Structures, Pointers, and Memory Allocation in C

Tom Kelliher, CS 311

Jan. 27, 2012

1 Administrivia

Announcements

Assignment

Project 0 handout.

From Last Time

"Grand Tour."

Outline

1. Introduction: structures, pointers, memory allocation.
2. Simple Examples: array, doubly-linked list.

Coming Up

Continued "Grand Tour."

2 Structures

1. Public classes without methods.

2. General form:

```
struct <struct_identifier>
{
   <member_declaration>
   [<member_declaration> ...]
};
/* Don't forget the semicolon!!! */
```

3. Example:

```
#include <stdio.h>

/* "struct dimension" becomes a new type. */

   struct dimension
   {
      double length;
      double width;
      double height;
   };


/* Prototypes */

void printDimension(struct dimension);


int main()
{

   struct dimension box1 = { 1.0, 1.0, 1.0 };
   struct dimension box2;

   box2.length = 2.0;
   box2.width = 4.0;
   box2.height = 6.0;

   printDimension(box2);
```

```
   return 0;
}


void printDimension(struct dimension dim)
{
   printf("Length: %g\nWidth: %g\nHeight: %g\n", dim.length,
          dim.width, dim.height);
}
```

3 Pointers

1. Pointer variables hold the address of another variable.

2. Examples similar to what we've already seen:

```
double data[10];
double *p_data;
int sum;
int *p_sum;

p_data = data;
p_data[3] = 0.0;

p_sum = &sum;
sum = 10;
printf("Sum: %d\n", *p_sum);

*p_sum = 12;   /* Dereference the pointer */;
```

3. What's going on here?

data	1000	
		0.0
...		...
p_data	1050	1000
sum	1054	12
p_sum	1058	1054

4 Dynamic Memory Allocation

1. `malloc()` — memory allocation.

   ```
   void *malloc(size_t size);
   ```

2. `free()` — memory de-allocation.

   ```
   void free(void *ptr)
   ```

Both require `#include <stdlib.h>`.

5 Small Examples

1. A dynamically sized `int` array:

   ```
   void createAndDestroy(int size)
   {
      int *data;
      int i;

      data = (int *) malloc(size * sizeof(int));

      if (data == NULL)
         return;
   ```

```
   for (i = 0; i < size; i++)
      data[i] = 0;

   free((void *) data);
}
```

2. Dynamically allocating and accessing structures:

```
/* Note use of typedef. */
typedef struct qnode {
  struct qnode *next;
  struct qnode *prev;
  int key;
  /* Additional or alternate fields here. */
} qnode;

qnode* head, temp, insert, delete;

/* Don't forget to check malloc's return value! */

head = (qnode*) malloc(sizeof(qnode));
head->next = head->prev = NULL;
head->key = 5;

temp = (qnode*) malloc(sizeof(qnode));
temp->key = 99;
head->next = head->prev = temp;
temp->next = temp->prev = head

temp = (qnode*) malloc(sizeof(qnode));
temp->key = 73;

insert = head;  /* New item inserted AFTER item pointed to by insert. */

temp->next = insert->next;
temp->prev = insert;
insert->next->prev = temp;
insert->next = temp;

delete = head->prev;

/* The following is incorrect, from a memory allocation point-of-view.
 * Why?  How can it be fixed?
```

```
 */

free((void*) delete);
delete->prev->next = delete->next;
delete->next->prev = delete->prev;
```