Pre-lab Exercises - Worksheet 9: Miscellaneous C

Even A. Nilsen 29.10.2016

1. Declarations

a) Create a constant pointer pointing to a constant float value. Here we can not change neither `*ptr` nor `ptr`.

b) Create a pointer to a constant pointer to a float.

c) `ptr` is a pointer to a constant volatile pointer to a volatile constant pointer to a constant volatile int.

d) A constant pointer to a void function that takes zero parameters. Because it is a constant pointer, it needs to be initialized.

e) Create an enumeration with three options. `ROCK == 0`, `PAPER == 5` and `SCISSORS == 6`.

f) Declare a function named `order` which takes a constant character pointer as a parameter and either returns enum value `ASC` or `DESC`.

g) Create a union with two fields and name the union `things`. A union is like a struct, but the fields occupy the same memory address. This means that only one of the fields can be set at any one time.

h) Create a struct with multiple fields named `Racer` and typedef it to `Racer`. `const char *model` and `const char *identity` are both pointer to constant strings. `enum {ELECTRIC, PETROL} type` is an enumeration named `type`. The union `stats` contains two structs, `electric` and `petrol` and only one of them can be set at a time. This is done because the Racer's type can be either `ELECTRIC` or `PETROL`. Lastly it has got a field which is a pointer to function taking a pointer to a `Racer` struct as a parameter and returning a `double`.

i) Create a struct and typedef it to `Info`. The struct's fields are all bit fields. That means that you can only store up to a certain amount of bits in each of the `int` values. `unsigned int` means that the values cannot be negative.

2. Bit Manipulation

I am doing these on the assumption that `var` and `n` are both non-negative integers.

a) Shift the bits of `1` `n` bits to the left.

b) Shift the bits of `1 n` bits to the left and invert each bit in the operand.

c) Calculate the bitwise AND of `var` and `1` and assign it to `bit`. Meaning that each resulting bit is `1` if both of the operand bits are `1`.

```
a 01100111
&
b 00101011
= 00100011
```

d) Shift the bits of `var n` bits to the right and calculate the bitwise AND of `var` and `1` and assign it to `bit`.

e) Shift the bits of `1 n` bits to the right and calculate the bitwise OR of `var` and `1` and assign it to `var`. Bitwise OR flips each bit to `1` where one or both of the operand bits are `1`.

```
a 01100111
|
b 00101011
= 01101111
```

f) Shift the bits of `1 n` bits to the right and invert the bits of `1`. Then calculate the bitwise AND of `var` and `1` and assign it to `var`.

g) Shift the bits of `1 n` bits to the right and calculate the bitwise XOR of `var` and `1`. Bitwise XOR flips each bit to `1` where only one of the operand bits is `1`.

```
a 01100111
^
b 00101011
= 01001100
```

3. Random Numbers

a) If the function returns a value that is expected to be within some boundary, I would check if it stays within the boundary I have given. Another way to test if the function is doing what one expects of it, is to call it multiple times with the same seed and check if the values match.

b)
```
void rand_int(time_t seed)
{
    int random;
    random = (rand() % 100) + 1;
    if (random < 50)
    {
        printf("Not so random\n");
    }
    else
    {
        printf("%d\n", (rand() % 100) + 1);
    }
}
```