CMSC 216
Introduction to Computer Systems
Lecture 6
Pointers & Command Line Arguments
The -> operator

• Dereferencing of a pointer to a structure must occur before accessing a field of the structure; due to precedence, parentheses are needed.

```c
Section s = {101, 25, 1300};
Section *sp = &s;
*sp.num_students += 5;   /* WRONG */
(*sp).num_students += 5; /* RIGHT */
```

• C has a special operator to make this easier:
  
  - "(*sp).num_students" is equivalent to "sp->num_students"
Generic pointers

• Pointers to `void` (variables declared as `void *`) can point to any type
• No casts needed with `void *` pointers
• You can't dereference a `void *` - you first need to cast or assign it to a real pointer type
  – the value obtained from a dereference depends on the type of pointer
• Must be careful to cast correctly
The **const** modifier

- Indicates that a variable can't be changed, and enforced by compiler
  ```c
  int const i = 4;
  const int j = 5;
  i++;  /* ERROR */
  j++;  /* ERROR */
  ```

- Order of type specifier and **const** modifier does matter when dealing with pointers:
  ```c
  int i = 4, j = 5;
  const int *p = &i;  /* pointer to constant int */
  int * const q = &j;  /* constant pointer to int */
  p = &j;    /* OK */
  *p += 5;   /* ERROR */
  q = &i;    /* ERROR */
  *q += 23;  /* OK */
  ```

- The program **cdecl** can be useful for decoding some more complex declarations
Arrays vs. Pointers

• `int nums[4];`
  – declares an array, allocates 4 `ints' worth of space, and points the name `nums` to the beginning of this space
  – `nums` cannot be changed to point elsewhere
  – by itself, `nums` is treated as a constant pointer that points to the beginning of the array

• `int *nump;`
  – declares a pointer, doesn't allocate anything more than space to store an address, connects the name `nump` to that space
  – `nump` can be changed and assigned to
Arrays of pointers

- We can also have an array of pointers:
  ```
  int *nums[60];
  ```
  - an array of 60 pointers to `int`

- This is useful when dealing with arrays of pointers to structures
  - allows us to sort the pointers, without moving around tons of memory

- This is also how the `argv` array is implemented
An array of pointers

char *arg_vector[] = {
    "prog",
    "1",
    "two",
    "THREE",
    "-4",
    NULL
};

• argv looks just like this, including the NULL at the end
Section 13.4, Reek

COMMAND LINE ARGUMENTS
Command line arguments

• When executing a command like "ls -l", or "emacs puzzles.c", the various parts of the command line are called arguments to the program.

• We can access these by including parameters to the main() function.

• These parameters are represented as strings (similar to the String[] args you'd use in Java's main() method).
Accessing command line arguments

• This program (named `prog`) will print out each argument, one per line
  – Note: the type "char *" is also used to represent strings in C - we'll learn why soon

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i;
    for (i = 0; i < argc; i++)
        printf("Arg #\%d: %s\n", i, argv[i]);
    return 0;
}
```

• What if we execute "../prog -l 53 -c -d"?
Accessing command line args, cont.

• If we execute "./prog -l 53 -c -d"?
  – Output is:
    Arg #0: ./prog
    Arg #1: -l
    Arg #2: 53
    Arg #3: -c
    Arg #4: -d
void perror(const char *message);

• prints a description of the most recent error to have occurred (in a system call and some library calls), along with the message you provide (format: "message: error desc.\n")

• For example, say we have an error opening a file; the call perror("Can't open filename.txt") could result in:
  Can't open filename.txt: No such file or directory

• System knows what error occurred by setting the global variable errno (defined in errno.h)
Terminating execution

```c
void exit(int status);
```

– prototype in `stdlib.h`

– "immediately" ends execution when called

– status is viewable by the shell
  
  • `exit(0);` generally means OK
  
  • `exit(1);` or any nonzero generally means error encountered
  
  • can use "`echo $?`" in tcsh to see the exit status of the last program executed

– can use constants `EXIT_SUCCESS` and `EXIT_FAILURE` instead of 0/1