Question 1A

Problem Definition/Specification

A program is needed that will raise a specified number to a specified power. For this implementation of the program, iteration must be used to accomplish this task. See Exercise 5.16 for details.

Problem Analysis

The program is to raise a given number to a given power, and this must be accomplished using loops rather than recursive functions. The number of appropriate iterations of a loop to get the desired output must be determined based on the input exponent value.

Program Inputs

- A base integer
- An exponent integer

Program Outputs

- The result of the base number raised to the input exponent

Formula or Relations

Raising a base number to a power is the same as multiplying the base number by itself the amount of times of its power number.

Algorithm Design

Initial Algorithm

1. Read in the base number from the user
2. Read in the exponent number from the user
3. If the exponent less than or equal to zero display an error message. Otherwise continue
4. Create an integer to keep track of the result and set it equal to base number
5. Loop an amount of times determined by exponent and multiply the result by the base number on each iteration of the loop.
6. Display the result number

Refinement

1. Read in the base as an integer using the scanf command
2. Read in the exponent as an integer using the scanf command
3. Use an if statement to determine whether the exponent is less than or equal to zero and if necessary, display error message: `Invalid input: Exponent must be a positive nonzero integer`

4. Initialize a new integer at the top of the method called “result” and set it equal to the input base number
5. Using a while loop, continue to iterate while exponent > 1. In each iteration, multiply the result by the base integer, and decrement the exponent integer
6. Print out the result number using a printf line

Desk Checking/Test Plan

This program will first be tested using a positive nonzero integer for both the base and the exponent, and they should not cause any errors to occur. A second test will be performed using a number less than or equal to zero for the exponent, which should display an error message if the program is working correctly. A third test will be performed with a number less than or equal to zero for the base, which should NOT cause an error in the program.

Implementation

```c
/*
* Programmer: Kollin Labowski
* Date Completed: September 27, 2020
* Course: CS 350
* Description: This program includes a function integerPower(int a, int b) and the main method to test this function. The program prompts the
* user for two inputs, a base number and an exponent to raise the base number to. This program accomplished the task using loops instead of
* using recursion.
*/

# include <stdio.h>

//Function prototype
int integerPower(int a, int b);

//Main method
void main()
{
    int x, y;

    printf("What is your base number?\n");
    scanf("%d", &x);

    printf("What is your exponent?\n");
    scanf("%d", &y);

    int answer = integerPower(x, y);

    printf("The result of %d^%d is %d", x, y, answer);
}

/*
* This method accepts 2 parameters, a base number and an exponent, and raises the base number to the exponent using a while loop. It returns a
* value of type integer, and returns a -1 when there is an invalid input
*/
int integerPower(int base, int exponent)
{
    if(exponent <= 0)
    {
        printf("Invalid input: Exponent must be a positive nonzero integer");
        return -1;
    }

    int counter = exponent;
    int result = base;
    while(counter > 1)
    {
        result = result * base;
        counter--;
    }

    return result;
}
```

Testing

Test 1:

```
What is your base number?
8
What is your exponent?
4
The result of 8^4 is 4096
```

Test 2:

```
What is your base number?
6
What is your exponent?
-3
Invalid input: Exponent must be a positive nonzero integer
```

Test 3:

```
What is your base number?
-3
What is your exponent?
5
The result of -3^5 is -243
```

The tests showed that the program behaved as was expected

Question 1B

Problem Definition/Specification

A program is needed that will raise a specified number to a specified power. For this implementation of the program, recursion must be used to accomplish this task. See Exercise 5.34 for details.

Problem Analysis

The program is to raise a given number to a given power, and this must be accomplished using a recursive function. The appropriate base case for the recursive function must be determined so that the correct output is produced.

Program Inputs

- A base integer
- An exponent integer

Program Outputs

- The result of the base number raised to the input exponent

Formula or Relations

Raising a base number to a power is the same as multiplying the base number by itself the amount of times of its power number.

Algorithm Design

Initial Algorithm

1. Read in the base number from the user
2. Read in the exponent number from the user
3. If the exponent less than or equal to zero display an error message. Otherwise continue
4. In the function, create base case to determine when to stop recursion
5. Return the recursive result
6. Display the result number

Refinement

1. Read in the base as an integer using the scanf command
2. Read in the exponent as an integer using the scanf command
3. Use an if statement to determine whether the exponent is less than or equal to zero and if necessary, display error message: `Invalid input: Exponent must be a positive nonzero integer`

4. Create an if statement to act as a base case, ending the recursion when the exponent is <= 0
5. Return the base number * a recursive call to the function, so that each time the function is called the base is multiplied again. Decrement exponent in the method so that the base case is eventually reached
6. Print out the result number using a printf line

Desk Checking/Test Plan

This program will first be tested using a positive nonzero integer for both the base and the exponent, and they should not cause any errors to occur. A second test will be performed using a number less than or equal to zero for the exponent, which should display an error message if the program is working correctly. A third test will be performed with a number less than or equal to zero for the base, which should NOT cause an error in the program.

