We now address the problem of what we pass to **bubble()**. We want to make the first parameter of that function a void pointer also. But, that means that within **bubble()** we need to do something about the variable **t**, which is currently an integer. Also, where we use **t = p[j-1];** the type of **p[j-1]** needs to be known in order to know how many bytes to copy to the variable **t** (or whatever we replace **t** with).

Currently, in bubble_4.c, knowledge within **bubble()** as to the type of the data being sorted (and hence the size of each individual element) is obtained from the fact that the first parameter is a pointer to type integer. If we are going to be able to use **bubble()** to sort any type of data, we need to make that pointer a pointer to type **void**. But, in doing so we are going to lose information concerning the size of individual elements within the array. So, in bubble_5.c we will add a separate parameter to handle this size information.

These changes, from bubble4.c to bubble5.c are, perhaps, a bit more extensive than those we have made in the past. So, compare the two modules carefully for differences.

```
/*---------------------- bubble5.c ---------------------------*/

/* Program bubble_5.c from PTRTUT10.HTM    6/13/97 */

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

long arr[10] = { 3,6,1,2,3,8,4,1,7,2};

void bubble(void *p, size_t width, int N);
int compare(void *m, void *n);

int main(void)
{
    int i;
    putchar('\n');

    for (i = 0; i < 10; i++)
    {
        printf("%d ", arr[i]);
    }
    bubble(arr, sizeof(long), 10);
    putchar('\n');

    for (i = 0; i < 10; i++)
    {
        printf("%ld ", arr[i]);
    }

    return 0;
}

void bubble(void *p, size_t width, int N)
{
    int i, j;
    unsigned char buf[4];
    unsigned char *bp = p;
```

```
    for (i = N-1; i >= 0; i--)
    {
        for (j = 1; j <= i; j++)
        {
            if (compare((void *)(bp + width*(j-1)),
                        (void *)(bp + j*width)))  /* 1 */
            {
/*              t = p[j-1];   */
                memcpy(buf, bp + width*(j-1), width);
/*              p[j-1] = p[j];   */
                memcpy(bp + width*(j-1), bp + j*width , width);
/*              p[j] = t;   */
                memcpy(bp + j*width, buf, width);
            }
        }
    }
}

int compare(void *m, void *n)
{
    long *m1, *n1;
    m1 = (long *)m;
    n1 = (long *)n;
    return (*m1 > *n1);
}

/*--------------------- end of bubble5.c ---------------------*/
```

Note that I have changed the data type of the array from **int** to **long** to illustrate the changes needed in the **compare()** function. Within **bubble()** I've done away with the variable **t** (which we would have had to change from type **int** to type **long**). I have added a buffer of size 4 unsigned characters, which is the size needed to hold a long (this will change again in future modifications to this code). The unsigned character pointer ***bp** is used to point to the base of the array to be sorted, i.e. to the first element of that array.

We also had to modify what we passed to **compare()**, and how we do the swapping of elements that the comparison indicates need swapping. Use of **memcpy()** and pointer notation instead of array notation work towards this reduction in type sensitivity.

Again, making a careful comparison of bubble5.c with bubble4.c can result in improved understanding of what is happening and why.

We move now to bubble6.c where we use the same function bubble() that we used in bubble5.c to sort strings instead of long integers. Of course we have to change the comparison function since the means by which strings are compared is different from that by which long integers are compared. And,in bubble6.c we have deleted the lines within **bubble()** that were commented out in bubble5.c.

```
/*--------------------- bubble6.c ---------------------*/
/* Program bubble_6.c from PTRTUT10.HTM   6/13/97 */
```

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

#define MAX_BUF 256

char arr2[5][20] = {  "Mickey Mouse",
                      "Donald Duck",
                      "Minnie Mouse",
                      "Goofy",
                      "Ted Jensen" };

void bubble(void *p, int width, int N);
int compare(void *m, void *n);

int main(void)
{
    int i;
    putchar('\n');

    for (i = 0; i < 5; i++)
    {
        printf("%s\n", arr2[i]);
    }
    bubble(arr2, 20, 5);
    putchar('\n\n');

    for (i = 0; i < 5; i++)
    {
        printf("%s\n", arr2[i]);
    }
    return 0;
}

void bubble(void *p, int width, int N)
{
    int i, j, k;
    unsigned char buf[MAX_BUF];
    unsigned char *bp = p;

