CSE 410 Section 6: Objects in C

Objects ain’t nothin’ but a thang

At this point, we’ve covered some of the fundamentals of C and assembly. Now, we’d like to talk at a higher level about the differences between C and Java. Java is all about object-oriented programming, but we haven’t seen any objects in C so far. That’s because C doesn’t really have a notion of objects; the closest thing to an object that we’ve seen is a struct.

Today, we’re going to consider how we might translate a class in Java into equivalent code in C. Using the class definition for Person below, translate it into C using the code skeleton at https://courses.cs.washington.edu/courses/cse410/18sp/sections/person.c.

```java
public class Person {
    int age;
    String name;
    int money;

    public Person(int age, String name, int money) {
        this.age = age;
        this.name = name;
        this.money = money;
    }

    public void printPerson() {
        System.out.println("My name is " + name + " and I am " + age + " old.");
    }

    public void changeName(String new_name) {
        this.name = new_name;
        this.money -= 20;
    }

    public void birthday(int gifted_money) {
        this.age++;
        this.money += gifted_money;
    }

    // A.isRicher(B)
    // Returns true if A has more money than B.
    public boolean isRicher(Person other) {
        return other.money < this.money;
    }
}
```

If you have any issues, ask us!
For reference, here is struct syntax in C.

```c
struct Stuff {
    int value;
};

struct Stuff a; // Creates a Stuff struct with identifier "a"
a.value = 4; // "." is used to access the data of the struct.
printf("a.value=%d\n", a.value); // Outputs "a.value=4".

struct Stuff *p_a = &a; // p_a is a pointer to the struct a.
printf("a.value=%d\n", (*p_a).value); // Outputs "a.value=4".
printf("a.value=%d\n", p_a->value); // Outputs "a.value=4".

// (*a).b is equivalent to a->b
```

Smashing the Virtual Address Space

When we compile our C program it’s converted into binary instructions. When we run it, the instructions are loaded into memory and executed by the CPU.

What happens when we write into the address space where our binary instructions exist?

You will write a program that will attempt to write into the ‘instructions’ area of the virtual address space. Create a C program with the following main:

```c
void main(void)
```

We can access the address of where the main method is loaded into memory... by using main. First, let’s print it out.

```c
printf("Main Address:%08x", &main)
```

We will set a pointer equal to main, dereference it and try to write data.

```c
int *p = &main
```

This syntax creates a pointer and sets it equal to the address of main.