Implementation

```c
/*
* Programmer: Kollin Labowski
* Date Completed: September 27, 2020
* Course: CS 350
* Description: This program includes a function integerPower(int a, int b) and the main method to test this function. The program prompts the
* user for two inputs, a base number and an exponent to raise the base number to. This program accomplished the task using a recursive function.
*/

# include <stdio.h>

//Function prototype
int integerPower(int a, int b);

//Main method
void main()
{
    int x, y;

    printf("What is your base number?\n");
    scanf("%d", &x);

    printf("What is your exponent?\n");
    scanf("%d", &y);

    int answer = -1;
    if(y <= 0)
        printf("Invalid input: Exponent must be a positive nonzero integer");
    else
        answer = integerPower(x, y);

    if(answer != -1)
        printf("The result of %d^%d is %d", x, y, answer);
}

/*
* This method accepts 2 parameters, a base number and an exponent, and raises the base number to the exponent using recursion. It returns a
* value of type integer.
*/
int integerPower(int base, int exponent)
{
    if(exponent <= 0)
        return 1;

    return base * integerPower(base, exponent - 1);
}
```

Testing

Test 1:

```
What is your base number?
6
What is your exponent?
4
The result of 6^4 is 1296
```

Test 2:

```
What is your base number?
4
What is your exponent?
0
Invalid input: Exponent must be a positive nonzero integer
```

Test 3:

```
What is your base number?
-7
What is your exponent?
3
The result of -7^3 is -343
```

The tests showed that the program behaved as was expected

Question 2A1

Answer to Questions on Assignment (Question 2)

Q: How many of these 10,000 numbers do you really have to test before making sure you have found all the primes?

A: Rather than test all n = 10,000 numbers, it is only necessary to test sqrt(n) = 100 numbers. In fact, because you can immediately determine that any even number other than 2 is not a prime number, you only need to test the odd numbers from 1 – 100, or in other words sqrt(n)/2. For even larger values of n, this will be significant in reducing the runtime of the program.

Problem Definition/Specification

A program is needed that will be able to generate all the prime numbers from 1 – 10,000 in an efficient manner. See Exercise 5.27 for more details.

Problem Analysis

The program is to use a function that will determine whether a number is a prime number to print out every number from 1-10,000 that is a prime number. First, it must be considered what makes a number prime and how to determine whether a given number is prime. An efficient way to make this determination is necessary to ensure that the program can run relatively quickly. For this specific implementation of the algorithm, each number that is checked will be checked from 1 to n/2, as specified in the assignment.

Program Inputs

- This program has no inputs

Program Outputs

- Every prime number within the range 1-10,000

Formula or Relations

A prime number is defined as a number which is greater than 1 and which has no factors other than 1 and itself. In addition, because every even number is divisible by 2, 2 is the only even number that is considered a prime number.

Algorithm Design

Initial Algorithm

1. Loop from 1 to 10,000 and test each of the numbers on the function which determines whether a number is prime.

2. Within the function, first test whether the input is a 1, which is nonprime
3. Within the function, test whether the input is a 2, which is prime
4. Within the function, test whether the input is an even number, which is nonprime
5. Within the function, loop through all odd numbers from 3 to the input/2 and test whether the input is divisible by each number, returning the appropriate determination.
6. Within the main method, each number that evaluates as prime will be printed to the screen

Refinement

1. Use a for loop from i = 1 to a constant integer on 10,000 and increment i by 1 in each iteration. Within the loop, use an if statement and call the method int isPrime(int test) on the current value of i.
2. Within the function isPrime(int test), use an if statement with test == 1 to determine if the input is 1, and return 0 (false) if it is.
3. Within the function isPrime(int test), use an if statement with test == 2 to determine if the input is 2, and return 1 (true) if it is.
4. Within the function, use an if statement with test % 2 == 0 to determine whether the number is even (divisible by 2), and return 0 (false) if it is.
5. Within the function, use a for loop that starts at i = 3 and iterates while i <= test/2. Increment i by 2 each iteration, and use an if statement each loop to determine whether the test number is divisible by the current value of i. Return 0 (false) if this is the case, and 1 (true) if the end of the loop is reached and the if statement has not evaluated as true.
6. In the main method, whenever the if statement in step 1 evaluates as true, print out the current value of i to the screen.

