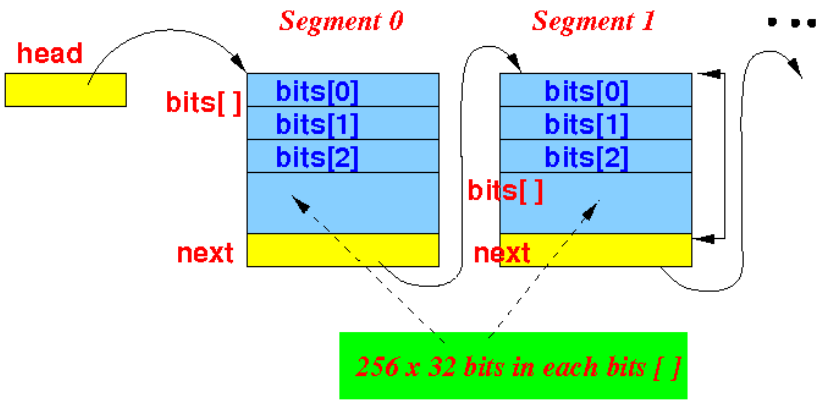
CS 255 **Project 5 – Handling Bit Arrays and Pointers in C**

Due: Thursday, May 1st by 3:00pm. No late submissions!

- **Assignment:**
  - This homework is adapted from the CS450 Assignment #1 that Prof. Mandelberg uses to measure students' C programming ability.
  - Description: complete a C program that takes one command line argument (N) and:
    - Finds all the prime numbers that are <= N
    - Then reads in integers from the terminal and uses the prime numbers found to factor the integer inputs.
- **Provided Files**
  - This project uses the following files, all of which are available in the share/project5 directory.
    - header.h: header file containing the bit array (linked list) structure used in this project
    - main.c the test program that invokes the functions you must write
    - Makefile
    - primes.c: a sample (correct) solution to the last homework which you may use in this project.
- **Data structure used in the project**
  - This project is designed to reinforce your understanding of linked lists, bit manipulation, and pointers in C.
  - The project will still implement the Sieve of Erathothenes, and you will not have to write the algorithm from scratch. Rather you can adjust the solution given in the primes.c file.
  - We will use a different way to represent a bit array than in the previous assignment
    - The data structure used to represent prime numbers in this project is defined in header.h
    - Description of the data structure used in this project:
      - The bit array (used to represent prime numbers) is stored as a linked list (pointed to by the variable head) and each list element contains a bit array of 256 × 32 bits:



      - We refer to each element in this linked list as a "segment" of the bit array.
      - The header.h file contains the definition of the data structure seg which is one

element of the linked list.

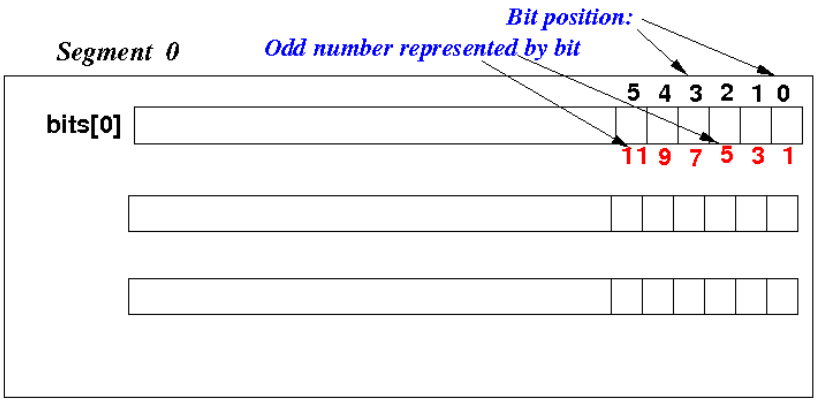
- The main program has code to construct a linked list containing sufficient number of segments to represent numbers <= N. So you don't have to make this list, just use it. (The number N is a command line argument of the main program, just like the previous assignment.)

Here is the program code in the main program that constructs the linked list of segments of bit arrays:

```
/* ========================================================
Allocate the linked list of segments containing pieces
of the bit array
======================================================== */

NSegs = N/(2*BITS_PER_SEG) + 2;      // You need NSegs segments

head = NULL;                           // Empty list

for (i = 0; i < NSegs; i++) {
   p = (  seg *) malloc(sizeof (seg)); // Make a new segment
   p->next = head;                     // Add the next segment to the list
   head = p;
}
```

- **How to use the bits in the segmented bit array to represent prime numbers**
  - In the previous C programming project, we use bit *i* in the bit array to represent whether the number *i* is prime or not prime
  - In this C programming assignment, we will improve the efficiency and use the bit array to represent *odd* numbers *only*. We can do this because we know that no even numbers are prime (this is a trivial case).

