CST8130: Data Structures
Using *qsort()*: Interpreting Microsoft Documentation

The documentation shown below is a direct copy of the C manual. I have added more comments and diagrams on the following pages to help explain this documentation.

I have intentionally left out some important details on your copy — details that I will clarify in the lecture. You should be prepared to make notes on this document during the lecture.

qsort

Performs a quick sort.

void qsort(void **base*,
size_t *num*,
size_t *width*,
int (__cdecl **compare* **)(const void** **elem1*, **const void** **elem2* **));**

Required Header

<stdlib.h>

Return Value

None

Parameters

- *base*: Start of target array
- *num*: Array size in elements
- *width*: Element size in bytes
- *compare*: Comparison function
- *elem1*: Pointer to the key for the search
- *elem2*: Pointer to the array element to be compared with the key

Remarks

The **qsort** function implements a quick-sort algorithm to sort an array of *num* elements, each of *width* bytes. The argument *base* is a pointer to the base of the array to be sorted. **qsort** overwrites this array with the sorted elements. The argument *compare* is a pointer to a user-supplied routine that compares two array elements and returns a value specifying their relationship. **qsort** calls the *compare* routine one or more times during the sort, passing pointers to two array elements on each call:

Compare Function

int *compare***((void *)** *elem1***, (void *)** *elem2* **);**

The routine must compare the elements, then return one of the following values:

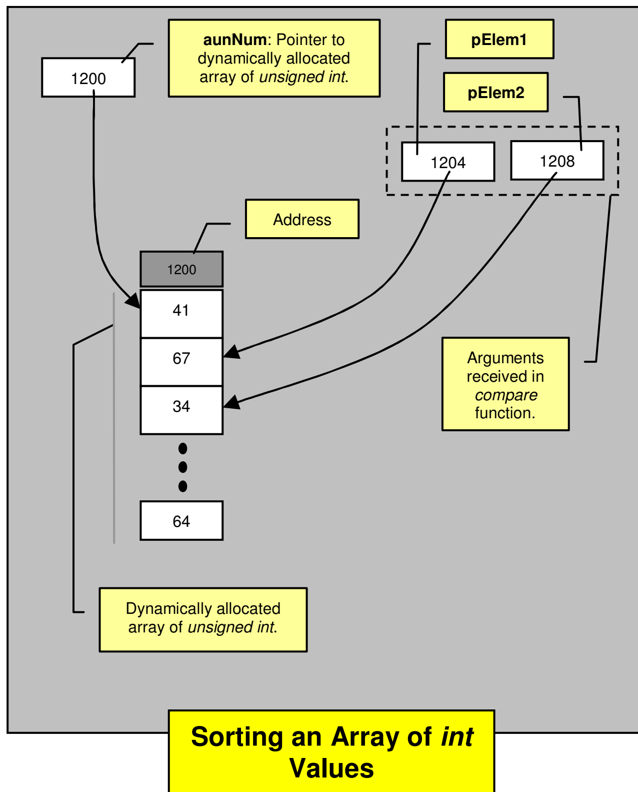
- *< 0* *elem1* less than *elem2*
- *== 0* *elem1* equivalent to *elem2*
- *> 0* *elem1* greater than *elem2*

The array is sorted in increasing order, as defined by the comparison function. To sort an array in decreasing order, reverse the sense of “greater than” and “less than” in the comparison function.

Example 1: Sorting Integers

```
// qsortIntegers.cpp

#include <iostream>
using namespace std;
#include <stdlib.h>

int compare(const void* pElem1, const void* pElem2)
{
return   * (unsigned int*)   pElem1    -    *  (unsigned int*)   pElem2     ;
}

void main()
{
const int ARRAY_SIZE = 10;
unsigned int aunNum[ARRAY_SIZE];

cout << "\n\nUnsorted\n";
for (int i = 0; i < ARRAY_SIZE; ++i) {
      aunNum[i] = rand() % 100;
      cout << aunNum[i] << ' ';
      }

qsort( (void *)aunNum, ARRAY_SIZE, sizeof(unsigned int), compare );

cout << "\n\nSorted\n";
for (i = 0; i < ARRAY_SIZE; ++i)
      cout << aunNum[i] << ' ';
}
```

Example 1: Memory Map: Sorting Integers



Sorting an Array of *int* Values

Example 1 Output

```
[C:\code] qsortIntegers
Unsorted
41 67 34 0 69 24 78 58 62 64
Sorted
0 24 34 41 58 62 64 67 69 78
```

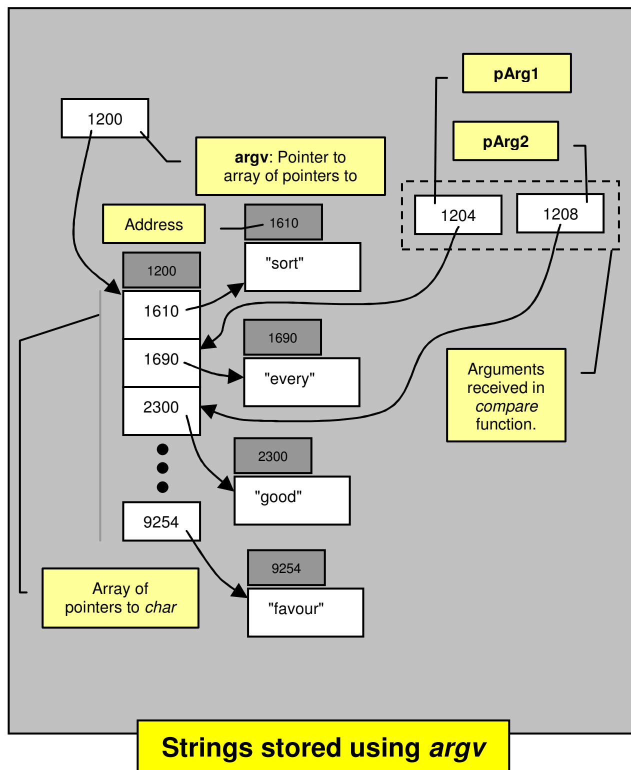
Example 2: Sorting Words Stored Using *char**

```
// SORT.C: Reads command-line parameters. Uses qsort to sort. Displays sorted strings.

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int compare(const void* pArg1, const void* pArg2);

void main( int argc, char** argv )
{
int i;
argv++; // Eliminate argv[0] from sort
argc--;

qsort( (void *)argv, (size_t)argc, sizeof(char*), compare );  // Sort

for( i = 0; i < argc; ++i )   // Output sorted list
      printf( "%s ", argv[i] );
printf( "\n" );
}

int compare( const void* pArg1, const void* pArg2 )
{
return  stricmp  ( * (char**) pArg1,  *  (char**)   pArg2      ) ;   // Compare strings
}
```

Example 2 Output

```
[C:\code] sort every good boy deserves favor
boy deserves every favor good
```

Example 2: Memory Map: Sorting Words Stored Using *char**



Strings stored using *argv*

Using *qsort()* with other Data Types

The *qsort()* library routine can be used to sort any objects that are organized in an array. Shown below, are memory maps comparing two approaches to sorting a list of words: once stored as C++ *string* types; and once stored using pointers to dynamically allocated arrays of char (*char**).

The *string* version is easier to program, but the *char** version will run much faster and have a smaller memory footprint.

Memory Map: Sorting Words Stored as *string* Objects and Using *char**