Desk Checking/Test Plan

Because there are no inputs to this program, a single run will be enough to determine whether the program works as intended. If the program output all the prime numbers from 1 to 10,000, then it can be determined that the program works correctly. However, an additional test was added that shows the amount of time the program takes to run (using function from the time.h include statement in C). This was important for comparing this program to the other 2 in question 2 of this assignment, and a table comparing the runtimes of the three programs can be seen below all programs in question 2.

Implementation

```c
/*
* Programmer: Kollin Labowski
* Date Completed: September 27, 2020
* Course: CS 350
* Description: This program includes a function isPrime(int a) that determines whether a number is prime or not (more description below), and a main method
* which tests this function on the numbers 1-10000, printing out all of the numbers for which the function evaluates as true. This specific version uses an
* upper bound to check primes of n/2, as stated in the programming assignment.
*/

//Include statements
#include <stdio.h>
#include <math.h>
#include <time.h>

//Function prototype
int isPrime(int a);

//Main method
void main()
{
    clock_t start = clock();//Used for timing the program

    //Variables (including constant 10000 which specifies amount of iterations)
    int format = 0;//Used to format the printed values
    const int LOOPS = 10000;

    for(int i = 1; i <= LOOPS; i++)
    {
        if(isPrime(i))
        {
            printf("%d ", i);
            if(format % 10 == 0)
                printf("\n");
            format++;
        }
    }

    //These methods determine the elapsed time and print it to the terminal
    clock_t end = clock();
    double time = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\n\nTime elapsed: %lf", time);
}
```

```c

 /*
 * This function determines whether a number is prime or not by using loops. It first checks and evaluates the unique cases of 1 and 2 individually. It also
 * is able to determine immediately that a number is not prime by checking if it is even. Otherwise, the function will loop and determine whether the input
 * value is divisible by each odd integer that is <= the square root of the input number. This function uses a loop that terminates at n/2.
 */
 int isPrime(int test)
 {
     //1 evaluates as nonprime
     if(test == 1)
         return 0;

     //2 evaluates as prime
     if(test == 2)
         return 1;

     //All other even numbers evaluate as nonprime
     if(test % 2 == 0)
         return 0;

     //Loop used to determine whether the number is prime if not even or an extraneous case
     for(int i = 3; i <= test/2; i+=2)//The loop checks until n/2 is reached
     {
         if(test % i == 0)
             return 0;
     }
     return 1;
 }
```

