

CSE 333 – Section 4

MultiSet, SortableArray, and File I/O

Johnny Yan

Department of Computer Science & Engineering
University of Washington

October 17, 2013

Administrivia

- Exercise 4 graded, feedback returned via email
- Exercise 5 & 6 will be graded together
- Next section is a quiz section

Buggy 1

MultiSet.h

```
1 typedef struct {
2     size_t size;
3     int valueVec[20];
4 } MultiSet;
```

MultiSet.c

```
1 MultiSet *multiset_new(size_t size, int *elements) {
2     int i;
3     MultiSet *this = (MultiSet*) malloc(sizeof(MultiSet));
4
5     if (elements != NULL) {
6         for (i = 0; i < size; i++) {
7             this->valueVec[i] = elements[i];
8         }
9     }
10    return this;
11 }
12
13 void multiset_destroy(MultiSet *this){
14     free(this);
15 }
```

Buggy 2

MultiSet.h

```
1 typedef struct {
2     size_t size;
3     int *valueVec;
4 } MultiSet;
```

MultiSet.c

```
1 MultiSet *multiset_new(size_t size, int *elements){
2     MultiSet *newMulti = (MultiSet *)malloc(sizeof(MultiSet));
3     newMulti->size = size;
4     int *allElements = malloc(sizeof(int) * size);
5     for(int i=0; i<size; i++){
6         allElements[i] = elements[i];
7     }
8     newMulti->valueVec = allElements;
9     free(allElements);
10
11     return newMulti;
12 }
13
14 void multiset_destroy(MultiSet *this){
15     free(this);
16 }
```

Buggy 3

MultiSet.c

```
1 MultiSet* multiset_new(size_t size, int *elements) {
2     int *array = (int*) malloc(size * sizeof(int));
3     array = elements;
4     MultiSet *result = (MultiSet*) malloc(sizeof(MultiSet));
5     result->size = size;
6     result->valueVec = array;
7     return result;
8 }
```

Buggy 4

MultiSet.c

```
1 MultiSet *multiset_new(size_t size, int *elements) {
2     MultiSet *multiSet = (MultiSet*)malloc(sizeof(MultiSet));
3     int *valueVec = (int*)malloc(size * sizeof(int));
4     for (size_t i = 0; i < size; i++) {
5         valueVec[i] = elements[i];
6     }
7     multiSet->size = size;
8     multiSet->valueVec = valueVec;
9     return multiSet;
10 }
11
12 MultiSet multiset_union(MultiSet *A, MultiSet *B) {
13     MultiSet *C = multiset_new(A->size + B->size, B->valueVec);
14     for (size_t i = 0; i < A->size; i++) {
15         C->valueVec[i] = A->valueVec[i];
16     }
17
18     for (size_t i = 0; i < B->size; i++) {
19         C->valueVec[i + A->size] = B->valueVec[i];
20     }
21
22     return *C;
23 }
```

Buggy 5

MultiSet.c

```
1 MultiSet *multiset_new(size_t size, int *elements) {
2     MultiSet *m = (MultiSet *) malloc(sizeof(size_t) + (size * sizeof(int)));
3     m->size = size;
4     for(int i = 0; i < size; i++)
5         m->valueVec[i] = elements[i];
6
7     return m;
8 }
9
10 void multiset_destroy(MultiSet *this) {
11     free(this);
12 }
13
14 MultiSet multiset_union(MultiSet *A, MultiSet *B) {
15     size_t newsize = A->size + B->size;
16     MultiSet C = {newsized, {}};
17     int i;
18     for(i = 0; i < A->size; i++)
19         C.valueVec[i] = A->valueVec[i];
20
21     for(; i < C.size; i++)
22         C.valueVec[i] = B->valueVec[i - A->size];
23
24     return C;
25 }
```

Buggy 5 cont.

MultiSet.h

```
1 typedef struct {
2     size_t size;
3     int valueVec[20];
4 } MultiSet;
```

Inspired a new way of implementation!

Buggy 5 cont.

MultiSet.h

```
1 typedef struct {  
2     size_t size;  
3     int valueVec[20];  
4 } MultiSet;
```

Inspired a new way of implementation!

What is SortableArray?

