a, const struct list\_elem \*b, void\* aux)
  + Parameter : 값을 비교할 두 element와 추가적인 값.
  + Return : 첫번째 element가 두번째 element보다 크면 true, 아니면 false
  + Function : 첫번째 element가 두번째 element보다 크면 true, 아니면 false를 반환한다.

1. **Hash Table**

* Hash\_init
  + Prototype : bool hash\_init (struct hash \*h, \_hash\_func \*hash, hash\_less\_func \*less, void \*aux)
  + Parameter : 초기화할 hash의 주소 값과 hash fucntion의 주소값, 그리고 두개의 hash element 중 어떤 값이 더 작은지 Boolean 값으로 반환해주는 함수의 주소 값, 그리고 추가적인 값을 전달받는다. 본 프로그램에서는 추가적인 값에 이름을 전달한다.
  + Return : hash를 초기화하는 것을 성공하면 true, 아니면 false를 반환한다.
  + Function : struct hash 자료형을 선언한 것을 초기화 시켜주는 역할을 한다. Element의 개수의 0개로 bucket의 개수를 4개로 값을 저장하고 bucket을 동적할당 하여 크기 4의 배열로 만든다. 그리고 hash function과 두 hash element 중 작은 값을 판별하는 less function을 할당하고 aux에 해당 해시 테이블의 이름을 저장한다.
* Hash\_first
  + Prototype : void hash\_first (struct hash\_iterator \*i, struct hash \*h)
  + Parameter : 초기화할 hash\_iterator의 주소 값과 iterator을 사용할 hash의 주소값.
  + Return : 없음
  + Function : iterator란 해시 테이블을 반복문을 통해 element를 탐색하기 위한 구조체이며 현제 어떤 element를 탐색하고 있는지에 대한 값을 저장한다. 해당 함수는 iterator 전달받아 이를 초기화한다.
* Hash\_next
  + Prototype : struct hash\_elem \*hash\_next (struct hash\_iterator \*i)
  + Parameter : 탐색을 위한 해시테이블의 iterator의 주소값
  + Return : 현제까지 탐색한 element의 다음 element의 주소값.
  + Function : iterator 구조체의 경우 해시테이블에서 어떤 element까지 탐색하고 있는지를 저장하고 있다. 이를 참조하여 다음 element값을 반환하고 또한 이러한 진행사항을 iterator에 저장한다.
* Hash\_entry
  + Prototype : hash\_entry(HASH\_ELEM, STRUCT, MEMBER)
  + Parameter : hash\_element의 주소값과 해당 hash\_element를 멤버로 가지고 있는 구조체의 자료형, 해당 자료형의 hash\_element의 변수명
  + Reurn : hash\_element를 가지고 있는 구조체의 주소값
  + Function : hash\_element와 해당 hash\_element를 멤버로 가지고있는 구조체의 정보를 전달하여 해당 구조체의 주소를 반환한다.
* Hash\_destroy
  + Prototype : void hash\_destroy (struct hash \*h, hash\_action\_func \*destructor)
  + Parameter : 제거하고 싶은 해시테이블의 주소와 element가 주어 졌을 때 이를 제거하는 함수의 주소값
  + Return : 없음
  + Function : 해시테이블과 제거하는 함수가 주어지면 해당 해시테이블을 제거한다.
* Hash\_insert
  + Prototype : struct hash\_elem \*hash\_insert (struct hash \*h, struct hash\_elem \*new)
  + Parameter : 새로운 값을 넣을 해시테이블의 주소값과 hash\_element의 주소값
  + Return : 만약 넣을값이 이미 해시테이블에 있을시 해당 element의 주소값을 없을시 NULL을 반환한다.
  + Function : 해시테이블에 새로운 값을 넣는다. 만약 이미 값을 있을시 해당 값을 저장하는 element의 주소값을, 없을시 NULL을 반환한다.