Testing

Test 1:

```
2
3 5 7 11 13 17 19 23 29 31
37 41 43 47 53 59 61 67 71 73
79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179
181 191 193 197 199 211 223 227 229 233
239 241 251 257 263 269 271 277 281 283
293 307 311 313 317 331 337 347 349 353
359 367 373 379 383 389 397 401 409 419
421 431 433 439 443 449 457 461 463 467
479 487 491 499 503 509 521 523 541 547
557 563 569 571 577 587 593 599 601 607
613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739
743 751 757 761 769 773 787 797 809 811
821 823 827 829 839 853 857 859 863 877
881 883 887 907 911 919 929 937 941 947
953 967 971 977 983 991 997 1009 1013 1019
1021 1031 1033 1039 1049 1051 1061 1063 1069 1087
1091 1093 1097 1103 1109 1117 1123 1129 1151 1153
1163 1171 1181 1187 1193 1201 1213 1217 1223 1229
1231 1237 1249 1259 1277 1279 1283 1289 1291 1297
1301 1303 1307 1319 1321 1327 1361 1367 1373 1381
1399 1409 1423 1427 1429 1433 1439 1447 1451 1453
1459 1471 1481 1483 1487 1489 1493 1499 1511 1523
1531 1543 1549 1553 1559 1567 1571 1579 1583 1597
1601 1607 1609 1613 1619 1621 1627 1637 1657 1663
1667 1669 1693 1697 1699 1709 1721 1723 1733 1741
1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901
1907 1913 1931 1933 1949 1951 1973 1979 1987 1993
1997 1999 2003 2011 2017 2027 2029 2039 2053 2063
2069 2081 2083 2087 2089 2099 2111 2113 2129 2131
2137 2141 2143 2153 2161 2179 2203 2207 2213 2221
2237 2239 2243 2251 2267 2269 2273 2281 2287 2293
2297 2309 2311 2333 2339 2341 2347 2351 2357 2371
2377 2381 2383 2389 2393 2399 2411 2417 2423 2437
2441 2447 2459 2467 2473 2477 2503 2521 2531 2539
2543 2549 2551 2557 2579 2591 2593 2609 2617 2621
2633 2647 2657 2659 2663 2671 2677 2683 2687 2689
2693 2699 2707 2711 2713 2719 2729 2731 2741 2749
2753 2767 2777 2789 2791 2797 2801 2803 2819 2833
2837 2843 2851 2857 2861 2879 2887 2897 2903 2909
2917 2927 2939 2953 2957 2963 2969 2971 2999 3001
3011 3019 3023 3037 3041 3049 3061 3067 3079 3083
3089 3109 3119 3121 3137 3163 3167 3169 3181 3187
3191 3203 3209 3217 3221 3229 3251 3253 3257 3259
3271 3299 3301 3307 3313 3319 3323 3329 3331 3343
```

```
3347 3359 3361 3371 3373 3389 3391 3407 3413 3433
3449 3457 3461 3463 3467 3469 3491 3499 3511 3517
3527 3529 3533 3539 3541 3547 3557 3559 3571 3581
3583 3593 3607 3613 3617 3623 3631 3637 3643 3659
3671 3673 3677 3691 3697 3701 3709 3719 3727 3733
3739 3761 3767 3769 3779 3793 3797 3803 3821 3823
3833 3847 3851 3853 3863 3877 3881 3889 3907 3911
3917 3919 3923 3929 3931 3943 3947 3967 3989 4001
4003 4007 4013 4019 4021 4027 4049 4051 4057 4073
4079 4091 4093 4099 4111 4127 4129 4133 4139 4153
4157 4159 4177 4201 4211 4217 4219 4229 4231 4241
4243 4253 4259 4261 4271 4273 4283 4289 4297 4327
4337 4339 4349 4357 4363 4373 4391 4397 4409 4421
4423 4441 4447 4451 4457 4463 4481 4483 4493 4507
4513 4517 4519 4523 4547 4549 4561 4567 4583 4591
4597 4603 4621 4637 4639 4643 4649 4651 4657 4663
4673 4679 4691 4703 4721 4723 4729 4733 4751 4759
4783 4787 4789 4793 4799 4801 4813 4817 4831 4861
4871 4877 4889 4903 4909 4919 4931 4933 4937 4943
4951 4957 4967 4969 4973 4987 4993 4999 5003 5009
5011 5021 5023 5039 5051 5059 5077 5081 5087 5099
5101 5107 5113 5119 5147 5153 5167 5171 5179 5189
5197 5209 5227 5231 5233 5237 5261 5273 5279 5281
5297 5303 5309 5323 5333 5347 5351 5381 5387 5393
5399 5407 5413 5417 5419 5431 5437 5441 5443 5449
5471 5477 5479 5483 5501 5503 5507 5519 5521 5527
5531 5557 5563 5569 5573 5581 5591 5623 5639 5641
5647 5651 5653 5657 5659 5669 5683 5689 5693 5701
5711 5717 5737 5741 5743 5749 5779 5783 5791 5801
5807 5813 5821 5827 5839 5843 5849 5851 5857 5861
5867 5869 5879 5881 5897 5903 5923 5927 5939 5953
5981 5987 6007 6011 6029 6037 6043 6047 6053 6067
6073 6079 6089 6091 6101 6113 6121 6131 6133 6143
6151 6163 6173 6197 6199 6203 6211 6217 6221 6229
6247 6257 6263 6269 6271 6277 6287 6299 6301 6311
6317 6323 6329 6337 6343 6353 6359 6361 6367 6373
6379 6389 6397 6421 6427 6449 6451 6469 6473 6481
6491 6521 6529 6547 6551 6553 6563 6569 6571 6577
6581 6599 6607 6619 6637 6653 6659 6661 6673 6679
6689 6691 6701 6703 6709 6719 6733 6737 6761 6763
6779 6781 6791 6793 6803 6823 6827 6829 6833 6841
6857 6863 6869 6871 6883 6899 6907 6911 6917 6947
6949 6959 6961 6967 6971 6977 6983 6991 6997 7001
7013 7019 7027 7039 7043 7057 7069 7079 7103 7109
7121 7127 7129 7151 7159 7177 7187 7193 7207 7211
7213 7219 7229 7237 7243 7247 7253 7283 7297 7307
7309 7321 7331 7333 7349 7351 7369 7393 7411 7417
7433 7451 7457 7459 7477 7481 7487 7489 7499 7507
```

```
7517 7523 7529 7537 7541 7547 7549 7559 7561 7573
7577 7583 7589 7591 7603 7607 7621 7639 7643 7649
7669 7673 7681 7687 7691 7699 7703 7717 7723 7727
7741 7753 7757 7759 7789 7793 7817 7823 7829 7841
7853 7867 7873 7877 7879 7883 7901 7907 7919 7927
7933 7937 7949 7951 7963 7993 8009 8011 8017 8039
8053 8059 8069 8081 8087 8089 8093 8101 8111 8117
8123 8147 8161 8167 8171 8179 8191 8209 8219 8221
8231 8233 8237 8243 8263 8269 8273 8287 8291 8293
8297 8311 8317 8329 8353 8363 8369 8377 8387 8389
8419 8423 8429 8431 8443 8447 8461 8467 8501 8513
8521 8527 8537 8539 8543 8563 8573 8581 8597 8599
8609 8623 8627 8629 8641 8647 8663 8669 8677 8681
8689 8693 8699 8707 8713 8719 8731 8737 8741 8747
8753 8761 8779 8783 8803 8807 8819 8821 8831 8837
8839 8849 8861 8863 8867 8887 8893 8923 8929 8933
8941 8951 8963 8969 8971 8999 9001 9007 9011 9013
9029 9041 9043 9049 9059 9067 9091 9103 9109 9127
9133 9137 9151 9157 9161 9173 9181 9187 9199 9203
9209 9221 9227 9239 9241 9257 9277 9281 9283 9293
9311 9319 9323 9337 9341 9343 9349 9371 9377 9391
9397 9403 9413 9419 9421 9431 9433 9437 9439 9461
9463 9467 9473 9479 9491 9497 9511 9521 9533 9539
9547 9551 9587 9601 9613 9619 9623 9629 9631 9643
9649 9661 9677 9679 9689 9697 9719 9721 9733 9739
9743 9749 9767 9769 9781 9787 9791 9803 9811 9817
9829 9833 9839 9851 9857 9859 9871 9883 9887 9901
9907 9923 9929 9931 9941 9949 9967 9973

Time elapsed: 0.178000
```

