Documentation for Set.h and Set.c

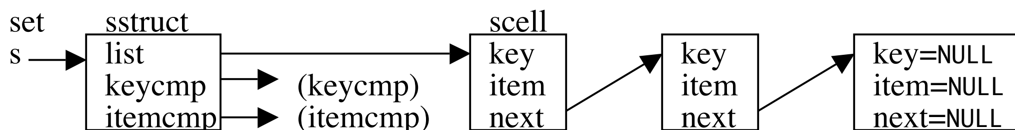
Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
struct scell{
	void *key;
	void *item;
	struct scell *next;
	};

typedef struct sstruct{
	struct scell *list;
	int (*keycmp)(void *,void *);
	int (*itemcmp)(void *,void *);
	} *set;

set SetAlloc(int (*keycmp)(void *,void *),int (*itemcmp)(void *,void *));
void SetFree(set s,int freek,int freex);
void SetNull(set s,int freek,int freex);
int SetInsert(void *k,void *x,set s);
int SetDelete(void *k,void *x,set s,int freek,int freex);
int SetMember(void *k,void *x,set s);
void *SetItem(void *k,set s);
void **SetItemAt(void *k,set s);
void *SetKey(void *x,set s);
struct scell *SetNext(struct scell *ptr,void **k,void **x,set s);
int Subset(set s1,set s2);
int SetEqual(set s1,set s2);
set SetUnion(set s1,set s2);
set SetIntersect(set s1,set s2);
set SetDiff(set s1,set s2);
int SetCard(set s);
```

requires: `<stdlib.h>`
Example: `LibTest.c,Spectfit.c`

Partially written 5/12/95. Completed and modified substantially 9/98. A change made 1/99 is that sets are no longer ordered. Moderately tested. Checked and cleaned up 1/21/02; in the process changed parameters to a couple routines. Moved comparison routines to VoidComp.c 3/14/02.

Diagram of set structure:



This implements basic set operations, on unordered sets of pairs of arbitrary pointers using a linked list. The pointers are called `key` and `item`, emphasizing their intended use as a search key and a referenced item, but they are treated identically by the

routines. A NULL value is valid for a key or item and is treated like any other value. However, several routines return NULL values to indicate that an element was not found, which could be confused with valid items or keys that happen to be equal to NULL. This can be clarified with the SetMember routine if necessary.

There needs to be a way of determining if two elements are equal, which is done with the keycmp and/or itemcmp routines which are identified during set allocation and used in most routines thereafter. Both of these functions should return an integer not equal to 0 if the items are different or 0 if they are equal, and they need to be able to handle NULL inputs. Several useful routines are included in this library.

In general, element testing is strict, such that elements are considered equal only if both the key and the item are equal if both can be tested. If only one can be tested, then that is all that is used for determining equality and the other member is ignored.

SetAlloc is used to create a new set, returning the new set, or NULL if one could not be allocated or if there was an error. The parameters required are the addresses of functions that compare pairs of keys and pairs of items. Either function may be NULL, but the routine returns 0 if both are NULL.

SetFree is used to free a set. freek and freex are flags, where 1 indicates that keys and/or items, respecively, should be freed as well, and 0 indicates that they should not be freed. This can also free partially allocated sets.

SetNull empties the set, but keeps it initallized. As before, freek and freex are flags, where 1 indicates that keys and/or items, respecively, should be freed, and 0 indicates that they should not be freed.

SetInsert adds an element to the set, returning 0 if unsuccessful, 1 if it was inserted, or 2 if element was already in the set, in which case it is not added again. The new element is added at the end of the linked list.

SetDelete deletes an element, returning 1 if the element was found or 0 if not; again, freek and freex may be used to free the members. Note that both the key and the item need to be known.

SetMember is used to ask if a key and item is an element; it returns 1 if the element was found or 0 if not.

SetItem returns the item that corresponds to the given key. It returns NULL if the element is not in the set.

SetItemAt is similar to SetItem, but it returns a pointer to the item instead of the item.

SetKey returns the key that corresponds to the given item. It returns NULL if the element is not in the set.

SetNext is intended for retrieving elements of a set sequentially. For the first call, pass in NULL as the set element pointer and the addresses of variables for retrieving the key and item. The function returns the key, item, and a pointer to the next element, which may be used in subsequent calls to SetNext. At the end, the function returns a NULL key, a NULL item, and a NULL pointer simultaneously. Here is a typical code fragment, in which integer items are summed,

```
for(c=SetNext(NULL,&k,&x,s);c;c=SetNext(c,&k,&x,s)) a+=(int) x;
```

Subset returns 0 if s1 is not a subset of s2 and 1 if it is (including equal sets). Comparison routines are used from s2.

SetEqual returns 0 if two sets are unequal and 1 if they are equal, using comparison routines for both sets.

SetUnion returns a set that is the union of s1 and s2. For this routine and the two following ones, s1 and s2 are allowed to be the same set. The comparison routines for the two sets should behave identically, although this is not checked. As elements of the result set are the same pointers as those used in s1 or s2, care needs to be taken when the sets are freed to free each element exactly once.

`SetIntersection` returns a set that is the intersection of `s1` and `s2`.
`SetDiff` returns a set that contains all the elements of `s1`, less those that are also in `s2`.
`SetCard` returns the number of elements in the set.