* Hash\_apply
  + Prototype : void hash\_apply (struct hash \*h, hash\_action\_func \*action)
  + Parameter : action 함수를 적용할 해시 테이블과 해시테이블의 element를 조작하는 사용자 정의 함수의 주소
  + Return : 없음
  + Function : 해시 element를 조작하는 사용자 정의 함수를 해시 테이블 내 모든 element에게 적용한다.
* Hash\_delete
  + Prototype : struct hash\_elem \* hash\_delete (struct hash \*h, struct hash\_elem \*e)
  + Parameter : 지울 element를 가진 해시 테이블과 제거하고자 하는 elment
  + Return : 제거할 hash\_element의 주소값
  + Function : hash element 를 전달받은 해시테이블에서 제거한다.
* Hash\_empty
  + Prototype : bool hash\_empty (struct hash \*h)
  + Parameter : 빈 해시테이블인지 확인 할 해시테이블
  + Return : 해시테이블이 비었을 경우 true, 아니면 false
  + Function : 해시테이블이 비어있을경우 true를 아니면 false를 반환한다.
* Hash\_size
  + Prototype : size\_t hash\_size (struct hash \*h)
  + Parameter : 크기를 알고 싶은 해시테이블
  + Return : 해시테이블의 크기
  + Function : 해시테이블의 크기를 반환한다.
* Hash\_clear
  + Prototype : void hash\_clear (struct hash \*h, hash\_action\_func \*destructor)
  + Parameter : element를 모두 지울 해시테이블과 element를 제거하는 사용자 정의 함수
  + Return : 없음
  + Function : 해시테이블의 모든 element를 전달받은 사용자 정의 함수를 통해 제거한다.
* Hash\_find
  + Prototype : struct hash\_elem \* hash\_find (struct hash \*h, struct hash\_elem \*e)
  + Parameter : element를 찾을 해시테이블과 찾고 싶은 해시테이블
  + Return : element를 찾으면 해당 element의 주소값을 반환
  + Function : 해시테이블에서 element를 찾은 후 있을 경우 해당 주소값을 반환한다. 없을 경우 NULL을 반환한다.
* Hash\_replace
  + Prototype : struct hash\_elem \* hash\_replace (struct hash \*h, struct hash\_elem \*new)
  + Parameter : 새로운 element로 element를 바꿀 해시 테이블과 교환하고자 하는 element
  + Return : 교환을 했을 경우 이전의 hash element의 주소값을 하지 않았을 경우 NULL값 반환
  + Function : 새로운 element를 해시테이블에서 같은 값의 element와 교환한다.
* hash\_int\_2
  + prototype : unsigned hash\_int\_2(int i)
  + parameter : 해시 키 값
  + return : 해시테이블의 bucket index
  + function : 해시테이블의 키를 반환한다.
* HashLessFunction
  + prototype : unsigned HashFunction (const struct hash\_elem \*e, void \*aux)
  + parameter : 해시테이블의 bucket index를 받을 element와 추가적인 값
  + return : 해당 element의 해시테이블에서 bucket의 index값
  + function : element를 받아 해시테이블의 키를 반환한다
* HashDestructor
  + Prototype : void HashDestructor (struct hash\_elem \*e, void \*aux)
  + Parameter : 제거할 element와 추가적인 값
  + Return : 없음
  + Function : element를 받아 동적할당을 해제한다.
* HashSquare
  + Prototype : void HashSquare (struct hash\_elem \*e, void \*aux)
  + Parameter : 값을 바꿀 element와 추가적인 값
  + Return : 없음
  + Function : element를 받아 해당 element의 값을 제곱한다.
* HashTriple
  + Prototype : void HashTriple (struct hash\_elem \*e, void \*aux)
  + Parameter : 값을 바꿀 element와 추가적인 값
  + Return : 없음
  + Function : element를 받아 해당 element의 값을 세제곱한다.

1. **Bitmap**

* Bitmap\_create
  + Prototype : struct bitmap \*bitmap\_create (size\_t bit\_cnt)
  + Parameter : 만들고 싶은 bitmap의 크기
  + Return : 새롭게 만든 bitmap의 주소 값
  + Function : bitmap을 동적할당하고 해당 bitmap 구조체의 멤버 값을 초기화 해준 뒤 새롭게 만든 bitmap의 주소값을 반환한다.