Test with size 100,000:

```
99409 99431 99439 99469 99487 99497 99523 99527 99529 99551
99559 99563 99571 99577 99581 99607 99611 99623 99643 99661
99667 99679 99689 99707 99709 99713 99719 99721 99733 99761
99767 99787 99793 99809 99817 99823 99829 99833 99839 99859
99871 99877 99881 99901 99907 99923 99929 99961 99971 99989
99991

Time elapsed: 1.169000
```

Question 2A2

Problem Definition/Specification

A program is needed that will be able to generate all the prime numbers from 1 – 10,000 in an efficient manner. See Exercise 5.27 for more details. This program differs from the previous in that the loop in the function must terminate at sqrt(n) rather than n/2.

Problem Analysis

The program is to use a function that will determine whether a number is a prime number to print out every number from 1-10,000 that is a prime number. First, it must be considered what makes a number prime and how to determine whether a given number is prime. An efficient way to make this determination is necessary to ensure that the program can run relatively quickly. For this specific implementation of the algorithm, each number that is checked will be checked from 1 to sqrt(n), as specified in the assignment.

Program Inputs

- This program has no inputs

Program Outputs

- Every prime number within the range 1-10,000

Formula or Relations

A prime number is defined as a number which is greater than 1 and which has no factors other than 1 and itself. In addition, because every even number is divisible by 2, 2 is the only even number that is considered a prime number.

Algorithm Design

Initial Algorithm

1. Loop from 1 to 10,000 and test each of the numbers on the function which determines whether a number is prime.
2. Within the function, first test whether the input is a 1, which is nonprime
3. Within the function, test whether the input is a 2, which is prime
4. Within the function, test whether the input is an even number, which is nonprime
5. Within the function, loop through all odd numbers from 3 to the sqrt(input) and test whether the input is divisible by each number, returning the appropriate determination.
6. Within the main method, each number that evaluates as prime will be printed to the screen

Refinement

1. Use a for loop from i = 1 to a constant integer on 10,000 and increment i by 1 in each iteration. Within the loop, use an if statement and call the method int isPrime(int test) on the current value of i.
2. Within the function isPrime(int test), use an if statement with test == 1 to determine if the input is 1, and return 0 (false) if it is.
3. Within the function isPrime(int test), use an if statement with test == 2 to determine if the input is 2, and return 1 (true) if it is.
4. Within the function, use an if statement with test % 2 == 0 to determine whether the number is even (divisible by 2), and return 0 (false) if it is.
5. Within the function, use a for loop that starts at i = 3 and iterates while i <= sqrt(test). Increment i by 2 each iteration, and use an if statement each loop to determine whether the test number is divisible by the current value of i. Return 0 (false) if this is the case, and 1 (true) if the end of the loop is reached and the if statement has not evaluated as true.
6. In the main method, whenever the if statement in step 1 evaluates as true, print out the current value of i to the screen.