    Example:



Explanation:

- The figure above shows the bit array portion in segment 0.
- Bit 0 in array element bits[0] represents the first odd number (1)
- Bit 1 in array element bits[0] represents the second odd number (3)
- Bit 2 in array element bits[0] represents the third odd number (5)

        - Hint: Look for the mathematical pattern between the bit number and the number it represents:
            - 0 => 1
            - 1 => 3
            - 2 => 5
            - j => ??? (This is a crucial understanding in this project.)
- **Summary:**
    - You will write the following functions which are described in detail below. Many of them are similar in purpose but different in structure to the ones your wrote in the previous assignment:
        - clearAllBits
        - sieveOfE
        - testBitIs0
        - setBit
        - countPrimes
        - printPrimes
        - factor
    - Put all your functions inside a file named project5.c. Make sure your project5.c includes the following two lines:

      ```
      #include <stdio.h>
      #include "header.h"
      ```

      Otherwise, your program can't use the correct data type (seg) nor the variable head.

- **Details**

  This section provides a detailed description of the main program and the functions that *you* must provide to complete the program

    - Open main.c and look for the comment:

      ```
      /* ------------------------
          Initialize bit array
      ------------------------ */
      ```
    - Beginning at that comment, the main program does the following: (you do not need to make any changes to it!)
        - The main program first clears out (ie fills with 0's) the segmented bit array (linked list).
            - You must write the `clearAll()` function that clears out the segmented bit array (linked list)
            - The variable `head` points to the first segment --- the `next` field in the last segment has the value `NULL`.
        - Then it finds all the *odd* prime numbers (stores them in the segmented bit array (linked list) that starts at `head`) . This is the sieveOfE function.
            - You may use the posted solution for the previous assignment, but you *must* modify the posted solution appropriately (it does not work without changes - figure out what needs to be changed)

- Hint: The changes are minimal.
- You have to add an if statement around some statements in several places in SieveOfE() because the sieve algorithm can skip over the even numbers. Remember, we're eliminating the trivial case of even numbers.
- Change the parameter to the testBitIs0() function because the meaning of the parameter is changed.

- You do need to re-write the following functions that are used by the `SieveOfE(N)` function:
  - void setBit(int n)
    Note: the input n is the bit position in the segmented bit array. The bit position n represents the number (2*n+1) as given in the figure above!
    1. find the segment that contains the portion of the bit array used (See a programming hint below)
    2. find the array index i of bits[i] in segment
    3. find the bit position pos
    4. set that bit position
  - int testIsBit0(int n)
    Note: the input n is the bit position in the segmented bit array. The bit position n represents the number (2*n+1) as given in the figure above!
    1. find the segment that contains the portion of the bit array used (See a programming hint below)
    2. find the array index i of bits[i] in the segment
    3. find the bit position pos
    4. test if that bit position is 0
- The main program then prints a count of the number of primes found and prints out a list of primes if there are ≤ 100 primes.
  - You must write the `countPrimes(N)` and `printPrimes(N)` functions --- use the primes stored in the segmented bit array (linked list) The variable `head` points to the first segment.
- Finally, the main program reads in a number of integers and calls `factor()` to print out the prime factors for each input.
  - You must write the void factor(int inp) function.
    1. first, factor and print out all the number 2's from the input inp
    2. then use the odd prime numbers stored in the segmented bit array to find and print the odd factors

- **Programming hints**
  - You must first work out how to find these values:
    - The segment number of the segment that stores the bit position representing the number n
    - The index i of the array `bits[i]` within that segment that stores the bit position representing the number n
    - The position pos within the array element `bits[i]` that represents the number n.
  - If necessary, use `printf()` within the `setBit()` and `testBitIs0()` functions to

print out n, the segment number, the array element index and the bit position to help you check the computation.

- You can use the following loop to go to the k^{th} segment:

```
seg *p;
int j;

p = head;

for ( j = 0; j < k; j++)
   p = p->next;
```

  - When the loop ends, the pointer variable p will point to the k^{th} segment in the list.
- To help you debug, you should write a placeholder factor() function initially:

```
void factor( int inp ){}
```

  and test your program. This way, you can make sure that the SieveofE() finds the prime numbers correctly. Use your solution to the previous assignment or the provided primes.c to check. The outputs should be the same!
- DO NOT update the head variable! Otherwise you will lose the linked list!

- Compile and run
  - Use this command to compile:

    make
    - You must have saved a copy of all the provided files and have written the necessary functions in project5.c before you can make. Use placeholder functions if you need to.
  - Use this command to run:

    ./main
- **Debugging**
  - I have set the debug flag (-g) in the Makefile so if you get a core dump, you can use the command

    gdb main

    and run the program to see which statement caused it to crash.
  - Here is a webpage which will outline how to use gdb in detail.
- **Sample output:**

```
main 10000

Number of primes found = 1229
Enter number >> 456
2
2
2
3
19
Enter number (type control-D to end input) >> 12345
3
5
823
Enter number (type control-D to end input) >> control-D
Done
```

  - Note: You should not actually type the string “control-D”. This is meant to indicate you should type the d key while simultaneously holding down the Control key. I just needed some way to indicate this on the sample output.

- **Turn in**

Your grade includes commenting and making your code readable. You should include your collaboration statement at the top of your project5.c file as a comment. Failure to do either of these things will result in a point deduction.

From your project4 directory, issue the following command to submit your work:

```
/home/cs255000/turnin project5.c project5
```

You will see an "allowed" message if your submission was submitted correctly. If you do not see this message, make sure you are issuing the command from your `/home/yourID/cs255/project5` directory and that you do not have any typos in your command. If you are still unsuccessful, contact Dr. Summet or the TA for assistance.