Function arguments by reference

In C++ it is the same principle as in C. Assumingly you're already know pointers and functions, so you are aware of that function arguments are passed by value, which means they are copied in and out of functions. But what if we pass pointers to values instead of the values themselves? This will enable us to give functions control over variables and structures of the parent functions, and not just a copy of them, thus directly reading and writing the original object.

Let's say we want to write a function which increments a number by one, called `addone`. This will not work:

```
void addone(int n) {
    // n is local variable which only exists within the function scope
    n++; // therefore incrementing it has no effect
}

int n;
printf("Before: %d\n", n);
addone(n);
printf("After: %d\n", n);
```

However, this will work:

```
void addone(int *n) {
    // n is a pointer here which point to a memory-adress outside the function scope
    (*n)++; // this will effectively increment the value of n
}

int n;
printf("Before: %d\n", n);
addone(&n);
printf("After: %d\n", n);
```

The difference is that the second version of `addone` receives a pointer to the variable `n` as an argument, and then it can manipulate it, because it knows where it is in the memory.

Notice that when calling the `addone` function, we **must** pass a reference (note the "&"-sign) to the variable `n`, and not the variable itself - this is done so that the function knows the address of the variable, and won't just receive a copy of the variable itself.

Pointers to structures

Let's say we want to create a function which moves a point forward in both `x` and `y` directions, called `move`. Instead of sending two pointers, we can now send only one pointer to the function, of the point structure:

```
void move(point * p) {
    (*p).x++;
    (*p).y++;
}
```

However, if we wish to dereference a structure and access one of it's internal members, we have a shorthand syntax for that, because this operation is widely used in data structures. We can rewrite this function using the following syntax:

```
void move(point * p) {
    p->x++;
    p->y++;
}
```

Exercise

Write a function called `birthday`, which adds one to the `age` of a `person`.

Code

Run | Reset | Solution

```
#include <stdio.h>

typedef struct {
  char * name;
  int age;
} person;

/* function declaration */
void birthday(person * p);

void birthday(person * p){
    p->age++; // This is the same..
    //(*p).age++; // ... as this would be
}

int main() {
  person john;
  john.name = "John";
  john.age=27;

  printf("%s is %d years old.\n", john.name, john.age);
  birthday(&john);
  printf("Happy birthday! %s is now %d years old.\n", john.name, john.age);

  return 0;
}
```

Output

Expected Output

```
John is 27 years old.
Happy birthday! John is now 28 years old.
```