

ECS 36A, May 16, 2025

Sorting

- Function is:

```
void qsort(void *base, size_t nmemb, size_t size,  
          int (*compar)(const void *, const void *));
```

- Here *compar* is function that takes 2 pointers to elements of the array *base*, with *nmemb* members of size *size*
- *compar* returns negative if first is less than second; 0 if the two are equal; and positive if the first is greater than the second
- You supply *compar*

Example `cmp()`

```
int cmp(const void *x, const void *y)
{
    int *px, *py;
    px = (int *) x;
    py = (int *) y;

    return(*px - *py);
}
```

Calling `qsort()`

```
int arr[100];      /* array of integers to be sorted */
int narr;          /* number of integers in arr */
/* ... put random numbers into arr */
/* now sort them */
qsort(arr, narr, sizeof(int),
      (int (*)(const void *, const void *)) cmp);
```

Oops . . .

Remember `qsort()` ? Here is its call:

```
qsort(base, nelts, sizeof(double),  
      (int (*)(const void *, const void *)) cmp);
```

I used this for *cmp*:

```
int cmp(const void *x, const void *y) {  
    double *px, *py;  
    px = (double *)x;  
    py = (double *)y;  
    return(*px - *py);  
}
```

What is wrong with this?

Oops . . .

It's the $*px - *py$ — if it returns something less than 1.0, the function returns 0 (equal), even if there is a difference of (say) 0.5 or -0.5

```
int cmp(const void *x, const void *y){  
    double *px, *py;  
    px = (double *)x;  
    py = (double *)y;  
    if (*px > *py) return(1);  
    else if (*px < *py) return(-1);  
    return(0);  
}
```

The lines in red replace the return in the earlier version

Random Numbers

- `int rand(void)`
 - Generate pseudorandom number between 0 and RAND_MAX inclusive
 - **This function is dangerous — avoid it!!** In older versions, it is *not* pseudorandom in the low order bits. (On newer Linux systems, it's OK)
- `long random(void)`
 - Generate pseudorandom number between 0 and $2^{31}-1$ inclusive
- All require a starting point — called a *seed*

Random Number Seeds

- `void srand(unsigned int seed)`
 - Initialize the `rand()` pseudorandom number generator with *seed*
- `void srandom(unsigned int seed)`
 - Initialize the `random()` pseudorandom number generator with *seed*
- Pick *seed* as randomly as possible
- There are defaults, useful for regenerating the same sequence for debugging
 - `rand/srand` default seed is 1
 - `random/srandom` default seed is 1

Random Numbers

- Linux has a pool of bits generated from sources such as hardware timings and other natural sources that are considered random
 - They are *not* generated by an algorithm as pseudorandom numbers are
- `getrandom(void *buf, size_t sz, unsigned int flags)`
- Generates *sz* random bytes and store them in the given *buf*
- Returns number of bytes stored in *buf*
- Flags:
 - **GRND_NONBLOCK** prevents `getrandom()` from blocking; if it would block it returns `-1` and sets `errno` to **EAGAIN**
 - **GRND_RANDOM** draws from a random source more limited than the one used when this flag is given (avoid using this one)

Example Use

```
unsigned int rnd;
int count;
count = getrandom(&rnd, sizeof(unsigned int), GRND_NONBLOCK);
if (count == -1)
    perror("getrandom");
else
    for(i = 0; i < count; i++)
        printf("0x%02x\n", count, (rnd>>i)&0xff);
```

String Functions

- strcpy, strcat, strcmp, strncpy, strncat, strncmp, strlen
 - You've seen these
- char *strdup(char *s): make a duplicate of string s
 - Space is malloc'ed
- char *strchr(char *s, int c): return pointer to first occurrence of character c in s; NULL if not there
- char *strrchr(char *s, int c): like strchr, but points to last occurrence
- char *strstr(char *s, char *t): like strchr, but looks for first occurrence of string t

