CSE 31
Computer Organization

Lecture 6 – C Srings (cont.),
Dynamic Memory Allocation

Announcement

- Labs
  - Lab 2 due this week (with 7 days grace period after due date)
    - Demo is REQUIRED to receive full credit
  - Lab 3 out this week
    - Due at 11:59pm on the same day of your next lab
    - You must demo your submission to your TA within 14 days
- Reading assignment
  - Reading 01 (zyBooks 1.1 – 1.5) due 20-SEP and Reading 02 (zyBooks 2.1 – 2.9) due 27-SEP
    - Complete Participation Activities in each section to receive grade towards Participation
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Announcement

- Homework assignment
  - Homework 01 (zyBooks 1.1 – 1.5) due 27-SEP
    - Complete *Challenge Activities* in each section to receive grade towards Homework
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

C Strings (review)

- A string in C is just an array of characters.

```
char string[] = "abc";
```

- How do you tell how long a string is?
  - Last character is followed by a 0 byte (null terminator)

```
int strlen(char s[])
{
    int n = 0;
    while (s[n] != 0)
    n++;
    return n;
}
```

C Strings Headaches

- One common mistake is to forget to allocate an extra byte for the null terminator.
- More generally, C requires the programmer to manage memory manually (unlike Java or C++).
  - When creating a long string by concatenating several smaller strings, the programmer must ensure there is enough space to store the full string!
  - What if you don't know ahead of time how big your string will be?
    - Buffer overrun security holes!

C String Standard Functions

- `int strlen(char *string);`
  - compute the length of `string`
- `int strcmp(char *str1, char *str2);`
  - return 0 if `str1` and `str2` are identical
  - how is this different from `str1 == str2`?
- `char *strcpy(char *dst, char *src);`
  - copy the contents of string `src` to the memory at `dst`. The caller must ensure that `dst` has enough memory to hold the data to be copied.

Dynamic Memory Allocation (1/4)

- C has operator **`sizeof()`** which gives size in bytes (of type or variable)
- To assume the size of objects can be misleading and is bad style, so use **`sizeof(type)`**
  - Many years ago an `int` was 16 bits, and programs were written with this assumption.
  - What is the size of integers now?
- "**`sizeof`**" knows the size of arrays:

```
int ar[3]; // Or:   int ar[] = {54, 47, 99}
sizeof(ar) →12
```

  - …as well for arrays whose size is determined at run-time:

```
int n = 3;
int ar[n]; // Or: int ar[function_that_returns_3()];
sizeof(ar) →12
```

Dynamic Memory Allocation (2/4)

- To allocate room for something new to point to, use `malloc()` (with the help of a typecast and `sizeof`):

```
ptr = (int *) malloc (sizeof(int));
```

  - Now, `ptr` points to a space somewhere in memory of size `(sizeof(int))` in bytes.
  - `(int *)` simply tells the compiler what will go into that space (called a typecast).
- `malloc` is almost never used for 1 value

```
ptr = (int *) malloc (n*sizeof(int));
```

  - This allocates an array of **`n`** integers.

Dynamic Memory Allocation (3/4)

- Once `malloc()` is called, the memory location can contain garbage, so don't use it until you've initialized it.
- After dynamically allocating space, we must dynamically free it:

```
free(ptr);
```

- Use this command to clean up.
  - Even though the program `frees` all memory on `exit` (or when `main` returns), don't be lazy!
  - You never know when your `main` will get transformed into a subroutine!

Dynamic Memory Allocation (4/4)

- The following two things will cause your program to crash or behave strangely later on, and cause VERY VERY hard to figure out bugs:
  - `free()`ing the same piece of memory twice
  - calling `free()` on something you didn't get back from `malloc()`
- The runtime **does not** check for these mistakes
  - Memory allocation is so performance-critical that there just isn't time to do this
  - The usual result is that you corrupt the memory allocator's internal structure
  - You won't find out until much later on, in a totally unrelated part of your code!