Recitation 01

Reading code – Algorithms – Makefile

George Mappouras

9/8/2017

Variables and functions

```
int var;
int var1, var2;
int var3 = 5;

int division (int n, int d){

}
```

Scope of variables

```
int division (int n, int d){
        int q = 0;
        …
        return q;
}
```

```
void main(){
        int q;
        printf (“The result of 5/3 is: \n”);
        q = division (5, 3);
        printf (“Q=%d\n”,q);
}
```

What is the scope of ‘n’ and ‘d’?

What about ‘q’?

Is the ‘q’ in **division** and **main** the same variable?

Loops

- While loops

```
while (i < 10){
        i++;
}
```

conditional expressions

- For loops

```
for (int i=0; i<10; i++){
        q++;
}
```

Integer Division Algorithm

How do you find the result of 10/3 ?

Quotient (Q) = 0, Remainder (R) = 0,

Numerator (N) = 10, Denominator (D) = 3

10 – 3 = 7   Q=1, R=7

7 – 3 = 4   Q=2, R=4

4 – 3 = 1   Q=3, R=1

Result: Q=3, R=1

Integer Division Algorithm

Lets right down the algorithm:

Integer Division Algorithm

Lets right down the algorithm:

Inputs: N, D

Integer Division Algorithm

Lets right down the algorithm:

Inputs: N, D

Q=0, R=0

Integer Division Algorithm

Lets right down the algorithm:

Inputs: N, D

```
Q=0, R=0
while N >= D {


}
```

Integer Division Algorithm

Lets right down the algorithm:

Inputs: N, D

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
```

Integer Division Algorithm

Lets right down the algorithm: N/D = ?

Inputs: N, D

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
R = N
```

Integer Division Algorithm

Lets right down the algorithm:

Inputs: N, D

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
R = N
return (Q, R)
```

Integer Division Algorithm

What if N < D. For example N = 2 and D = 5

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
R = N
return (Q, R)
```

Integer Division Algorithm

What if D = 0? For example N = 10, D = 0

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
R = N
return (Q, R)
```

Integer Division Algorithm

What if D = 0? For example N = 10, D = 0

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
R = N
return (Q, R)
```

This algorithm is not correct

Write division in C

```
Q=0, R=0
while N >= D {
        Q = Q + 1
        N = N - D
}
R = N
return Q
```

```
int division(int n, int d){

}
```

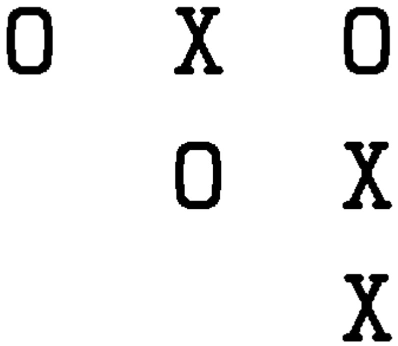
Write division in C

```
int division(int n, int d){
        int q = 0;
        int r = n;
        if (d == 0) return EXIT_FAILURE;
        while (r>=d){
                q++;
                r = r – d;
        }
        return q;
}
```

- Question 2.2 : What does the following code print when it is executed?

```
int main(void) {
  for (int x = 0; x < 3; x++) {
    for (int y = 0; y < 3; y++) {
      if (x-y % 2 ==0) {
        printf(" O ");
      }
      else if (x <= y) {
        printf(" X ");
      }
      else {
        printf("   ");
      }
    }
    printf("\n");
  }
  return EXIT_SUCCESS;
}
```

