**Project #1 : MyLib**

|  |  |
| --- | --- |
| 담당 교수 : | 이영민 교수 |
| 학번 : | 20201572 |
| 이름 : | 김지섭 |
|  |  |

1. **Additional Implementation**

**struct data**

struct data **{**

struct list list**;**

struct hash hash**;**

struct bitmap**\*** bitmap**;**

char name**[**NAME\_LEN**];**

int type;

**};**

하나의 구조체로 자료구조를 관리하기 위해 선언하였다. 해당 구조체는 list, hash와 bitmap의 포인터를 내재하고, 프로그램에서 각 구조체를 구별할 이름, 그리고 해당 구조체가 생성될 때 정해지는 type가 있다.

**struct data\* ds[MAX\_STRUCTS]**

최대 10개까지의 자료구조를 저장하기 위해 선언한 전역 변수 배열. create 명령어를 통해 새로운 자료구조가 생길 때 동적으로 할당한 data 구조체의 메모리 주소를 담는다.

**int struct\_cnt**

현재 프로그램에서 선언한 자료구조의 개수를 저장하는 전역 변수. 이름으로 특정 자료구조에 접근하거나, 반복문을 돌릴 때 사용되며, create와 delete 명령어에 의해 값이 변한다.

**int quit\_flag**

0으로 설정되며, quit 명령어를 읽어들일 경우 해당 flag가 1로 세팅되며 parse\_file 함수의 무한 반복문에서 벗어나 프로그램 종료 절차를 밟는다.

**TYPE\_LIST, TYPE\_HASH, TYPE\_BITMAP**

값은 각각 0, 1, 2에 해당하며, 프로그램 내에서 현재 다루는 자료구조를 구별하기 위해 switch문, if 조건문 등에 사용된다.

|  |  |
| --- | --- |
| **Prototype** | void eval (int argc, char\* argv[ ]) |
| **Parameter** | int argc (# of strings inside argv)  char\* argv[ ] (Strings of parameters) |
| **Return** | None |
| **Function** | Parameter로 받은 명렁어에 따라 지정된 함수를 호출하는 역할을 한다. testlib 자체에서 지원하는 명령어 (create, dumpdata, delete, quit)의 경우 필요한 처리를 한 후 각 함수를 호출하고, 그 외 각 자료구조에 해당하는 함수들은 eval\_func를 통해 처리하도록 구현하였다. |

|  |  |
| --- | --- |
| **Prototype** | void eval\_func(int argc, char\* argv[ ]) |
| **Parameter** | int argc (# of strings inside argv)  char\* argv[ ] (Strings of parameters) |
| **Return** | None |
| **Function** | get\_type 함수를 이용해 명령어의 이름에 따라 해당하는 자료구조에 대한 처리용 함수를 switch문으로 호출한다. 유효하지 않은 명령어에 대해서는 에러 처리를 하였다. |

|  |  |
| --- | --- |
| **Prototype** | int get\_type (char\* arg) |
| **Parameter** | char\* arg (String containing data struct-specific command) |
| **Return** | int (Pre defined type) |
| **Function** | 인자로 받은 명령어의 첫 4글자에 대해 strncmp 함수를 이용하여 사전 정의된 TYPE\_LIST, TYPE\_HASH, TYPE\_BITMAP을 반환한다. 유효하지 않은 자료구조에 대해서는 에러 처리를 하였다. |

|  |  |
| --- | --- |
| **Prototype** | void parse\_file (int mode, const char\* filename) |
| **Parameter** | int mode (By default, read from stdin, if set to 1, read from given file argument)  const char\* filename (Contains name of file argument, if no files were given, is NULL) |
| **Return** | None |
| **Function** | main 함수에서 파일을 인자로 받았다면 파일 포인터를 생성하여 한 줄씩 읽어들인다. 무한 반복문에서 strtok 함수를 이용해 명령 한 줄을 띄어쓰기 표시로 나눠 읽으며 args 배열에 저장 후, eval 함수에 한 줄의 명령어 개수와 함께 이를 넘긴다. 만약 quit 명령어를 읽어 eval 함수에서 quit\_flag를 활성화 하면 반복문에서 나와 파일 포인터를 정리하고, cleanup 함수를 호출하여 필요한 메모리 할당 해제를 진행한다. |

