Lecture 10

CIS 208

Wednesday, February 16th, 2005.
Mystery

char *bar(char *s1, char *s2) {
    char *p = s1;
    while (*p++ = *s2++)
    {
    }
    return s1;
}
String Copy from string.h

char *strcpy(char *s1, register const char *s2) {
    register char *p = s1;
    while (*p++ = *s2++)
        ;
    return s1;
}
Malloc/Calloc

- Returns 0 if error
  - Not enough memory error is common

```c
int *p;
if (!p = (int *) calloc(10, sizeof(int)))
    printf("out of memory");
```
Drip Drip Drip Drip

int *p;
p = (int *)malloc(10 * sizeof(p));
//do some stuff here

p = (int *) malloc(20 * sizeof(p));

????
Memory Leaks

- Operating system usually deals after program execution

- Problem for big programs

- Must find a way to get used memory back
void free(void *ptr)

- in stdlib.h
- frees up chunk pointed to by ptr
- destroys all info in chunk
free(void *ptr)

IMPORTANT!!!!!

Only give pointers that were allocated with either malloc or calloc

Do this or else. Seriously. ptr must be exact pointer place given in allocate function
free

int *p;
p = (int *)malloc(10 * sizeof(p));  
//do some stuff
free(p);  // p must still point to old address
p = (int *) malloc(10 * sizeof(p));
Assignment 4

- Due Wednesday, February 23\textsuperscript{th},
- Pointer Practice.
- Check stdlib.h
Sorting

- Returns array in ascending order
- Some methods are better

- Bubble Sort
  - Easy to implement
  - Terrible performance
Bubble Sort

- swap adjacent elements
- worse case is descending order

- Gets really really slow if the array is big
  - $n^2$ algorithm.
Other sorting

- Selection Sort
- Insertion Sort
- Shell Sort
- Quick Sort
Quick sort

- Best sorting solution yet found
- Fairly hard to understand/implement
- C makes it easy.
qsort()

- void qsort(void *buf, size_t num, size_t size,
  int (*compare) (const void *, const void *))

- in stdlib.h
- Sort array in ascending order
- can sort any type of element.
- use on large arrays.
qsort

- buf: pointer to array or buffer to sort
- num: number of elements in array
- size: size of element, in bytes (sizeof)
- compare: pointer to function
  - tells qsort how to sort.
*compare

- given two elements in the array
  if arg1 < arg2: return less than zero
  if arg1 == arg2: return zero;
  if arg1 > arg2: return greater than one;

intcomp(const void *j, const void *k) {
  return *(int *)j - *(int *)j;
}
pointers to functions

- functions have memory address
  - Just like variable
- pointer can point to function

- special declaration for function pointers
  return_type (*name)();
pointers to functions

```c
int foo(int j, char k) {
    ... return 0;
}
```

```c
int (*p)();
p = foo; // no address or star
```

*p now holds the location of foo*
using the pointers

```c
p = foo;

Deference p and give arguments
int bar = (*p)(1, 'a');

Don’t try to free a function pointer
  • Wasn’t malloc/calloc’d
```
back to qsort

```c
int a[] = {4, 5, 2, 1, 3};
int (*p)() = intcomp;
qsort(a, 5, sizeof(int), p);
```
Qsort

might get some warnings on types
may ignore them
cast to fix them

qsor((void *)a, 5, sizeof(int), p);