Answer to Question 2.2

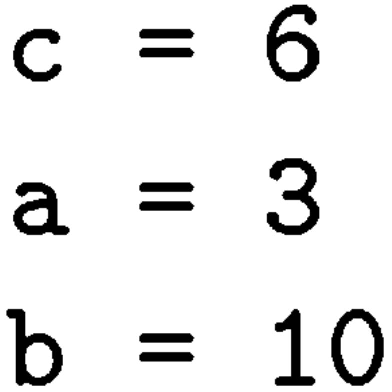


O	X	O
	O	X
		X

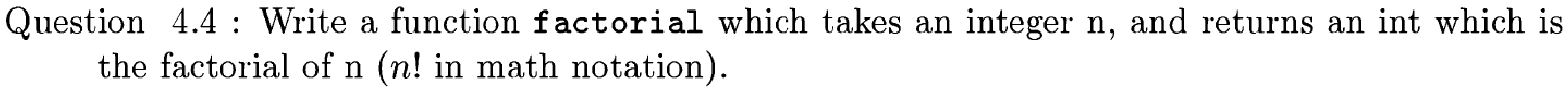
- Question 2.1 : What does the following code print when it is executed?

```
int f (int x, int y) {
  if (x < y) {
    return y - x;
  }
  return x + 5 - y;
}

int main (void) {
  int a = 3;
  int b = 4;
  int c = f (b, a);
  printf("c = %d\n", c);
  a = f(a , c);
  printf("a = %d\n", a);
  b = f(c, f(a, b));
  printf("b = %d\n", b);
  return 0;
}
```

Answer to Question 2.1



```
c = 6
a = 3
b = 10
```



Question 4.4 : Write a function **factorial** which takes an integer n, and returns an int which is the factorial of n ($n!$ in math notation).

Example:

factorial(4) = 1*2*3*4 = 24

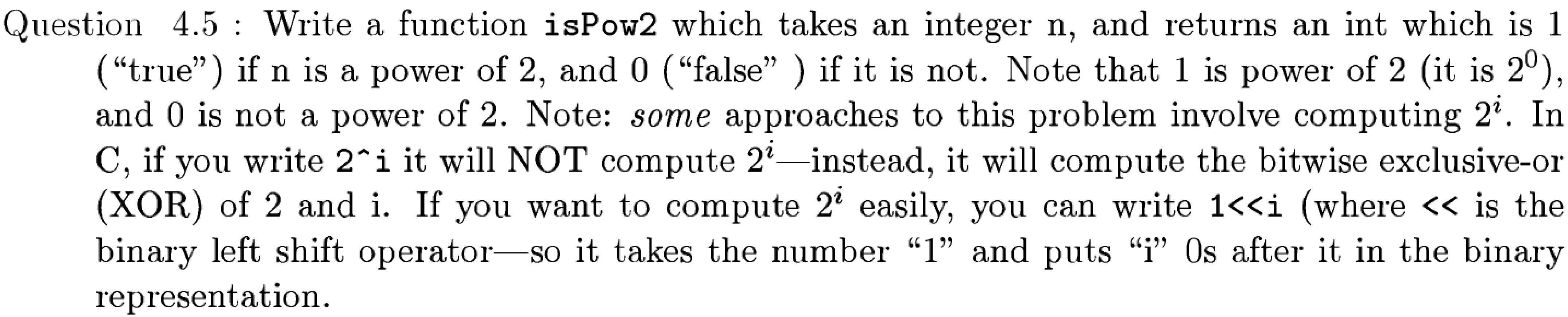
Solution-1 to Question 4.2

```
#include <stdio.h>
unsigned long factorial(int n)
{
    int i;
    unsigned long factorial = 1;

    if (n < 0)
        return EXIT_FAILURE;
```

```
else
    {
        for(i=1; i<=n; ++i)
            factorial *= i;
    }

    return factorial;
}
```

Solution-2 to Question 4.2 (glimpse to the future - recursion)

```
unsigned long recFactorial(int n)
{
    if (n >= 1)
        return n*recFactorial(n-1);
    else
        return 1;
}
```



Question 4.5 : Write a function **isPow2** which takes an integer n, and returns an int which is 1 (“true”) if n is a power of 2, and 0 (“false”) if it is not. Note that 1 is power of 2 (it is 2^0), and 0 is not a power of 2. Note: *some* approaches to this problem involve computing 2^i. In C, if you write `2^i` it will NOT compute 2^i—instead, it will compute the bitwise exclusive-or (XOR) of 2 and i. If you want to compute 2^i easily, you can write `1<<i` (where `<<` is the binary left shift operator—so it takes the number “1” and puts “i” 0s after it in the binary representation.