Desk Checking/Test Plan

Because there are no inputs to this program, a single run will be enough to determine whether the program works as intended. If the program output all the prime numbers from 1 to 10,000, then it can be determined that the program works correctly. However, an additional test was added that shows the amount of time the program takes to run (using function from the time.h include statement in C). This was important for comparing this program to the other 2 in question 2 of this assignment, and a table comparing the runtimes of the three programs can be seen below all programs in question 2.

Implementation

```c
/*
* Programmer: Kollin Labowski
* Date Completed: September 27, 2020
* Course: CS 350
* Description: This program includes a function isPrime(int a) that determines whether a number is prime or not (more description below), and a main method
* which tests this function on the numbers 1-10000, printing out all of the numbers for which the function evaluates as true. This particular version of the
* program uses a loop with size sqrt(n) instead of n/2, which was stated in the problem assignment.
*/

//Include statements
#include <stdio.h>
#include <math.h>
#include <time.h>

//Function prototype
int isPrime(int a);

//Main method
void main()
{
    clock_t start = clock();//Used for timing the program

    //Variables (including constant 10000 which specifies amount of iterations)
    int format = 0;//Used to format the printed values
    const int LOOPS = 10000;

    for(int i = 1; i <= LOOPS; i++)
    {
        if(isPrime(i))
        {
            printf("%d ", i);
            if(format % 10 == 0)
                printf("\n");
            format++;
        }
    }

    //These methods determine the elapsed time and print it to the terminal
    clock_t end = clock();
    double time = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\n\nTime elapsed: %lf", time);
}
```

```c

/*
* This function determines whether a number is prime or not by using loops. It first checks and evaluates the unique cases of 1 and 2 individually. It also
* is able to determine immediately that a number is not prime by checking if it is even. Otherwise, the function will loop and determine whether the input
* value is divisible by each odd integer that is <= the square root of the input number. This function uses a loop that terminates at sqrt(n) instead of n/2.
*/
int isPrime(int test)
{
    //1 evaluates as nonprime
    if(test == 1)
        return 0;

    //2 evaluates as prime
    if(test == 2)
        return 1;

    //All other even numbers evaluate as nonprime
    if(test % 2 == 0)
        return 0;

    //Loop used to determine whether the number is prime if not even or an extraneous case
    for(int i = 3; i <= sqrt(test); i+=2)//The loop checks until sqrt(n) is reached (NOT n/2)
    {
        if(test % i == 0)
            return 0;
    }
    return 1;
}
```