|  |  |
| --- | --- |
| **Prototype** | struct data\* find\_data(const char\* name) |
| **Parameter** | const char\* name (name of data structure being dealt with) |
| **Return** | struct data\* (Pointer to said data) |
| **Function** | 사용자가 생성한 자료구조가 저장된 배열 ds를 순회하며 각 data의 이름을 비교해 이름이 같은 data에 대한 포인터를 반환한다. 해당하는 포인터가 없을 경우 NULL을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void cleanup( ) |
| **Parameter** | None |
| **Return** | None |
| **Function** | parse\_file 함수의 무한 반복문에서 나와 프로그램을 종료하기 전 heap에 할당한 메모리를 정리하는 함수이다. delete 함수를 호출하여 해당하는 자료구조가 차지한 메모리를 해제해주고, ds 배열에서 동적으로 할당한 메모리 또한 해제한다. |

|  |  |
| --- | --- |
| **Prototype** | void create(int type, const char\* name, int\* size) |
| **Parameter** | int type (Pre defined type)  const char\* name (Name of user-created data structure)  int\* size (Auxiliary data used for bitmap initialization) |
| **Return** | None |
| **Function** | 입력으로 받은 새 자료구조에 대한 메모리를 할당한 후, 필요한 정보를 넣고, 각 자료구조에 대한 초기화 함수를 호출한다. 새로운 자료구조를 생성한 후에 전역 변수로 존재하는 배열 ds에 해당 자료구조의 메모리 주소 포인터를 저장하고, struct\_cnt를 1 올려 현재 생성한 자료구조의 개수를 업데이트한다. |

|  |  |
| --- | --- |
| **Prototype** | void dumpdata(struct data\* d) |
| **Parameter** | struct data\* d (Pointer to target data structure) |
| **Return** | None |
| **Function** | 입력으로 받은 자료구조의 내용물을 모두 출력하는 함수로, list의 경우 list\_elem을 통해 순회하며 각 내용을 출력하고, hash table의 경우 hash\_apply 함수에 별도로 선언한 hash\_print 함수를 넣어 각 내용을 출력하도록 하며, bitmap의 경우 각 비트에 대해 출력하도록 하였다. |

|  |  |
| --- | --- |
| **Prototype** | void delete(struct data\* d) |
| **Parameter** | struct data\* d (Pointer to target data structure) |
| **Return** | None |
| **Function** | 입력으로 받은 자료구조를 삭제하며 메모리 할당을 해제한다. 각 자료구조에 알맞게 destroy 함수 등을 호출하여 메모리 누수가 일어나지 않도록 하며, 이후 배열 ds에서 대상 자료구조 뒤에 저장되어 있던 다른 자료구조들의 포인터를 한 칸씩 앞으로 당겨준 후, 해당 자료구조를 가리키던 포인터 역시 메모리 해제를 하고, struct\_cnt를 1 내려 자료구조의 개수를 업데이트한다. |

