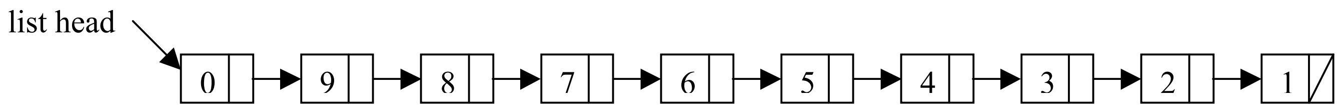
Computer Science I
Program 3: BigIntegers implemented with Linked Lists
Please Check WebCourses2 for the due date

The Problem

The `unsigned int` type in C requires 4 bytes of memory storage. With 4 bytes we can store integers as large as 2^{32}-1; but what if we need bigger integers, for example ones having hundreds of digits? If we want to do arithmetic with such very large numbers we cannot simply use the `unsigned` data type. One way of dealing with this is to use a different storage structure for integers, such as linked lists. If we represent an integer as a linked list, we can represent very large integers, where each digit is a node in the list. Since the integers are allowed to be as large as we like, linked lists will prevent the possibility of overflows in representation. However we need new functions to work with these big integers. In this assignment you'll implement reading in the integers, storing the integers and multiplying the integers. Since you've previously implemented this functionality storing the digits in an array, this time each integer should be represented as a linked list of digits. Once you've completed the assignment, it should be apparent that the internal implementation of a feature may be different even though the functionality provided is the same.

Your program should store each decimal digit (0-9) in a separate node of the linked list. In order to perform addition and subtraction more easily, it is better to store the digits in the list in the reverse order. For instance, the value `1234567890` might be stored as:



Implementation Restrictions

You must use the following struct:

```
struct integer {
     int digit;
     bigint* next;
};

typedef struct integer BigInt;
```

Whenever you store or return a big integer, always make sure not to return it with any leading zeros. Namely, make sure that the last node in the linked list returned does NOT store 0, unless the number stored is 0, in that case a single node should have 0 in it.

Here are the prototypes of the functions you must write:

```
//Preconditions: the first parameter is string that stores
//               only contains digits, doesn't start with
//               0, and is 10000 or fewer characters long.
//Postconditions: The function will read the digits of the
//             large integer character by character,
//             convert them into integers and place them in
//             nodes of a linked list. The pointer to the
//             head of the list is returned.
BigInt* makeBigInt(char* stringInt);

//Preconditions: p is a pointer to a big integer, stored in
//               reverse order, least to most significant
//               digit, with no leading zeros.
//Postconditions: The big integer pointed to by p is
//                printed out.
void printBigInt(BigInt* p);

//Preconditions: p and q are pointers to big integers,
//               stored in reverse order, least to most
//               significant digit, with no leading zeros.
//Postconditions: A new big integer is created that stores
//                the product of the integers pointed to by
//                p and q and a pointer to it is returned.
BigInt* multiply(BigInt* p, BigInt* q);
```

You may write other functions to help you with this task. In fact, you are encouraged to do so. You are free to design the prototypes of these other functions.

Notes:

- **Follow the spec PRECISELY! Including the input file name and program filename**
- **Your program's main function must return a 0.**

Input/Output Specification

Your program will read input from the file, “bigint.txt”, which will specify a number of multiplication operations to carry out between pairs of large integers.

The input file format is as follows:

The first line will contain a single positive integer, *n*, representing the number of operations to carry out. The next *n* lines will contain one problem each. Each of these lines will have two non-negative integers separated by white space. The two integers on the line will be the two operands for the problem. You may assume that these two integers are non-negative, are written with no leading zeros (unless the number itself is 0) and that the number of digits in either of the numbers will never exceed 10000. (Note: Thus, the answer of the operation will never exceed 20000 digits.)

You should generate your output to the screen. In particular, you should generate one line of output per each input case, with the following format:

```
Problem #k: X * Y = Z
```

where k is the problem number, starting at 1, X is the first operand given in the problem, Y is the second operand given in the problem, and Z is the product of the two operands.

Sample Input File

```
3
4 5
27 10
1111111111111111111 4
```

Corresponding Output

```
Problem #1: 4 * 5 = 20
Problem #2: 27 * 10 = 270
Problem #3: 1111111111111111111 * 4 = 4444444444444444444
```

Deliverables

You must turn in one file over WebCourses2:

1) bigintllmult.c, a source file containing your solution to the problem.