Homework Two

Q1. Modify the stack implementation in the lecture notes (Stack.h and Stack.c) to implement a stack of integers.

Q2. Write a test program for your stack code in **Q**1 that does the following:

- initialise the stack
- prompt the user to input a number *n*
- check that *n* is a positive number
- prompt the user to input *n* numbers and push each number onto the stack
- use the stack to output the *n* numbers in reverse order

An example of the program executing could be

```
Enter a positive number: 3
Enter a number: 2017
Enter a number: 12
Enter a number: 24
24
12
2017
```

Q3. Modify your program in **Q**2 so that it takes the *n* numbers from the command line. An example of the program execution could be

```
prompt$./tester 2017 12 24
24
12
2017
```

Q4. A stack can be used to convert a positive decimal number *n* to a different numeral system with base *k* according to the following algorithm:

while n>0 **do**
  push *n*%*k* onto the stack
  n = *n* / *k*
end while

The result can be displayed by printing the numbers as they are popped off the stack. Example (*k*=2):

```
n = 13        --> push 1 (= 13%2)
n = 6  (= 13/2) --> push 0 (= 6%2)
n = 3  (=  6/2) --> push 1 (= 3%2)
n = 1  (=  3/2) --> push 1 (= 1%2)
n = 0  (=  1/2)
Result: 1101
```

Using your stack code in Q1, write a C program to implement this algorithm to convert to base *k*=2 a number given on the command line. Design a Makefile to compile this program along with the integer stack implementation.

An example of program compilation and execution could be

```
prompt$ make
gcc -Wall -Werror -c binary.c
gcc -Wall -Werror -c IntStack.c
gcc -o binary binary.o IntStack.o
./binary 13
1101
./binary 128
10000000
./binary 127
1111111
```

Q5. Implement a queue of integers in C using an array to store all the integers. All the function prototypes of the integer queue are defined in IntQueue.h as follows:

```
// Integer queue header file
void queueInit();       // set up an empty queue
int  isEmpty();     // check whether the queue is empty
void enqueue(int);  // insert int at the end of queue
int  dequeue();     // remove int from the front of queue
```

Q6. Given the following definition:

```
int data[12] = {5, 3, 6, 2, 7, 4, 9, 1, 8};
```

and assuming that *&data[0] == 0x10000*, what are the values of the following expressions?

data + 4
*data + 4
*(data + 4)
data[4]
*(data + *(data + 3))
data[data[2]]

Q7. Consider the following piece of C code:

```
typedef struct {
  int   studentID;
  int   age;
  char  gender;
  float WAM;
} PersonT;

PersonT per1;
PersonT per2;
PersonT *ptr;

ptr = &per1;
per1.studentID = 3141592;
ptr->gender = 'M';
ptr = &per2;
ptr->studentID = 2718281;
ptr->gender = 'F';
per1.age = 25;
per2.age = 24;
ptr = &per1;
per2.WAM = 86.0;
ptr->WAM = 72.625;
```

What are the values of the fields in the *per1* and *per2* record after execution of the above statements?

Note that ptr->t means the same as (*ptr).t

Q8. Write a C program that takes 1 command line argument and prints all its *prefixes* in decreasing order of length. You are *not* permitted to use any library functions other than printf(). You are also *not* permitted to use any array other than argv[].

An example of the program execution could be

```
prompt$ ./prefixes Programming
Programming
Programmin
Programmi
Programm
Program
Progra
Progr
Prog
Pro
Pr
P
```