Announcements

1. On Wednesday, May 10, we will resume in-person classes

2. I will also hold office hours in person beginning then
   • Until then, same Zoom link for both until then:
     https://ucdavis.zoom.us/j/95840281592?pwd=a1NhNmpLNpFvVrYkpGY3pDcWdlQT09

3. The midterm will be next Friday, May 12
Another Recursive Program: sort.c

• This sorts integers by finding the smallest number and putting it at the beginning

• Basic idea:
  - if number of elements in list is 1 or 0:
    - list is sorted – just return
  - find the smallest number in the list
  - swap it and the first number
  - sort the rest of the list
Problem

- sort.c reads from an array of known length
- User must enter numbers into the program
- The compiler can compute the length (or the user can enter it)

So how do we get around this?
Dynamic Memory Allocation

• Static memory allocation occurs when you declare a variable
  
  ```c
  int num;
  ```

• Compiler creates space for this variable

• There is also a pool of memory (the “heap”) that is available but initially unused

• Dynamic memory occurs when you obtain memory space this pool
  • Allocate: obtaining the space from the pool
  • Allocation: the amount of space you get
  • Deallocate, free: releasing memory that has been allocated; it goes back to the pool
A Useful Operator

• To get the number of bytes in a data type, use sizeof

• Example: on a 32-bit machine:
  • sizeof(char) is 1
  • sizeof(int) is 4
  • sizeof(float) is 4
  • sizeof(double) is 8

• Works for variables, too
  • if a is an int, sizeof(a) is 4
But Be Careful!

```c
char a[100]
```

• Tempting to get the size of an array like this:

```c
sizeof(a)
```

  • Here, a is a pointer constant, so `sizeof` returns the number of bytes in that pointer, *not* the size of the array!

• To get the number of bytes in an array, use

```c
sizeof(a[0]) * 100
```

where 100 is the number of elements in the array

  • The a[0] is one element; works as all elements are of the same type
Allocation Functions: `malloc()`

- Basic function
  
  ```c
  void *malloc(size_t space)
  ```
  
- Allocate `space` bytes of memory, returning its address; returns NULL if not available
  
  - Type `size_t` is same as unsigned int

- Declared void * so that it can be coerced into any type of pointer
  
  ```c
  char *p;
  if ((p = (void *) malloc(100)) == NULL)
  
  error handling
  ```
Allocation Functions: `calloc()`

• Variant

```c
void *calloc(size_t nelt, size_t space)
```

• Like `malloc`, but:
  • Gives you space in terms of elements and size of element, rather than a number of bytes
  • Memory is zeroed out; `malloc()` does not do so, and whatever is in that memory before call to `malloc()` is there once allocated
Allocation Functions: `realloc()`

- Enlargening space already allocated (say `pmem` points to it):
  ```c
  void *realloc(void *pmem, size_t nbytes)
  ```
- This allocates `nbytes` of space, and the contents of `*pmem` are copied into the beginning of the new space
  - The new space may simply extend what `pmem` points to
  - Or, it may be completely new space, in which case what `pmem` points to is deallocated
  - If insufficient memory available, returns NULL and leaves the space `pmem` points to untouched, neither moved nor deallocated
Allocation Functions: `realloc()`

• Common way to use this:

```c
if ((pmem = realloc(pmem, 1000)) == NULL) . . .
```

  • On success, `pmem` now points to a chunk of memory of size 1000 bytes
  • On failure, `pmem` is now NULL — and you lose the address of the memory `pmem` used to point to

• Here's the right way:

```c
  tempptr = realloc(pmem, 1000);
  if (tempptr == NULL) error handling;
  else pmem = tempptr;
```
Deallocation Function: `free()`

- To release memory allocated by one of the allocation functions, use:
  - `void free(void *pmem)`
  - If `pmem` is NULL, this does nothing
  - Do *not* free memory that has already been freed!
    - Called a *double free error* and can often be a vulnerability
    - In all cases, the result is undefined
Another Recursive Program: usort1.c

• Problem with earlier sort.c: numbers are embedded in program
• Better: have users enter the numbers
• Basic idea:
  ask user how many numbers they want sorted
  allocate the space
  read in that many integers – if EOF entered, quit at once