CSE 303 Concepts and Tools for Software Development

Richard C. Davis
UW CSE – 10/20/2006

Lecture 10 – Structs and the Heap

Administravia

- Optional Exercise for Assignment 3
  - Write 4 programs that average numbers
    1. Input on command line, all in `main`
    2. Make input come from stdin
    3. Move computation into function (static array)
    4. Make function take heap allocated array
  - Don't turn this in!

Tip of the Day

- Searching in man pages
  - less (used by man)
    - To search for word, type /word
    - To repeat previous search, type /
    - Backspace to quit
  - Emacs
    - M-x man
    - Search as usual

Last Lecture

- Syntax
  - Type names
  - Arrays
  - Strings
- Command-line Arguments

Today

- Defining New Types
  - Structs
  - Enumerations
- Manual Memory Management
  - The Heap

Java Types vs. C Types

- Java
  - Define enumeration (Java 5)
  - Define class
- C
  - Define enumeration
  - Define struct

Enumerations

- `enum` defines global integer constants
  - Default: first constant has value 0
  - Subsequent constants have increasing values
  - Values of individual constants can be set
- Enumerations are awful
  - No type safety
  - No namespace
  - No pretty printing
- Examples in *enum.c*

C Structs vs. Java Classes

- Similarities
  - Both define a named type
  - Both have <type, name> pairs
- Differences
  - All struct fields "public"
  - Structs not automatically grouped w/functions
  - Structs can be allocated on the stack
    - Don't need "new" (and can't be "null")
- Example in *struct.c*

Struct Syntax

- Defining

```
struct rec {
  int t;
  char *n;
  long x[4];
};
```

- Declaring
  - `struct rec r, *pr;`
- Accessing
  - `long var1 = r.t;`
  - `long var2 = pr->t;`
    - same as `long var2 = (*pr).t;`

Structs in Expressions

- Can be locations (left) or values (right)
  - Field: `r.t`, `pr->t`
  - Whole struct: `r`, `*pr`
- Examples
  - `r.t = pr->t;`
  - `r = (*pr);`
    - `*pr` is temporarily allocated on stack, copied to `r`
  - `(*pr) = r;`
    - `*pr` is location; no temporary allocation

Typedef creates new type name

- Defining structs with typedef

```
typedef struct _rec {
  int t;
  char *n;
} rec;
```

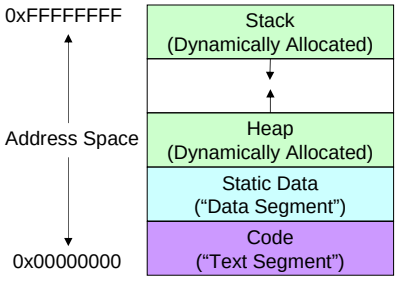
Optional

- Declaring
  - `rec r, *pr;`
- Compare *struct.c* with *structTypedef.c*

Manual Memory Management

- Up to now, all data has been on the stack
  - Allocated when declared
  - Reclaimed when function/block ends
  - Array size is constant (C90 only)
- Heap allocation
  - Allocated with a function call
  - Reclaimed with a function call
  - Array size determined at run time

Remember the Heap?



malloc

- Library function: `malloc`
  - Takes a number
  - Allocates that many bytes
  - Returns a pointer to newly allocated memory
- Behavior
  - returns `NULL` on failure
  - Does not initialize the memory
    - Great for hackers!

Using malloc

- Allocating one object
  - `type *t = (type*)malloc(sizeof(type));`
- Allocating an array of objects
  - `type *t = (type*)malloc(n*sizeof(type));`
- Variants
  - `calloc`: allocate arrays and initialize to 0
  - `realloc`: *attempts* to resize a block

malloc vs. Java's new

- malloc is missing some functionality
  - Fields of structures not initialized
  - No automatic call to a constructor
- But both return a pointer/reference!

Freeing Memory

- Heap allocated objects "live" forever
- Quick way to run out of memory!
- Solutions:
  - Java: garbage collector discards unused stuff
  - C: Explicitly discard using `free`
- Forget to free memory?
  - "Memory leak"

Examples

```
int *p = (int*)malloc(sizeof(int));
p = NULL; //LEAK!!
int *q = (int*)malloc(sizeof(int));
free(q);
free(q); // VERY BAD!!!
int *r = (int*)malloc(sizeof(int));
free(r);
int *s = (int*)malloc(sizeof(int));
free(s);
*s = 19;
*r = 17; // EVEN WORSE!! *s may be 17 !?
```

Memory Management Rules

- For every run-time call to `malloc`
  - Make sure there is a run-time call to `free`
- Burn this into your mind!!!
  - Avoid dangling pointers
  - Avoid memory leaks
- We'll see lots more of this

Summary

- Defining New Types
  - Structs
  - Enumerations
- Manual Memory Management
  - The Heap

Reading

- Programming in C
  - Chapter 9: Structs
  - pp240-244: Pointers and Structures
  - Chapter 14: More on Data Types
  - pp383-388: Dynamic memory allocation

Next Time

- A "real" data type
  - Linked lists