The GO programming language:

1. There are two different types of assignment statements in GO
   1. =
   2. :=

Please explain what the difference is and why the language might allow this? (10 point)

The use of "=" is the same as in other languages, the assignment operation, but here the variable must be declared in advance, or assigned a value while declaring.  
The use of ":=" is similar to the auto operation in C++, which automatically matches the type of the variable and eliminates the need to declare the type of the variable, i.e. you can use it without declaring it, and just add an extra colon when assigning it.

1. All three of these statements are equivalent:
   1. a := 10
   2. a := 012
   3. a := 0xa

Explain why the language allows for this? (Hint: All three of them are different bases.) (10 point)

In the given question same number is represented in different bases.If we are able to understand why different base-systems are used in our numbering system, then we will also be able to justify that why languages allow different bases.For digital hardware, the natural numbering system is binary (base 2). but expressing anything in binary causes way too many bits to be used, so , it's not practical. hexadecimal allows us to use 4x fewer bits, so, it is very useful. besides no processor uses anything less than 4bits, so, hexadecimal is the most common when expressing numbers in a way that is most suitable for digital hardware.example 1101 1010 is the 8 bit numbef which can be expressed a lot more cleaner as the he equivalent DA.on the other hand, humans are most comfortable with DECIMAL which is base 10, as opposed to HEXADECIMAL which base 16. In DECIMAL , 1101 1010 would ben 218.

1. In the GO programming language you can't declare a float, it must be a float32 or float64.
   1. Why do you think this is done if the data type is not necessary? (10 point)
   2. Why do you think they used float64 instead of double? Think about the lexical analyzer and how it interprets a symbol. (10 point)
2. The number of bits of the data type can be specified directly to facilitate parsing by the lexical analyzer.
3. Make the number of bits of data more intuitive without constructing new data types. The lexical analyzer will classify this symbol as a float, occupying 64 bits.
4. In your own words please explain how the method call f() produces the Fibonacci sequence : (30 points)

package main

import "fmt"

func fib() func() int {

a, b := 0, 1

return func() int {

a, b = b, a+b

return a

}

}

func main() {

f := fib()

// Function calls are evaluated left-to-right.

fmt.Println(f(), f(), f(), f(), f())

}

The output is:

1 1 2 3 5

The go language supports closures. In this program, fib() forms a closure internally so that variables a and b can be used as global variables.

In the first call of f(), variable a is assigned to 0 and b is assigned to 1, then a is assigned to b and b is assigned to a + b. Return a, get 1.

In the second call of f(), a is assigned to 1, b is assigned to 2, return a.

In the third call of f(), a is assigned to 2, b is assigned to 3, return a.

In the fourth call of f(), a is assigned to 3, b is assigned to 5, return a.

In the fifth call of f(), a is assigned to 5, b is assigned to 8, return a.

1. Please explain what the output of this code would be : (30 points)

package main

import "fmt"

func Generate(ch chan<- int) {

for i := 2; ; i++ {

ch <- i

}

}

func Filter(in <-chan int, out chan<- int, prime int) {

for {

i := <-in

if i%prime != 0 {

out <- i

}

}

}

func main() {

ch := make(chan int) // Create a new channel.

go Generate(ch) // Launch Generate goroutine.

for i := 0; i < 15; i++ {

prime := <-ch

fmt.Println(prime)

ch1 := make(chan int)

go Filter(ch, ch1, prime)

ch = ch1

}

}

The output of this code is the first 15 prime numbers.

First, the Generate function generates a set of integers greater than 2 and passes the data into the channel ch.

The filter function takes data from “in” and then passes it to “out” if it is not divisible by “prime”.

In the for loop in the main function

1. Each iteration will create a new “prime”

2. Each iteration will create a new “ch1”,

3. each iteration will create a new concurrent process Filter(in,out chan int, prime int), that is, this type of concurrent process will increase and keep running. Each for iteration adds a Filter(in,out chan int, prime int) process.

Then pass the “out” in the “ch1” channel to the “ch” channel with the ch = ch1 statement.

Each iteration outputs the smallest number produced in the last iteration, and then the rest is passed into the new concurrent in channel of this iteration. That is, each time the number in the output “ch” channel cannot be divided by the number before it, which is the current smallest prime number.

Output