String Functions

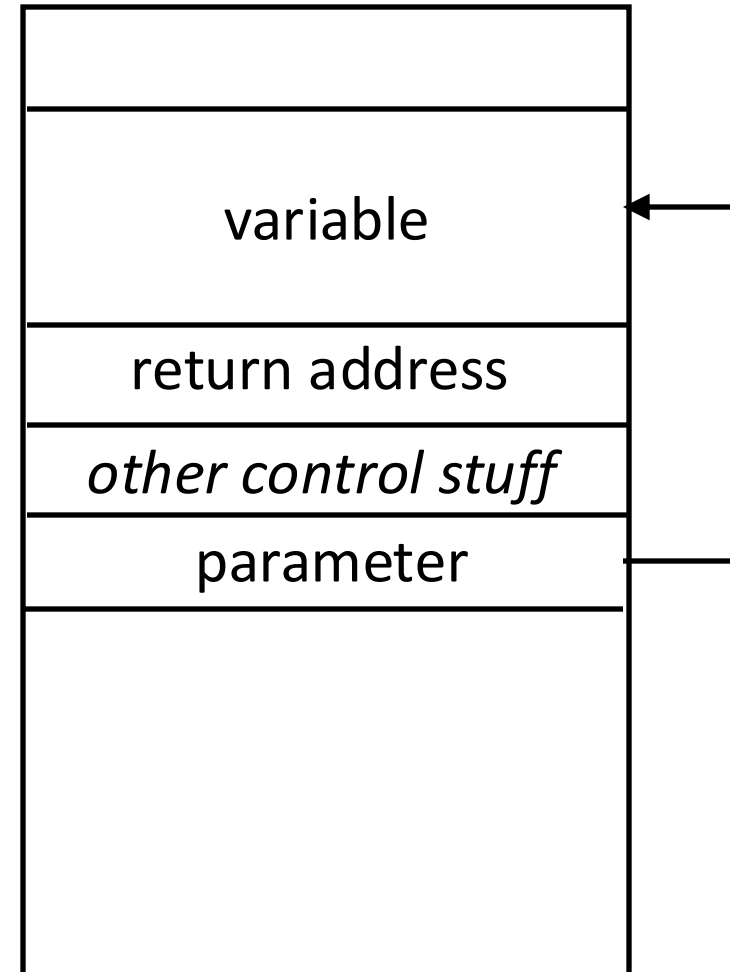
- `char *strtok(char *s, char *delim)`: breaks a string into a sequence of 0 or more nonempty tokens (substrings)
 - On first call, `s` points to string to be parsed
 - On subsequent calls for the same string, set `s` to `NULL`
 - `delim` is a string of characters that delimit tokens
 - `strtok` returns `NULL` when there are no more tokens to return
 - `strtok` *always* returns a nonempty token
 - Warning: `strtok` overwrites delimiters with `'\0'`, so don't give it a read-only string
- `int strcasecmp(char *a, char *b)`: ignore case during comparison

Memory Functions

- `void *memcpy(void *dest, void *src, unsigned int n)`: copy `n` bytes from `src` to `dest`
 - Behavior undefined if `src`, `dest` overlap
- `int memcmp(void *s1, void *s2, unsigned int n)`: compare first `n` bytes of `s1` and `s2`; returns negative, zero, positive depending on whether `s1` is less than, equal to, greater than `s2`

Bug: Stack Smashing

- Problem: failure to check input length
- Going back to the stack, here is what it looks like when a function is called:



The Program bad2.c

```
#include <stdio.h>
char *gets(char *);
int main(void)
{
    int above = 100;
    char input[24];
    int below = 200;
    printf("BEFORE INPUT: above = %#010x; below =  %#010x\n", above, below);
    if (gets(input) == NULL){
        fprintf(stderr, "Unexpected EOF\n");
        return(1);
    }
    printf(" AFTER INPUT: above = %#010x; below =  %#010x\n", above, below);
    return(0);
}
```

A Program Run

- BEFORE INPUT: `above = 0x00000064; below = 0x000000c8`

- `aaaaaaaaaaaaaaaaaaaaaaaaaaaa`

- AFTER INPUT:  `above = 0x00000064; below = 0x00006161`


26 a's (overflowing input by 2 chars)

'a' is represented by the number
0x61 in the computer

May Change Variable Values Unexpectedly

- Here is the stack frame after *gets* is called

