CSCI Lecture 1 Supplement

Review of C

Prelude:

C vs Java

- C and Java have similar syntax
- C is not object-oriented
  - No classes (just like you, after you graduate)
  - `struct` is like a class with only public data
- C is a “lower” level language than Java
  - Manual memory management (`malloc`/`free`)
  - Explicit pointers instead of “references”

Hello World

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Compiling Your Code

- We'll use the GCC C compiler (gcc)
  - `gcc foo.c  #output: a.out (binary)`
  - `gcc –o myprog foo.c  #output: myprog`
  - `gcc –c foo.c  # output: foo.o`
  - `gcc –S foo.c  # output: foo.s`
- Useful options:
  - `-Wall` : output all warnings
  - `-lm` : link in math library (pow, sin, cos, etc)
  - Check gcc's "man page" for full rundown

Types and Operators

- Basic Types:
  - Integers: `char`, `int`, `long`, `long long`
  - Floating Point: `float`, `double`
  - Integer types can be `unsigned` (e.g. `unsigned int`)
  - **No** boolean type (use `int` instead)
- Basic Operations:
  - Arithmetic: `+`, `-`, `*`, `/`
  - Bitwise-ops: `&`, `|`, `^`
  - Logical: `&&`, `||`, `==`, `!=`

Control Structures

- ```
  if (x == 0) { … }
  else if (x == 1) {… }
  else {…}
  ```
- ```
  for (i = 0; i < 10; ++i) { … }
  ```
- ```
  while (x < 10) { … }
  ```
- ```
  switch (x) {
      case 0: ...
      case 1: …
      default: …
  }
  ```

Pointers

- Pointers are probably the most confusing part of C
  - Pointers are memory addresses
  - Similar to references in Java (but explicit!)

```
#include <stdio.h>
int main() {
    int foo = 5;
    int *p = &foo;
    printf("foo = %d!\n", *p);
    return 0;
}
```

What gets printed?

```
int foo = 5;
int bar = 8;
int *p = &foo;
int *q = p;
p = &bar;
printf("%d,%d", *p, *q);
```

A. 5, 5 **B.** 8, 8 **C.** 5, 8 **D.** 8, 5

E. None of the above

Warnings vs. Errors

- C has a lot of behavior that is undefined
  - It will warn you about potential problems…
  - … but your program will still compile and run

You probably wanted a "&" there, right?

```
#include <stdio.h>
int main() {
    int foo = 5;
    int *p = foo;
    printf("foo = %d!\n", *p);
    return 0;
}
```

The NULL Pointer

- NULL is a special pointer
  - Points to “nothing” (address 0)
  - Convention: set all uninitialized pointers to NULL

```
double *p = NULL;
if (p != NULL) { … }
```

- Address 0 is inaccessible so trying to dereference NULL crashes program

```
double *p = NULL;
printf("%f", *p); // segfault!
```

structs

- `structs` are the basis for C data structures (e.g. linked lists, trees, graphs)

```
struct s { // defining a struct
  int val;
  double weight;
};

int main() {
  struct s my_struct; // create struct
  my_struct.val = 5;
  my_struct.weight = 8.6;
}
```

Dynamic Memory Management

- Java/C++ uses “new” keyword to allocate objects dynamically
  - Unused objects are automatically deleted
- C uses `malloc`/`free` to manage memory dynamically

```
int main() {
  struct s *my_s = malloc(sizeof(struct s));
  (*my_s).val = 5; // dereference ptr to my_s
  my_struct->weight = 8.6; // shortcut
  …
  free(my_s); // must free mem when finished
}
```