- How do we design SortableArray, data structure and interface?
- What is our goal?

SortableArray struct

```
typedef struct sortable_array_t {  
    unsigned int size;  
    void** data;  
} *SortableArray;
```

- The size field is required by the documentation
- Array of void pointers to achieve generic

Interface

SortableArray.h

```
1  #ifndef _SORTABLEARRAY_H
2  #define _SORTABLEARRAY_H
3
4  #include <stdbool.h>
5
6  struct sortable_array_t;
7  typedef struct sortable_array_t *SortableArray;
8
9  SortableArray SortableArray_new(unsigned int size);
10 bool SortableArray_delete(SortableArray sa);
11 unsigned int SortableArray_size(SortableArray sa);
12 bool SortableArray_set(SortableArray sa,
13                       unsigned int index,
14                       void *dataIn); // returns data
15 bool SortableArray_get(SortableArray sa,
16                       unsigned int index,
17                       void **dataOut);
18
19 // SortableArray_Map invokes a caller-supplied function on each
20 // element of the array. It guarantees to iterate over the elements
21 // in index order. The user supplied callback function should
22 // return false to continue the iteration, and true to stop it.
23 typedef bool (*SortableArray_Map_Callback)(void *el);
24 void SortableArray_map(SortableArray sa, SortableArray_Map_Callback callback);
25
26 // A SortableArray_Comparator function returns -1 if a < b,
27 // 0 if a == b, and 1 if a > b
28 typedef int (*SortableArray_Comparator)(const void *a, const void *b);
29 bool SortableArray_sort(SortableArray sa, SortableArray_Comparator compare);
30
31 #endif // _SORTABLEARRAY_H
```



Implementation

SortableArray.c

```
1  SortableArray SortableArray_new(unsigned int size) {
2      if ( size == 0 ) return NULL;
3      SortableArray sa = (SortableArray)malloc(sizeof(struct sortable_array_t));
4      if ( sa == NULL ) return NULL;
5      sa->size = size;
6      sa->data = (void**)malloc(sizeof(void*) * size);
7      return sa;
8  }
9
10 void SortableArray_map(SortableArray sa, SortableArray_Map_Callback callback) {
11     int i;
12     if ( sa == NULL ) return;
13     for ( i=0; i<sa->size; i++ ) callback(sa->data[i]);
14 }
15
16 bool SortableArray_sort(SortableArray sa, SortableArray_Comparator compare) {
17     if ( sa == NULL ) return false;
18     qsort(sa->data, sa->size, sizeof(void*), compare);
19     return true;
20 }
```

main.c

```
1  int string_compare(const void *a, const void *b) {
2      return strcmp( *(const char**)a, *(const char**)b);
3  }
```

Stdio I/O

```
#include <stdio.h>
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *fp);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- *fopen* opens the file whose name is the string pointed to by *path* and associates a stream with it.
- *mode* can be r–read, w–write or a–append combined with b–binary or t–text. All POSIX conforming systems, including Linux, will ignore b. Adding mode b may be a good idea to make your program portable to non-UNIX environments.
- *fclose* flushes the stream pointed to by *fp* and closes the underlying file descriptor.
- *fread/fwrite* reads/writes *nmemb* elements of data, each *size* bytes long, from/to the stream pointed to by *stream*, storing them at/obtaining them from the location given by *ptr*.

Stdio I/O cont.

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
int fflush(FILE *stream);
```

- *fseek* sets the file position indicator for the stream pointed to by *stream*. The new position, measured in bytes, is obtained by adding *offset* bytes to the position specified by *whence*. If *whence* is set to *SEEK_SET*, *SEEK_CUR*, or *SEEK_END*, the *offset* is relative to the start of the file, the current position indicator, or end-of-file, respectively.
- For output streams, *fflush* forces a write of all user-space buffered data for the given output or update stream via the stream's underlying write function. For input streams, *fflush* discards any buffered data that has been fetched from the underlying file, but has not been consumed by the application. The open status of the stream is unaffected. If the *stream* argument is *NULL*, *fflush* flushes all open output streams.

System I/O

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);

#include <unistd.h>
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

- Similar to their stdio counterparts, but at system level, non-buffered
- Notice that number of bytes read/write is not necessarily the same as desired.