FILES & DYNAMIC MEMORY ALLOCATION IN C

CSSE 120—Rose Hulman Institute of Technology

Final Exam Facts

- Date: Thursday, May 28, 2009
- Time: 6:00 to 10:00 PM
- Venue: **Olin 257**
- Chapters: Zelle chapters 1 to 12.1, Assigned C readings from Schildt plus Web resources linked from ANGEL Resources page
- You may bring two sheets of paper this time.

Review: Dynamically allocating an array

```
Inventory *createInventory(int count) {
        Inventory *inv;
        inv = (Inventory *)malloc(count*sizeof(Inventory));
        if (inv == NULL) {
                exit(EXIT_FAILURE);
        }
        // initialize
        …
        return inv;
}

int main() {
        Inventory *inv = createInventory(2000);
        …
        free(inv);
}
```

```
typedef struct {
    int itemNumber;
    int quantity;
    double unitPrice;

} Inventory;
```

Expanding or shrinking an array

- What if we wanted to add new items to our inventory? We will need to grow our internal array. How would we do this?
  - malloc another array of the bigger size
  - Then copy the data over to the new array using a loop
  - Free the old array
  - Make **inv** point to the new array.
- Or, use realloc:

```
void *realloc(void *ptr, int amount);
```

Expanding or shrinking an array

```
Inventory *inv;
inv = malloc(oldSize*sizeof(Inventory));
…
// want to resize
inv = (Inventory *) realloc(inv, newSize*sizeof(Inventory));
if (inv == NULL) {
    printf("Allocation error\n");
    exit(EXIT_FAILURE);
}
```

Note that realloc returns a pointer to the new memory! Any idea why?

Frees the block pointed to by inv and allocates a new block of (newSize * **sizeof(Inventory)**) bytes.

Re-alloc demo

- Do you think realloc will move the pointer to a new location on the heap?

- Let's find out…

Expanding or shrinking an array

```
Inventory *inv;
inv = malloc(oldSize*sizeof(Inventory));
…
// want to resize
inv = (Inventory *) realloc(inv, newSize*sizeof(Inventory));
if (inv == NULL) {
    printf("Allocation error\n");
    exit(EXIT_FAILURE);
}
```

Frees the block pointed to by inv and allocates a new block of (newSize * **sizeof(Inventory)**) bytes. The new block contains the contents of the original block up to the lesser of the old and new sizes. Any additional new space is **not** initialized. **The new and original blocks may be at different addresses.**

Using a function to resize an array

```
void resizeInventory(Inventory **inv, int newSize) {
    Inventory *tmp = *inv;
    tmp = (Inventory *)realloc(tmp, newSize*sizeof(Inventory));
    if (tmp == NULL) {
       printf("Allocation error\n");
    }
    *inv = tmp;
}
```

OPTIONAL: Advanced material for those interested. Can you draw a box-and-pointer diagram to illustrate why we need to pass a pointer to a pointer?

```
Inventory *inv = malloc(oldSize(sizeof(Inventory));
resizeInventory(&inv, 3600);
```

Dynamically allocate ***initialized*** memory

- malloc () allocates memory but the memory allocated is NOT initialized
- If some memory was allocated, but not initialized, what *bad* thing could happen?
- An uninitialized value (containing "junk") could be interpreted as an inventory item! Solution: use calloc:

```
void *calloc(int n,  int el_size);
```

Dynamically allocate initialized array

```
Inventory *getInventory(int count) {
    Inventory *inv;
    inv = (Inventory *) calloc(count,  sizeof(Inventory));
    if (inv == NULL) {
       exit(EXIT_FAILURE);
    }
    int i;

    ...

    return inv;
}
```

returns a void pointer (void *) to memory allocated for array of **count** elements. Each element is of size **sizeof(Inventory) bytes.** (Note, **two** arguments.) Returns NULL if fails. Memory is initialized.

vs. `malloc(count * sizeof(Inventory)); // uses single argument`

Q1b

Recap

- Use malloc to dynamically allocate uninitialized memory
- Use calloc to dynamically allocate initialized memory
- Use realloc to dynamically expand or shrink a block of memory

File handling

- Need to include <stdlib.h> to access many file handling functions and macros
- Open a file using **fopen()**
- Modes:
  - "r" (read)
  - "w" (write)
  - "a" (append)
- Returns a file pointer to access the file: **FILE***
- Close a file using **fclose()**

A simple example

```
FILE *inFile;
inFile = fopen("my_file.txt", "r");
if (inFile == NULL) {
   exit(EXIT_FAILURE);
}

// Read data from the file pointed to by inFile

fclose( inFile );
```

How do we read from a file?

- getc(my_fileptr) ; /* read the next character from the file*/
- fgets(buffer, n, my_fileptr);
  /* read the next line of text from file, up to **n-1** chars, into buffer */
- fscanf(my_fileptr, "%d", &num);
  /* read the next int value from file into variable num*/

How do we write to a file?

- putc(c, my_fileptr) ; /* Converts int c to a char and write it to file */
- fputs(my_string, my_fileptr);
  /* Copies my_string to file, except the string terminating char */
- fprintf(my_fileptr, "%s\n", my_string) ;
  /* Similar to printf() except the first parameter is a file pointer */

File Handling

- Check out ***FileDemo*** from your SVN repo
- See problem description in comments
- Work on solving problem for 10 minutes

Keep working on HW27

- See instructions linked from ANGEL
- Due Friday at 11:59 PM

- To get your 10 pts for milestone 1, show your code to your instructor or a TA.

- Work time in class today and during session 29