MICRO B+™ Function Calls: Language "C" Implementation

October 1980

The type and storage declarations used in the following material are based on the Whitesmiths Ltd implementation of the language "C".

```
VOID intree(nbufs,nkeys)

COUNT nbufs,nkeys;
```

intree initializes the **MICRO B+™** routines for use with nbufs I/O buffers, and nkeys distinct index files. In the remaining functions, the references to the index files (or keys) is done via the numbers 0 to nkeys-1. intree must be called prior to any other functions.

```
COUNT access(keyno,idxnam,keylen,keytyp,keyval,nsectr,prdsup)

COUNT keyno,keylen,keyval,nsectr;
TEXT *idxnam,keytyp,prdsup;
```

access opens and/or creates the index file with name pointed to by idxnam and assigns keyno as its refence number: 0 <= keyno < nkeys. keylen gives the length of key values in bytes. keytyp == 'a' for alphanumeric keys and keytyp == 'i' for two-byte integer key values. Additional key types can be added. keyval specifies the maximum number of key values per B-Tree node. keyval must be an even integer in the range 0 < keyval <= 128. nsectr specifies the number of 128 byte sectors comprising each index file record. Note that each index file record holds one B-Tree node. Currently, 1 <= nsectr <= 4. prdsup == 'y' forces the leaf nodes of the B-Trees to be linked in both directions enabling sequential access either forwards or backwards. prdsup != 'y' disables this feature; hence only forward sequential access is supported.

Assuming that the pointers associated with each key value will require two bytes (see addkey), the maximum value for keyval is given by the largest even integer less than or equal to:

((nsectr*128)-12)/(keylen+2)

access returns the following:

1 if successful;
2 if keyno already assigned;

For previously created index files, access returns the following values if the corresponding parameters do not agree with the values used when the index file was created:

3 keylen	6 nsectr
4 keytyp	7 prdsup
5 keyval	

```
COUNT addkey(keyno,entry,pntr)

COUNT keyno;
TEXT *entry;
POINTER pntr;
```

addkey inserts the key value pointed to by entry into the index file referenced by keyno. The value of pntr is associated with the key value. Presumably, pntr represents a relative record number in a data file which contains the key value. addkey assumes that the key value pointed to by entry is of the exact length specified by keylen in access. Note that '\0' is legitimate character in a key value string; it is not interpreted as a string terminator. These assumptions about key values are made by all the **MICRO B+™** functions.

POINTER refers to either two or four byte integers, depending on the implementation specifications. Note that it is not advisable to pass a value of zero (0) for pntr since some functions return a zero to indicate that a key value has not been found.

addkey returns the following:

0 if entry already exists in the index file;
1 if successful.

```
POINTER rtriev(keyno,entry)

COUNT keyno;
TEXT *entry;
```

rtriev searches the index file referenced by keyno for an exact instance of the key value pointed to by entry. rtriev returns the associated pointer (see addkey) for the key value if successful, or zero if the key value is not found.

```
POINTER search(keyno,target,entry)

COUNT keyno;
TEXT *target,*entry;
```

search searches the index file referenced by keyno for the first key value greater than or equal to the value pointed to by target. If such a value is found, the string pointed to by entry is set equal to the key value found in the index file and search returns its associated pointer. If no such key value is found in the index, then the first character pointed to by entry is set to '\0', and search returns a zero.

```
POINTER nxtkey(keyno,entry)

COUNT keyno;
TEXT *entry;
```

nxtkey returns the pointer associated with the next key value in the index file referenced by keyno, and the string pointed to by entry is set equal to the next key value. If no next key value exists, then nxtkey returns a zero and the character pointed to by entry is set to '\0'. Note that "next" is taken in the context of increasing key value order. Note also that before the first call to nxtkey will work, a call to either rtriev or search for the same index file must have been made.

```
POINTER prvkey(keyno,entry)

COUNT keyno;
TEXT *entry;
```

prvkey returns the pointer associated with the previous key value in the index file referenced by keyno, and the string pointed to by entry is set equal to the previous key value. If no previous key value exists, then prvkey returns a zero and the character pointed to by entry is set to '\0'. As with nxtkey, prvkey will not work unless a call has been made previously to either rtriev or search for the same index file. Note that calls to rtriev, search, nxtkey, and prvkey can be freely intermixed both for the same index file and for different index files.

prvkey will terminate catastrophically if prdsup != 'y' (see access).

```
COUNT delchk(keyno,entry,pntr)

COUNT keyno;
TEXT *entry;
POINTER pntr;
```

delchk removes the key value pointed to by entry from the index file referenced by keyno. Before the removal is made, the pointer associated with the index entry is compared with pntr. If they do not agree, no removal takes place and delchk returns a value of two (2). If they do agree, delchk returns a one (1) to signal a successful deletion. If the value pointed to by entry is not found in the index, delchk returns a zero (0).

```
POINTER delbld(keyno,entry)

COUNT keyno;
TEXT *entry;
```

delbld behaves like delchk except that the deletion is performed blind - that is no verification of the associated pointer is performed. If the key value pointed to by entry is found in the index, the removal is performed and delbld returns the value of the associated pointer. If the key value is not found, delbld returns a zero.

```
POINTER idxent(keyno)

COUNT keyno;
```

idxent returns the number of key values in the index file referenced by keyno.

```
COUNT trehgt(keyno)

COUNT keyno;
```

trehgt returns the height of the B-Tree stored in the index file referenced by keyno. A value of zero implies an empty tree.

```
COUNT rstrct(keyno)

COUNT keyno;
```

rstrct closes the index file referenced by keyno. If the keyno has not been accessed, rstrct causes no action and returns a value of zero (0). If successful, rstrct returns a value of one (1).

```
VOID intseq(keyno,maxent)

COUNT keyno,maxent;
```

intseq is used to initialize the virgin index file referenced by keyno prior to sequentially adding key values (and their associated pointers) to the index file. access must be called prior intseq.

maxent specifies the maximum number of key values to be loaded into each leaf node of the B-Tree. maxent must be in the range: keyval/2 <= maxent <= keyval (see access).

```
COUNT seqkey(entry,pntr)

TEXT *entry;
POINTER pntr;
```

seqkey adds the key value pointed to by entry to the index file previously referenced in intseq. seqkey returns a one (1) for a successful addition or a zero (0) if the key value is out of order (i.e., less than or equal to any previous entry). Note that calls to seqkey cannot be intermixed for different index files.

```
COUNT bldind()
```

Once all the sequentially ordered key values have been entered via seqkey, a call to bldind must be made to actually construct the B-Tree. bldind returns the height of the resulting B-Tree structure. Note that rstrct must be called to properly close the index file.