1. **List**

|  |  |
| --- | --- |
| **Prototype** | bool cmp\_list(struct list\_elem\* a, struct list\_elem\* b, void\* aux) |
| **Parameter** | struct list\_elem\* a, b (Pointers to list elements for comparison) |
| **Return** | bool (True if a->data < b->data, else False) |
| **Function** | list\_less\_func로 사용되는 함수, list\_entry 매크로를 이용하여 각 포인터를 내재하는 구조체의 값을 얻어, a의 값이 b보다 클 경우 true, 아니면 false를 반환한다. list\_sort, list\_unique 등의 함수의 인자로 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem\* a, struct list\_elem\* b) |
| **Parameter** | struct list\_elem\* a, b (Pointers to list elements for swapping) |
| **Return** | None |
| **Function** | 두 list\_elem의 포인터를 받아, doubly linked list 내의 위치를 서로 바꾸는 함수이다. a와 b가 이웃하는 경우 이웃하는 list\_elem 2개에 대해서만. 아닌 경우 총 4개의 이웃하는 list\_elem의 포인터를 서로 바꿔준다. 이후 a와 b 내부의 포인터를 서로 변경해준다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list\* list) |
| **Parameter** | struct list\* list (Pointer to list itself that will be shuffled) |
| **Return** | None |
| **Function** | Fischer-Yates 방식을 이용하여 리스트 내의 위치를 임의로 뒤섞는 함수이다. 우선 리스트 내부의 모든 list\_elem을 동적 할당한 배열에 넣어준 후, rand 함수를 이용해 뒤에서부터 임의의 index를 정해 둘의 위치를 list\_swap을 이용해 바꿔준다. 이후에 배열은 메모리를 해제해준다. |
| **Prototype** | void list\_to\_array(struct list\* list, struct list\_elem\*\* array, size\_t size) |
| **Parameter** | struct list\* list (Pointer to list that contains the elements)  struct list\_elem\*\* array (Pointer to array that will contain list\_elem)  size\_t size (Size of array) |
| **Return** | None |
| **Function** | Fischer-Yates 방식을 사용하기 위해 리스트에 있는 list\_elem들을 array에 넣어 빠른 shuffling을 돕는 함수이다. list를 while문을 통해 돌며 각 list\_elem을 동적 할당한 array에 삽입한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_idx(struct list\* list, int index) |
| **Parameter** | struct list\* list (List to search)  int index (Index of target list\_elem) |
| **Return** | struct list\_elem\* (List element of target index in list) |
| **Function** | 명령어에서 index 기반으로 list를 사용할 때 호출되는 함수이다. 반복문을 통해 list\_begin에서 시작하여 index만큼 반복하여 해당하는 list\_elem을 반환한다. 비어있는 list에서 사용 시 null pointer를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_cleanup(struct list\* list) |
| **Parameter** | struct list\* list (List to deallocate from memory) |
| **Return** | None |
| **Function** | list 구조체를 삭제하거나, 프로그램을 종료하기 전, 동적으로 할당된 list\_elem을 해제하여 메모리 누수를 방지하는 함수이다. list가 빌 때까지 list\_pop\_front를 통해 모든 list\_elem을 해제하도록 한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_func(int argc, char\* argv[ ]) (in main.c) |
| **Parameter** | int argc (Number of arguments stored in argv)  char\* argv[ ] (Arguments in a line as strings) |
| **Return** | None |
| **Function** | eval\_func 함수에서 인자들을 전달 받아 list 자료구조에 대한 처리를 하는 함수이다. argv에 저장된 명령의 이름에 알맞게 list.c에 구현된 함수를 호출하며, 문자열로 저장된 인자들은 find\_data, strtol, list\_idx 등의 함수를 이용하여 올바른 data type으로 변환하여 인자들을 전달한다. list\_unique 명령어의 경우 인자의 개수가 유동적이므로 각자 따로 처리를 해주었다. |

List 자료구조는 list.h에서 별도로 선언한 list\_item 구조체 내부에 list\_elem 구조체를 embedding 하여 구현하였다. 따라서 새로운 리스트를 생성할 때 malloc 함수를 통해 list\_item만큼의 메모리를 할당한 후, 내재된 list\_elem을 list\_insert 함수를 이용해 삽입하여 메모리 주소 자체는 유지하며 list\_elem을 통해서만 접근하도록 하였다.