* Bitmap\_size
  + Prototype : size\_t bitmap\_size (const struct bitmap \*b)
  + Parameter : size를 알고 싶은 bitmap의 주소값
  + Return : 전달받은 bitmap의 size
  + Function : bitmap을 전달받으면 해당 bitmap의 크기를 반환한다.
* Bitmap\_test
  + Prototype : bool bitmap\_test (const struct bitmap \*b, size\_t idx)
  + Parameter : bitmap의 주소값과 해당 bitmap에서 확인하고 싶은 index
  + Return : 전달받은 bitmap의 index의 값에 따른 true 혹은 false
  + Function : 전달받은 bitmap의 index의 값을 확인해 해당 값이 1일시 true를 0일시 false를 반환한다.
* Bitmap\_destroy
  + Prototype : void bitmap\_destroy (struct bitmap \*b)
  + Parameter : 제거하고 싶은 bitmap의 주소값
  + Return : 없음
  + Function : 비트맵이 주어지면 동적할당을 해제한다.
* Bitmap\_set
  + Prototype : void bitmap\_set (struct bitmap \*b, size\_t idx, bool value)
  + Parameter : 값을 바꿀 비트맵과 바꾸고자하는 index 그리고 설정하고하는 value
  + Return : 없음
  + Function : 비트맵의 특정 index에 ture 혹은 false를 설정한다.
* Bitmap\_mark
  + Prototype : void bitmap\_mark (struct bitmap \*b, size\_t bit\_idx)
  + Parameter : 값을 바꿀 비트맵과 바꾸고자하는 index
  + Return : 없음
  + Function : 비트맵의 특정 index에 값을 true로 바꾼다.
* Bitmap\_reset
  + Prototype : void bitmap\_reset (struct bitmap \*b, size\_t bit\_idx)
  + Parameter : 값을 바꿀 비트맵과 바꾸고자하는 index
  + Return : 없음
  + Function : 비트맵의 특정 index에 값을 false로 바꾼다.
* Bitmap\_all
  + Prototype : bool bitmap\_all (const struct bitmap \*b, size\_t start, size\_t cnt)
  + Parameter : 값이 true인지 확인할 비트맵과 확인을 시작하고하는 index와 index부터 확인하고자 하는 bit의 개수
  + Return : 확인한 값이 모두 true면 true 반환 아니면 false 반환
  + Function : 비트맵에서 시작 index부터 전달해준 개수만큼 모두 true이면 true 반환 아니면 false 반환
* Bitmap\_none
  + Prototype : bool bitmap\_none (const struct bitmap \*b, size\_t start, size\_t cnt)
  + Parameter : 값이 false인지 확인할 비트맵과 확인을 시작하고하는 index와 index부터 확인하고자 하는 bit의 개수
  + Return : 확인한 값이 모두 false면 true 반환 아니면 false 반환
  + Function : 비트맵에서 시작 index부터 전달해준 개수만큼 모두 false이면 true 반환 아니면 false 반환
* bitmap\_any
  + Prototype : bool bitmap\_any (const struct bitmap \*b, size\_t start, size\_t cnt)
  + Parameter : 값이 true인지 확인할 비트맵과 확인을 시작하고하는 index와 index부터 확인하고자 하는 bit의 개수
  + Return : 확인한 값 중 하나라도 true면 true 반환 아니면 false 반환
  + Function : 비트맵에서 시작 index부터 전달해준 개수의 값 중 하나만 true여도 true 반환 아니면 false 반환
* bitmap\_contains
  + Prototype : boolbitmap\_contains (const struct bitmap \*b, size\_t start, size\_t cnt, bool value)
  + Parameter : 값이 true혹은 false인지 확인할 비트맵과 확인을 시작하고하는 index와 index부터 확인하고자 하는 bit의 개수와 확인할 논리값
  + Return : 확인한 값중 하나라도 사용자가 지정한 논리값이면 true 반환 아니면 false 반환
  + Function : 비트맵에서 시작 index부터 전달해준 개수의 값 중 하나만 사용자가 지정한 논리값이면 true 반환 아니면 false 반환
* bitmap\_count
  + prototype : size\_t bitmap\_count (const struct bitmap \*b, size\_t start, size\_t cnt, bool value)
  + parameter : 사용자가 지정한 논리값의 개수를 확인할 비트맵과 확인을 시작하고하는 index와 index부터 확인하고자 하는 bit의 개수와 확인할 논리값
  + return : 해당 범위 내의 사용자가 지정한 논리값의 개수
  + function : 비트맵에서 시작 index부터 해당 개수의 bit가 사용자가 지정한 논리값이 몇 개인지 반환한다.
