

DYNAMIC MEMORY ALLOCATION, POINTERS TO STRUCTS

Final Exam Facts

- **Date:** Tuesday, Nov 18, 2008
- **Time:** 6:00 to 10:00 PM
- **Venue:** G219, G220, G221, G222
- **Chapters:** Zelle chapters 1 to 12, Assigned C readings from Schildt plus Web resources linked from ANGEL Resources page
- **Organization:** A paper part and a **computer part (in C only)**, similar to the first 2 exams.
 - ▣ You may bring two sheets of paper this time.

Sample Project for Today

- Check out ***MallocSample*** from your SVN Repository
- Verify that it runs, get help if it doesn't

How large is this?

- sizeof operator: gives the number bytes needed to store a value
- sizeof(char)
- sizeof(int)
- sizeof(double)
- sizeof(char *)
- sizeof(student)
- sizeof(jose)
- printf("size of char is %d bytes.\n", sizeof(char));

```
typedef struct {  
    char *name;  
    int year;  
    double gpa;  
} student;
```

```
char *firstName;  
int terms;  
double scores;  
student jose;
```

Returning Arrays from Functions

- In *maf-main.c*, remove the **exit()** call near the beginning.
- Run the program:
 - ▣ What happens?
 - ▣ Why?
- Original version of **getSamples()** just creates local storage that is recycled when function is done!
- If we want samples to persist, we need to allocate memory using "malloc".
 - ▣ Also need to **#include <stdlib.h>**

Dynamically allocating an array

Cast to desired pointer type

```
double *getSamples(int count) {  
    double *result;  
    result = (double *) malloc(count * sizeof(double));  
    if (result == NULL) {  
        exit(EXIT_FAILURE);  
    }  
    int i;  
    for (i=0; i<count; i++) {  
        result[i] = gaussian(82.5, 7.1);  
    }  
    return result;  
}
```

returns a void pointer (void *) to memory of specified size or NULL if request fails. Memory is uninitialized

Exit program if out of memory or cannot allocate for another reason

Using Dynamically Allocated Array

```
double *sampleA;  
double *sampleB;  
int sampleCount = 5;  
  
sampleA = getSamples(sampleCount);  
sampleB = getSamples(sampleCount);  
  
for (i=0; i<sampleCount; i++) {  
    printf%0.11f\n", sampleA[i] + sampleB[i]);  
}  
  
free(sampleA);  
free(sampleB);
```



Don't forget to free the memory that was previously "malloc-ed".

Recap: sizeof, malloc and free

- `sizeof` operator: gives the number of bytes needed to store a value
- `void *malloc(size_t amount)`: returns a pointer to space for an object of size `amount`, or `NULL` if the request cannot be satisfied. The space is uninitialized.
- `void free(void *p)`: deallocates the space pointed to by `p`; does nothing if `p` is `NULL`. `p` must point to memory that was previously dynamically allocated.

Dynamically allocating strings

- Consider:

```
char *s1 = "Sams shop stocks short spotted socks. ";
```

```
char *s2;
```

- What if we wanted to create a copy of s1 and store it in s2 ?

```
s2 = (char *) malloc((strlen(s1) + 1) * sizeof(char));
```

```
strcpy(s2, s1);
```

- free(s2) when s2 is no longer needed.

Dynamically Allocating Structs

- Can use **malloc** to dynamically allocate **structs**
- We'll use this to create an Array data type soon that's "smarter" than the basic C version
- Will need to use pointers to structs
 - ▣ `student *zeb;`
- Accessing elements of structs is different with pointers...

Pointers to Structs

- Direct reference

```
student debby = {"Deb", 2011, 2.9};  
debby.gpa = 3.2;  
printf("%s, Class of %d\n",  
       debby.name, debby.year);
```

- Use dot when you have the struct directly

- Pointer reference

```
student *aaron;  
aaron = (student *)  
        malloc(sizeof(student));  
aaron->name = "Aaron";  
aaron->year = 2009;  
aaron->gpa = 3.1;  
printf("%s, Class of %d\n",  
       aaron->name, aaron->year);
```

- Use "arrow" when you have a pointer to it

aaron->gpa is shorthand for (*aaron).gpa

Stop, Practice Time

- Problem:
 - ▣ One nice feature of lists in Python is that they "know" their own length
 - ▣ Suppose we want that in C
- Solution:
 - ▣ Make our own Array type and helper functions!
- Homework:
 - ▣ Check out *SmarterArrays* from your SVN repository
 - ▣ See homework description linked from ANGEL