1. **Hash Table**

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int key) |
| **Parameter** | int key (Key to perform hash function on) |
| **Return** | unsigned (The hash value) |
| **Function** | 32비트 unsigned 자료형을 반환하는 해시 함수로 Murmurhash3 방식을 이용하였다. 해당 방식은 입력으로 받은 key를 4비트 단위로 나눈 후, 큰 소수와의 곱셈 및 비트 쉬프트 연산, XOR 연산을 이용해 비트들을 섞도록 한다. 이때의 큰 소수는 해시값이 골고루 분포하도록 하는 역할을 한다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned my\_hash\_func(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e (Hash table element to perform hashing)  void aux (Unused auxiliary data) |
| **Return** | unsigned (The hash value) |
| **Function** | hash\_init 함수에서 hash table에 사용할 함수를 등록하기 위해 사용하는 함수이다. hash\_entry 매크로로 현재 hash\_elem에서 해시 함수에 넣을 값을 추출하고, hash\_int 함수를 이용하여 해시값을 계산하여 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool cmp\_hash(struct hash\_elem\* a, struct hash\_elem\* b, void\* aux) |
| **Parameter** | struct hash\_elem\* a, b (Hash table elements to compare)  void aux (Unused auxiliary data) |
| **Return** | bool (True if a->data < b->data, else False) |
| **Function** | hash\_less\_func로 사용되는 함수, hash\_entry 매크로를 이용하여 각 포인터를 내재하는 구조체의 값을 얻어, a의 값이 b보다 클 경우 true, 아니면 false를 반환한다. hash\_init 함수의 인자로 사용되어 hash\_elem 사이의 값을 비교할 때 사용된다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_print(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e (Hash table element to print to stdout)  void aux (Unused auxiliary data) |
| **Return** | None |
| **Function** | hash\_apply 함수에 인자로 넘겨져 값을 출력해주는 함수이다. hash\_entry 매크로를 이용하여 구조체의 값을 얻어 stdout으로 출력해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_square(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e (Hash table element to square)  void aux (Unused auxiliary data) |
| **Return** | None |
| **Function** | hash\_apply 함수에 인자로 넘겨져 값을 변경해주는 함수이다. hash\_entry 매크로를 이용하여 구조체의 값을 얻어 제곱한 값을 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_triple(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e (Hash table element to print to triple)  void aux (Unused auxiliary data) |
| **Return** | None |
| **Function** | hash\_apply 함수에 인자로 넘겨져 값을 변경해주는 함수이다. hash\_entry 매크로를 이용하여 구조체의 값을 얻어 세제곱한 값을 저장한다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_free(struct hash\_elem\* e, void\* aux) |
| **Parameter** | struct hash\_elem\* e (Hash table element to free)  void aux (Unused auxiliary data) |
| **Return** | None |
| **Function** | hash\_apply 함수에 인자로 넘겨져 hash table의 element를 삭제하는 함수이다. hash\_entry 매크로를 이용하여 구조체의 포인터를 얻은 후, 동적으로 할당된 메모리를 해제해준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_func(int argc, char\* argv[ ]) (in main.c) |
| **Parameter** | int argc (Number of arguments stored in argv)  char\* argv[ ] (Arguments in a line as strings) |
| **Return** | None |
| **Function** | eval\_func 함수에서 인자들을 전달 받아 hash table 자료구조에 대한 처리를 하는 함수이다. argv에 저장된 명령의 이름에 알맞게 hash.c에 구현된 함수를 호출하며, 문자열로 저장된 인자들은 find\_data, strtol 등의 함수를 이용하여 올바른 data type으로 변환하여 인자들을 전달한다.. |

Hash table 자료구조는 list로 구현된 bucket들을 별도의 list에 넣어 관리하는 방식으로 구현되었다. hash\_init 함수에서 사용할 해시 함수를 지정하며, 본 과제에서는 hash\_int 함수를 이용하였다.

1. **Bitmap**

|  |  |
| --- | --- |
| **Prototype** | struct bitmap\* bitmap\_expand(struct bitmap\* b, int size) |
| **Parameter** | struct bitmap\* b (Pointer to target bitmap to expand)  int size (Size to expand bitmap) |
| **Return** | struct bitmap\* (Pointer to newly expanded bitmap) |
| **Function** | 대상 bitmap과 크기를 인자로 받아 해당 크기만큼 기존 bitmap을 확장시키는 함수이다. 이는 새로운 크기로 bitmap을 생성하여 기존 bitmap의 값을 옮기고, 기존 bitmap은 제거하는 방식으로 구현되었다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_func(int argc, char\* argv[ ]) (in main.c) |
| **Parameter** | int argc (Number of arguments stored in argv)  char\* argv[ ] (Arguments in a line as strings) |
| **Return** | None |
| **Function** | eval\_func 함수에서 인자들을 전달 받아 bitmap 자료구조에 대한 처리를 하는 함수이다. argv에 저장된 명령의 이름에 알맞게 bitmap.c에 구현된 함수를 호출하며, 문자열로 저장된 인자들은 find\_data, strtol 등의 함수를 이용하여 올바른 data type으로 변환하여 인자들을 전달한다.. |

Bitmap 자료구조는 하나의 unsigned long으로 bit의 배열을 구현하며, bitwise 연산으로 빠르게 특정 값을 추출하거나, 지정할 수 있다.