Programming Gotchas

And some floating point oddities you should know

Today

1. Review
2. Common programming mistakes
3. Infinity, negative infinity, and NaN

Comparing floats

4. Programming: A look at memory and structs

Review

- Computational errors: rounding errors, conversion errors, human errors, formula errors, propagation errors
- Aim to be able to explain and give an example of each

Common programming errors

What does this print? The result should be 0.2 * 1000 which is 200.0

```
int main() {
  float val = 0.2;
  float tot = 0.0;
  int i;
  for (i = 0; i < 1000; i++) {
    tot = tot + val;
  }
  printf("tot is %f", tot);
} //tot is 199.998093
```

Common programming errors

What would happen if the next line in the program was this:

```
if (tot == 200.0) {
```

Common programming errors

What does this print? The result should be 5000.0001

```
int main() {

  float num1 = 5000.0;
  float num2 = 0.0001;
  float result = num1 + num2;

  printf("%f + %f = %f", num1, num2,
result);

}//5000.00000 + 0.00010 = 5000.00000
```

Common programming errors

- The last one was known as **subtractive cancellation**
- Adding a big number to a small number
- Big problems when adding a series of numbers
- Solve it by sorting from smallest to largest, and then add them in that order

GCC Oddities

- Floats can represent infinity, and also “not a number”, or “nan”.
- Sometimes surface when there is a problem with a calculation

int Overflows

```
#include <stdio.h>


int main() {
    unsigned int i = 0xFFFFFFFF;
    i ++;
    printf("i is %d\n", i);
    i ++;
    printf("i is %d\n", i);
}
```

float Overflows

```
#include <stdio.h>
int main(){
    float a = 3.4e38;
    float b = 3.5e38;
    printf("pretty big float: %f\n", a);
    printf("bigger: %f\n", b);
}
```

```
pretty big float:
339999995214436424907732413799364296704.000000|
bigger: inf
```

Positive Infinity

```
#include <stdio.h>
int main() {
    float a = 1.0f / 0.0f;
    printf("%f\n",a);
}

//prints out
//inf
```

Negative Infinity

```
#include <stdio.h>
#include <math.h>

int main() {
    float a = log(0);
    printf("%f\n", a);
}
// prints out
//-inf
```

NaN

```
#include <stdio.h>
int main() {
    float x = 0.0f / 0.0f; // or sqrt(-1)
    printf("%f\n", x);

    if (x != x)
      printf("This value"
             "is not a number.\n");

    return 0;

}

// prints out

// -nan This value is not a number.
```

Testing for a NaN value

```
#include <stdio.h>
int main() {
    //float f = 0.0f / 0.0f;
    float f = 1.0f;
    if (f != f) {
        printf("f is nan.\n");
    }
} // if (f == NAN) DOES NOT WORK!
```

Comparing floats

```
if (a == b) {.....
```

- Can't do this, but we can check with absolute error:

```
if (fabs(a - b) < 0.00001) {....
```

- Another oddity worth knowing:

  IEEE floats are lexicographically ordered

Comparing floats

```
float a,b;
```

if `a < b`, then comparing the bit patterns, gives the same result

```
(*(int*)&a < *(int*)&b)
```

is the same as

```
a < b
```

- Using this you can see there is no float between 1.99999988 and 2.0

Summary

- Common programming mistakes
- Infinity, negative infinity, NaNs in floats
- Briefly comparing floats

Some Revision

- Stack and heap memory
- Structs

Memory

- Doing this uses <u>stack</u> memory:

  ```
  int i;
  ```

  allocates 4 bytes for a number in memory
- Doing this uses <u>heap</u> memory:

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

  also allocates 4 bytes for a number in memory

Structs Revision

```
struct person {
  int age;
  char* name;
  char gender;
};
person andy;
andy.age = 35;
strcpy(andy.name, “andrew”);
andy.gender = 'm';
```

Structs on stack and heap

- The andy struct on the stack:

```
person andy;
```

- The andy struct on the heap:

```
person *andy =
 (person*)malloc(sizeof(person));
```

Accessing members in structs

- For structs on the stack, use the dot operator

  `andy.name`

- For structs on the heap, use the arrow

  `andy->name`

- This is the same as `(*andy).name`, just a shortcut

Passing structs around

```
void printName(person* p) {
    printf("%s\n", p->name);
}
void printName(person p) {
    printf("%s\n", p.name);
}
```

Structs are odd sometimes

```
struct person {
        int age;
        char* name;
        char gender;
        void printName() {
                printf(“%s\n”, name);
        }
};
// ...
andy.printName();
```