For example:

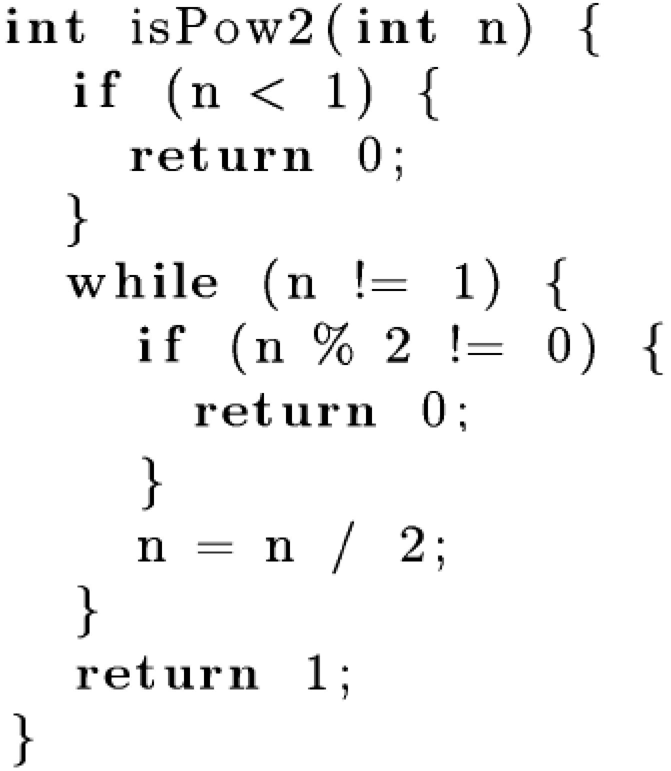
Is 12 a power of 2? -> 2 4 8 16 -> 16 is greater than 12 so no!

Is 32 a power of 2? -> 2 4 8 16 32 -> yes!

Solution 1 to Question 4.5

```
int isPow2(int n){
    int check = 2;
    while (check < n) {
        check = check * 2;
    }
    if (check == n)
        return 1;
    else
        return 0;
}
```

Solution 2 to Question 4.5



```
int isPow2(int n) {
  if (n < 1) {
    return 0;
  }
  while (n != 1) {
    if (n % 2 != 0) {
      return 0;
    }
    n = n / 2;
  }
  return 1;
}
```

Solution 1 vs Solution 2

Any differences between solution 1 and solution 2 of question 4.5?

Is one of them “better” than the other?

Compiling

What is a compiler?

What is GCC?

Compiling

What is a compiler?

A program that has as input our code and output an executable file

What is GCC?

A compiler!

How do I compile my program

```
gm118@ece551: gcc program.c  –o   program
```

How do I compile my program

```
gm118@ece551: gcc program.c  –o   program
```

my C code

the executable file

Flags while compiling

- -o : Define an output name (default is a.out)

- -Wall: Show me all warnings

- -Werror: Treat warnings as errors

Makefiles, object files and headers

- Headers:

avg.c:

```
#include <stdio.h>
#include <stdlib.h>
#include “avg3.h”
float avg3 (int num1, int num2, int num3){
    float result;
    result = num1 + num2 + num3;
    result = result / (float) 3;
    return result;
}
```

avg.h:

```
float avg3 (int num1, int num2, int num4);
```

Makefiles, object files and headers

- Object File:

To create an object file run:

```
gcc –c avg3.c
```

Makefiles, object files and headers

- Compiling the whole program

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "avg3.h"

int main (){
   float result;
   result = avg3(1, 1, 2);
   printf ("Result of avg3(1,1,2) = %f\n",result);
   return 0;
}
```

Makefiles, object files and headers

- Compiling the whole program – Makefile

A Makefile should be named “Makefile”

To run a makefile simply type “make” inside the directory containing the Makefile

To run a specific target of the Makefile run “make target”

Makefiles, object files and headers

- Compiling the whole program – Makefile

A Makefile looks like this:

```
target1: requirments
        instruction_to_execute
target2: requirments
        instruction_to_execute
```

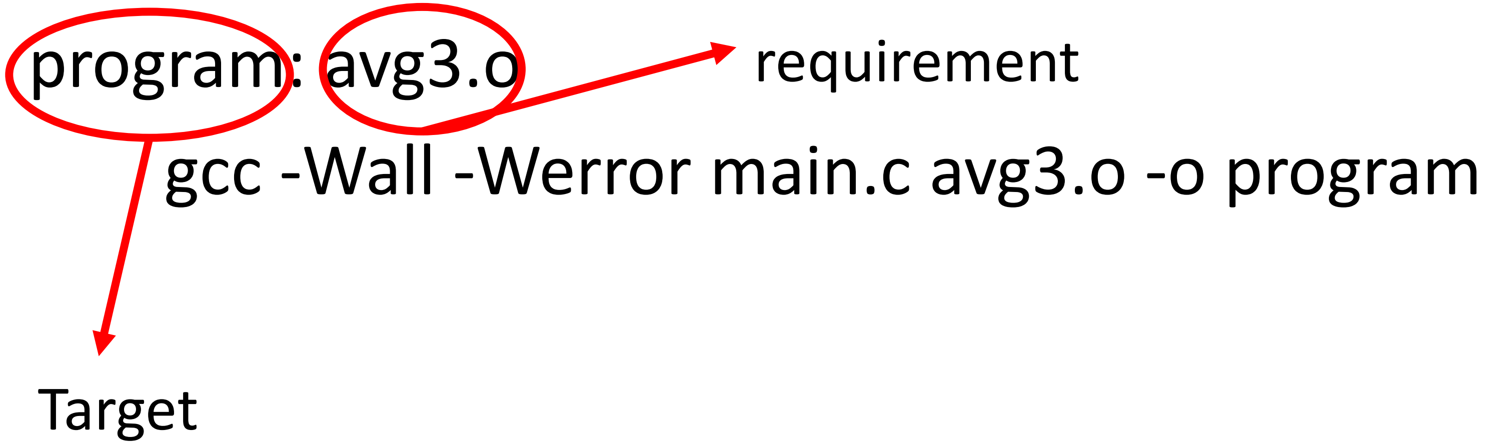
Makefile for avg3

Makefile:

```
program: avg3.o
	gcc -Wall -Werror main.c avg3.o -o program
```

Makefile for avg3

Makefile:



```
program: avg3.o
    gcc -Wall -Werror main.c avg3.o -o program
```

requirement

Target

Makefile for avg3

Makefile:

```
program: avg3.o
	gcc -Wall -Werror main.c avg3.o -o program
avg3.o:
```

Makefile for avg3

Makefile:

```
program: avg3.o
	gcc -Wall -Werror main.c avg3.o -o program
avg3.o:
	  gcc -Wall -Werror -c avg3.c
```

Makefile for avg3

Makefile:

```
program: avg3.o
	gcc -Wall -Werror main.c avg3.o -o program
avg3.o:
	  gcc -Wall -Werror -c avg3.c
clean:
	rm program *.o
```

Makefile for avg3

Makefile:

```
program: avg3.o main.c
	gcc -Wall -Werror main.c avg3.o -o program
avg3.o: avg3.c
	  gcc -Wall -Werror -c avg3.c
clean:
	rm program *.o
```

Makefile for avg3

Modifying those files will cause the compilation to run again

Makefile:

```
program: avg3.o main.c
        gcc -Wall -Werror main.c avg3.o -o program
avg3.o: avg3.c
            gcc -Wall -Werror -c avg3.c
clean:
        rm program *.o
```

Why use a Makefile

Useful for larger programs

The comping procedure can take a LONG time for large programs. With a Makefile we only compile the files that changed.

Makefile is a portable way to compile. No retyping long instructions

Try at home

1) Write a header file “functions.h” that would include the definitions of avg3, factorial and pow2
2) Write a c file “functions.c” that implements the above functions
3) Write a main.c file that calls those functions and prints the results for some inputs of your choice.
4) Write a Makefile that combines the above files to create an executable.
5) Test your Makefile and executable