Testing

Test 1:

```
2
3 5 7 11 13 17 19 23 29 31
37 41 43 47 53 59 61 67 71 73
79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179
181 191 193 197 199 211 223 227 229 233
239 241 251 257 263 269 271 277 281 283
293 307 311 313 317 331 337 347 349 353
359 367 373 379 383 389 397 401 409 419
421 431 433 439 443 449 457 461 463 467
479 487 491 499 503 509 521 523 541 547
557 563 569 571 577 587 593 599 601 607
613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739
743 751 757 761 769 773 787 797 809 811
821 823 827 829 839 853 857 859 863 877
881 883 887 907 911 919 929 937 941 947
953 967 971 977 983 991 997 1009 1013 1019
1021 1031 1033 1039 1049 1051 1061 1063 1069 1087
1091 1093 1097 1103 1109 1117 1123 1129 1151 1153
1163 1171 1181 1187 1193 1201 1213 1217 1223 1229
1231 1237 1249 1259 1277 1279 1283 1289 1291 1297
1301 1303 1307 1319 1321 1327 1361 1367 1373 1381
1399 1409 1423 1427 1429 1433 1439 1447 1451 1453
1459 1471 1481 1483 1487 1489 1493 1499 1511 1523
1531 1543 1549 1553 1559 1567 1571 1579 1583 1597
1601 1607 1609 1613 1619 1621 1627 1637 1657 1663
1667 1669 1693 1697 1699 1709 1721 1723 1733 1741
1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901
1907 1913 1931 1933 1949 1951 1973 1979 1987 1993
1997 1999 2003 2011 2017 2027 2029 2039 2053 2063
2069 2081 2083 2087 2089 2099 2111 2113 2129 2131
2137 2141 2143 2153 2161 2179 2203 2207 2213 2221
2237 2239 2243 2251 2267 2269 2273 2281 2287 2293
2297 2309 2311 2333 2339 2341 2347 2351 2357 2371
2377 2381 2383 2389 2393 2399 2411 2417 2423 2437
2441 2447 2459 2467 2473 2477 2503 2521 2531 2539
2543 2549 2551 2557 2579 2591 2593 2609 2617 2621
2633 2647 2657 2659 2663 2671 2677 2683 2687 2689
2693 2699 2707 2711 2713 2719 2729 2731 2741 2749
2753 2767 2777 2789 2791 2797 2801 2803 2819 2833
2837 2843 2851 2857 2861 2879 2887 2897 2903 2909
2917 2927 2939 2953 2957 2963 2969 2971 2999 3001
3011 3019 3023 3037 3041 3049 3061 3067 3079 3083
3089 3109 3119 3121 3137 3163 3167 3169 3181 3187
3191 3203 3209 3217 3221 3229 3251 3253 3257 3259
3271 3299 3301 3307 3313 3319 3323 3329 3331 3343
```

```
3347 3359 3361 3371 3373 3389 3391 3407 3413 3433
3449 3457 3461 3463 3467 3469 3491 3499 3511 3517
3527 3529 3533 3539 3541 3547 3557 3559 3571 3581
3583 3593 3607 3613 3617 3623 3631 3637 3643 3659
3671 3673 3677 3691 3697 3701 3709 3719 3727 3733
3739 3761 3767 3769 3779 3793 3797 3803 3821 3823
3833 3847 3851 3853 3863 3877 3881 3889 3907 3911
3917 3919 3923 3929 3931 3943 3947 3967 3989 4001
4003 4007 4013 4019 4021 4027 4049 4051 4057 4073
4079 4091 4093 4099 4111 4127 4129 4133 4139 4153
4157 4159 4177 4201 4211 4217 4219 4229 4231 4241
4243 4253 4259 4261 4271 4273 4283 4289 4297 4327
4337 4339 4349 4357 4363 4373 4391 4397 4409 4421
4423 4441 4447 4451 4457 4463 4481 4483 4493 4507
4513 4517 4519 4523 4547 4549 4561 4567 4583 4591
4597 4603 4621 4637 4639 4643 4649 4651 4657 4663
4673 4679 4691 4703 4721 4723 4729 4733 4751 4759
4783 4787 4789 4793 4799 4801 4813 4817 4831 4861
4871 4877 4889 4903 4909 4919 4931 4933 4937 4943
4951 4957 4967 4969 4973 4987 4993 4999 5003 5009
5011 5021 5023 5039 5051 5059 5077 5081 5087 5099
5101 5107 5113 5119 5147 5153 5167 5171 5179 5189
5197 5209 5227 5231 5233 5237 5261 5273 5279 5281
5297 5303 5309 5323 5333 5347 5351 5381 5387 5393
5399 5407 5413 5417 5419 5431 5437 5441 5443 5449
5471 5477 5479 5483 5501 5503 5507 5519 5521 5527
5531 5557 5563 5569 5573 5581 5591 5623 5639 5641
5647 5651 5653 5657 5659 5669 5683 5689 5693 5701
5711 5717 5737 5741 5743 5749 5779 5783 5791 5801
5807 5813 5821 5827 5839 5843 5849 5851 5857 5861
5867 5869 5879 5881 5897 5903 5923 5927 5939 5953
5981 5987 6007 6011 6029 6037 6043 6047 6053 6067
6073 6079 6089 6091 6101 6113 6121 6131 6133 6143
6151 6163 6173 6197 6199 6203 6211 6217 6221 6229
6247 6257 6263 6269 6271 6277 6287 6299 6301 6311
6317 6323 6329 6337 6343 6353 6359 6361 6367 6373
6379 6389 6397 6421 6427 6449 6451 6469 6473 6481
6491 6521 6529 6547 6551 6553 6563 6569 6571 6577
6581 6599 6607 6619 6637 6653 6659 6661 6673 6679
6689 6691 6701 6703 6709 6719 6733 6737 6761 6763
6779 6781 6791 6793 6803 6823 6827 6829 6833 6841
6857 6863 6869 6871 6883 6899 6907 6911 6917 6947
6949 6959 6961 6967 6971 6977 6983 6991 6997 7001
7013 7019 7027 7039 7043 7057 7069 7079 7103 7109
7121 7127 7129 7151 7159 7177 7187 7193 7207 7211
7213 7219 7229 7237 7243 7247 7253 7283 7297 7307
7309 7321 7331 7333 7349 7351 7369 7393 7411 7417
7433 7451 7457 7459 7477 7481 7487 7489 7499 7507
```

```
7517 7523 7529 7537 7541 7547 7549 7559 7561 7573
7577 7583 7589 7591 7603 7607 7621 7639 7643 7649
7669 7673 7681 7687 7691 7699 7703 7717 7723 7727
7741 7753 7757 7759 7789 7793 7817 7823 7829 7841
7853 7867 7873 7877 7879 7883 7901 7907 7919 7927
7933 7937 7949 7951 7963 7993 8009 8011 8017 8039
8053 8059 8069 8081 8087 8089 8093 8101 8111 8117
8123 8147 8161 8167 8171 8179 8191 8209 8219 8221
8231 8233 8237 8243 8263 8269 8273 8287 8291 8293
8297 8311 8317 8329 8353 8363 8369 8377 8387 8389
8419 8423 8429 8431 8443 8447 8461 8467 8501 8513
8521 8527 8537 8539 8543 8563 8573 8581 8597 8599
8609 8623 8627 8629 8641 8647 8663 8669 8677 8681
8689 8693 8699 8707 8713 8719 8731 8737 8741 8747
8753 8761 8779 8783 8803 8807 8819 8821 8831 8837
8839 8849 8861 8863 8867 8887 8893 8923 8929 8933
8941 8951 8963 8969 8971 8999 9001 9007 9011 9013
9029 9041 9043 9049 9059 9067 9091 9103 9109 9127
9133 9137 9151 9157 9161 9173 9181 9187 9199 9203
9209 9221 9227 9239 9241 9257 9277 9281 9283 9293
9311 9319 9323 9337 9341 9343 9349 9371 9377 9391
9397 9403 9413 9419 9421 9431 9433 9437 9439 9461
9463 9467 9473 9479 9491 9497 9511 9521 9533 9539
9547 9551 9587 9601 9613 9619 9623 9629 9631 9643
9649 9661 9677 9679 9689 9697 9719 9721 9733 9739
9743 9749 9767 9769 9781 9787 9791 9803 9811 9817
9829 9833 9839 9851 9857 9859 9871 9883 9887 9901
9907 9923 9929 9931 9941 9949 9967 9973

Time elapsed: 0.170000
```

