Lecture 8

CIS 208

Wednesday, February 8th, 2005
A simple model of Memory

- Continuous blocks of memory
- No blank spaces
- Not really like this
  - (so you won’t be tested)
Pointers

- Variables that point to other variables
- Address to memory
- Each pointer has a type
PointerType Variables

- declaration:
  - type *name;

  char *ch;

  int *j;

  int j,k,*p;
Pointer Operators (&)

- & : 'Address of' operator.

```c
int j;
&j is the memory location of j
```

- Like in `scanf()`.
& operator

Pointers are memory locations

```c
int k, *p;
p = &k;
```

p now has the location of k
Pointer Operators (*)

* : ‘at address’ operator

returns value in memory location stored in pointer

called ‘de-referencing’
* operator

Opposite of &

```c
m = &count;
q = *m;
```

q and count now have the same value. Not the same memory location.
* operator

```c
int j = 10, *p;
p = &j;

printf("%d is stored at %p\n", *p, p);
```
Pointer types

- Should always match types
- Can make casts
- Casting can be unpredictable
pointer types

#include <stdio.h>
int main(void) {
    double x = 100.1,y;
    int *p;
    p = (int *) &x;
    y = *p;
    printf("%f", y);
    return 0;
}
Pointer Assignments

```c
int x, *p1, *p2;
p1 = &x;
p1 = p2;
```

- p1 and p2 now both point to x.
**Arithmetic**

- **increment & decrement**
  - `p1++`: points to next possible location
  - actual movement based on type

- `p1 = p1 + 12;`
  - moves to 12\(^{th}\) memory location after `p1`
  - based on type.
Arithmetic

Add and subtract by integers, only.
- *p1, *p2
  p1+p2;
  p1 += 10.2;
  WRONG
  WRONG

No multiply or divide. No modulo.

p1 - p2 : number of elements between p1 and p2.
- must be of same type.

Possible to access bad memory
Arrays and pointers

- **Array variables are pointers**
  - char ch[] == char *ch;
  - ch[5] == *(ch+5);

- **Declaration Issues**
  - Pointer declaration doesn’t create space
Arrays and Pointers

◆ Initialization still works
  ■ char *str = "12345";
  the size of str[] is 6 characters

◆ Pointer arithmetic also applies.
  int j[5];
  j++; j++;
  new j[0] == old j[1];
Array vs Pointers

```c
void putstr(char s[]) {
    int j;
    for (j = 0; s[j]; ++j)
        putchar(s[t]);
}

void putstr(char *s) {
    while (*s) putchar(*s++);
}
```
A note of speed

Compilers like arrays better
- Easier to optimize

```c
for (j = 0; j < N; ++j) {
    a[j] = a[j] * 2000;
    a[j] = a[j] / 10000;
}

b = a;

for (j = 0; j < N; ++j) {
    *b = *b * 2000;
    *b = *b / 10000;
    b++;}
```