    for (i = N-1; i >= 0; i--)
    {
        for (j = 1; j <= i; j++)
        {
          k = compare((void *)(bp + width*(j-1)), (void *)(bp +
j*width));
          if (k > 0)
            {
             memcpy(buf, bp + width*(j-1), width);
             memcpy(bp + width*(j-1), bp + j*width , width);
             memcpy(bp + j*width, buf, width);
            }
        }
    }
}

int compare(void *m, void *n)
```

```
{
    char *m1 = m;
    char *n1 = n;
    return (strcmp(m1,n1));
}

/*------------------- end of bubble6.c ---------------------*/
```

But, the fact that **bubble()** was unchanged from that used in bubble5.c indicates that that function is capable of sorting a wide variety of data types. What is left to do is to pass to **bubble()** the name of the comparison function we want to use so that it can be truly universal. Just as the name of an array is the address of the first element of the array in the data segment, the name of a function decays into the address of that function in the code segment. Thus we need to use a pointer to a function. In this case the comparison function.

Pointers to functions must match the functions pointed to in the number and types of the parameters and the type of the return value. In our case, we declare our function pointer as:

```
int (*fptr)(const void *p1, const void *p2);
```

Note that were we to write:

```
int *fptr(const void *p1, const void *p2);
```

we would have a function prototype for a function which returned a pointer to type **int**. That is because in C the parenthesis () operator have a higher precedence than the pointer * operator. By putting the parenthesis around the string (*fptr) we indicate that we are declaring a function pointer.

We now modify our declaration of **bubble()** by adding, as its 4th parameter, a function pointer of the proper type. It's function prototype becomes:

```
void bubble(void *p, int width, int N,
            int(*fptr)(const void *, const void *));
```

When we call the **bubble()**, we insert the name of the comparison function that we want to use. bubble7.c illustrate how this approach permits the use of the same **bubble()** function for sorting different types of data.

```
/*------------------- bubble7.c ------------------*/

/* Program bubble_7.c from PTRTUT10.HTM  6/10/97 */

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

#define MAX_BUF 256
```

```
long arr[10] = { 3,6,1,2,3,8,4,1,7,2};
char arr2[5][20] = {  "Mickey Mouse",
                      "Donald Duck",
                      "Minnie Mouse",
                      "Goofy",
                      "Ted Jensen" };

void bubble(void *p, int width, int N,
            int(*fptr)(const void *, const void *));
int compare_string(const void *m, const void *n);
int compare_long(const void *m, const void *n);

int main(void)
{
    int i;
    puts("\nBefore Sorting:\n");

    for (i = 0; i < 10; i++)               /* show the long ints */
    {
        printf("%ld ",arr[i]);
    }
    puts("\n");

    for (i = 0; i < 5; i++)                  /* show the strings */
    {
        printf("%s\n", arr2[i]);
    }
    bubble(arr, 4, 10, compare_long);          /* sort the longs */
    bubble(arr2, 20, 5, compare_string);     /* sort the strings */
    puts("\n\nAfter Sorting:\n");

    for (i = 0; i < 10; i++)             /* show the sorted longs */
    {
        printf("%d ",arr[i]);
    }
    puts("\n");

    for (i = 0; i < 5; i++)            /* show the sorted strings */
    {
        printf("%s\n", arr2[i]);
    }
    return 0;
}

void bubble(void *p, int width, int N,
            int(*fptr)(const void *, const void *))
{
    int i, j, k;
    unsigned char buf[MAX_BUF];
    unsigned char *bp = p;

    for (i = N-1; i >= 0; i--)
    {
        for (j = 1; j <= i; j++)
        {
            k = fptr((void *)(bp + width*(j-1)), (void *)(bp +
j*width));
```

```
            if (k > 0)
            {
                memcpy(buf, bp + width*(j-1), width);
                memcpy(bp + width*(j-1), bp + j*width , width);
                memcpy(bp + j*width, buf, width);
            }
        }
    }
}

int compare_string(const void *m, const void *n)
{
    char *m1 = (char *)m;
    char *n1 = (char *)n;
    return (strcmp(m1,n1));
}

int compare_long(const void *m, const void *n)
{
    long *m1, *n1;
    m1 = (long *)m;
    n1 = (long *)n;
    return (*m1 > *n1);
}

/*----------------- end of bubble7.c -----------------*/
```

References for Chapter 10:

1. "Algorithms in C"
   Robert Sedgewick
   Addison-Wesley
   ISBN 0-201-51425-7