COURSE EVALUATIONS, FINAL EXAM PREP & DYNAMIC MEMORY ALLOCATION

CSSE 120—Rose Hulman Institute of Technology

Course Evaluations

- Course evaluations give us your feedback
- This feedback will be used to improve future offerings of the course and to evaluate me
- We want to make this course the best we can
- Be honest, and please take the time to explain your reasons
- Your responses will be anonymous, and not available to me until after grades are due
- I will leave the room as you begin your evaluation

Evaluation instructions

- Log on to Banner Web
- Click on the "Agreements/Authorizations/Surveys" link
- Find the survey for CSSE120
- Complete the evaluation
- This should take you about 15 minutes
- Let me know when every student has finished

Final Exam Facts

- Date: Monday, November 12, 2007
- Time: 6:00 to 10:00 PM
- Venue: Section 1 (Delvin) O157
  Section 2 (Claude) O167
  Section 3 (Curt) O159
- Chapters: Zelle chapters 1 to 12 & C material (Angel → Course Resources → C Programming)
- Organization: A paper part and a **computer part (in C only)**, just as on the first 2 exams. Same resources allowed.

Possible topics for final exam

- topics for exam 1
- topics for exam 2
- similarities and differences between Python and C
- declaring types
- C's "true" and "false"
- input of numbers with scanf
- passing variables to functions by reference
- arrays
- functional decomposition in C
- declaring and using pointers
- pointer arithmetic
- structs, passing structs to functions by reference
- #define
- typedef
- strings and char
- dynamically allocating arrays, strings

Optional Python Make-up Exam

- When: Session 30, that is, next class
- What: Python programming
- Venue: Normal class room
- Purpose: To give students who want to improve their programming grade for exam 2 an opportunity to do so
- For whom: Students who want to improve their programming grade for exam 2. Other students do not need to attend.

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(student)
- sizeof(firstName)
- 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 reused when function is done!
- If we want samples to persist, we need to allocate memory using "malloc".

Dynamically allocating an array

```
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;
}
```

Typecast to desired pointer type

returns a void pointer (void *) to space of specified size or NULL if request fails. Space 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);

printf("Elt. 2 of A: %0.2lf\n", sampleA[2]);
printf("Elt. 2 of B: %0.2lf\n", *(sampleB + 2));

free(sampleA);
free(sampleB);
```

Recap: sizeof, malloc and free

- sizeof operator: gives the number bytes needed to store a value
- void *malloc(size_t size): returns a pointer to space for an object of size size, 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 be a space previously allocated.

Descriptions from K&R, p. 252

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.

Cousins of malloc

- void *calloc(size_t n, size_t el_size): returns a pointer to contiguous space in memory allocated for an array of n elements, each element requiring el_size bytes. Returns NULL if the request cannot be satisfied. The space is initialized with all bits set to zero.
  - Example: **samples = calloc(count, sizeof(double));**
- void *realloc(void *ptr, size_t size): changes the size of the block pointed to by ptr to size bytes. The content of the space will be unchanged up to the lesser of the old and new size. Any new space is not initialized.
  - Example: **str = realloc(str, strlen(str) + 1);**

When C Gives You More Lemons…

- Problem:
  - Python includes high level functions/methods for strings
  - C (and some other languages) do not
  - What if you need to use C, but also want to dynamically create and mutate strings in high-level ways?
- Solution:
  - Make your own String type and String functions!
- Homework:
  - Check out ***AllocatingStrings*** from your SVN repository
  - See homework description linked from ANGEL

Success on your Finals