* bitmap\_expand
  + prototype : struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size)
  + parameter : 확장하고자하는 비트맵과 확장하고 싶은 크기
  + return : 확장된 비트맵의 주소값
  + function : 주어진 비트맵을 주어진 크기만큼 더 확장해서 비트맵을 바환한다.
* bitmap\_set\_all
  + prototype : void bitmap\_set\_all (struct bitmap \*b, bool value)
  + parameter : 값을 설장한 비트맵과 설정한 논리값
  + return : 없음
  + function : 전달된 비트맵을 전달한 논리값으로 모두 채운다.
* bitmap\_flip
  + prototype : void bitmap\_flip (struct bitmap \*b, size\_t bit\_idx)
  + parameter : 값을 바꿀 비트맵과 바꾸고 하는 비트의 index
  + return : 없음
  + function : 전달된 비트맵에서 전달된 index의 비트의 값을 반대로 바꾼다
* bitmap\_scan
  + prototype : size\_t bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value)
  + parameter : 값을 확인할 비트맵과 값을 확인하기 시작할 index와 얼만큼 연속적인 값을 찾을지에 대한 크기와 논리값
  + return : 사용자가 지정한 논리값으로 사용자가 전달한 크기만큼 연속될 경우 연속된 비트중 시작 index 없을시 BITMAP\_ERROR 반환
  + function : 비트맵에서 사용자가 지정한 시작값부터 사용자가 지정한 논리값으로 사용자가 지정한 개수만큼 연속된 비트가 있는지 확인 후 있을 경우 연속된 비트의 첫 비트의 index를 반환한다. 없을 경우 BITMAP\_ERROR를 반환한다.
* bitmap\_scan\_and\_flip
  + prototype : size\_t bitmap\_scan\_and\_flip (struct bitmap \*b, size\_t start, size\_t cnt, bool value)
  + parameter : 값을 확인 후 바꿀 비트맵과 값을 확인하기 시작할 index와 얼만큼 연속적인 값을 찾을지에 대한 크기와 논리값
  + return : 사용자가 지정한 논리값으로 사용자가 전달한 크기만큼 연속될 경우 연속된 비트중 시작 index 없을시 BITMAP\_ERROR 반환
  + function : 비트맵에서 사용자가 지정한 시작값부터 사용자가 지정한 논리값으로 사용자가 지정한 개수만큼 연속된 비트가 있는지 확인 후 있을 경우 해당 연속된 비트들을 모두 반대값으로 바꾼 후 연속된 비트의 첫 비트의 index를 반환한다. 없을 경우 BITMAP\_ERROR를 반환한다.
* bitmap\_set\_multiple
  + prototype : void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t cnt, bool value)
  + parameter : 값을 바꿀 비트맵과 바꾸기 시작할 index와 index부터 바꿀 비트의 개수, 그리고 설정하고 하는 논리값
  + return : 없음
  + function : 비트맵에서 시작 index부터 지정한 개수만큼 지정한 논리값으로 바꾼다.
* bitmap\_dump
  + prototype : void bitmap\_dump (const struct bitmap \*b)
  + parameter : 16진법으로 출력할 bitmap
  + return : 없음
  + function : 해당 bitmap을 16진법으로 stdout에 출력한다.