Test time with size 100,000:

```
99559 99563 99571 99577 99581 99607 99611 99623 99643 99661
99667 99679 99689 99707 99709 99713 99719 99721 99733 99761
99767 99787 99793 99809 99817 99823 99829 99833 99839 99859
99871 99877 99881 99901 99907 99923 99929 99961 99971 99989
99991

Time elapsed: 0.744000
```

Question 2B

Problem Definition/Specification

A program is needed that will be able to generate all the prime numbers from 1 – 10,000 in an efficient manner using a method called “The Sieve of Eratosthenes”. See Exercise 6.30 for more details. This program accomplishes the same task as the previous 2 but in a slightly different way that takes advantage of the helpful properties of arrays.

Problem Analysis

The program is to use a function that will determine whether a number is a prime number to print out every number from 1-10,000 that is a prime number. First, it must be considered what makes a number prime and how to determine whether a given number is prime. An efficient way to make this determination is necessary to ensure that the program can run relatively quickly. For this specific implementation of the algorithm, an array is to be created of size 10,000 and each of its entries are to be set to 1. Entries will be set to 0 throughout the program’s runtime, and this will indicate that those specific entries are nonprime.

Program Inputs

- This program has no inputs

Program Outputs

- Every prime number within the range 1-10,000

Formula or Relations

A prime number is defined as a number which is greater than 1 and which has no factors other than 1 and itself.

Algorithm Design

Initial Algorithm

1. Create an array of size 10,000 and set all its entries to 1
2. Begin to loop through the array, beginning at the index corresponding to the number 2, and for every entry in the array that is a 1, print out the number corresponding to that index and proceed to step 3.
3. In the case a 1 is reached while looping through the array, loop through the remainder of the array and set all multiples of that index’s corresponding number to 0.

Refinement

1. Initialize an array as follows: int array[ARRSIZE] where ARRSIZE is a constant integer of size 10,000. Set all entries to 1 by using a for loop and setting each entry at the loop's current index to 1.
2. In a new for loop from 1 to ARRSIZE, check each element to see if it is equal to 1 using an if statement with array[i] == 1. If the if statement evaluates as true, print out the current index of the loop + 1, because in this implementation index 0 corresponds to 1, 1 corresponds to 2, and so on.
3. While the if statement is true, loop through the remainder of the array using a nested for loop from the outer loop's index to the end of the array. Set each multiple of the outer loop's index to 0 by incrementing the inner loop by the outer loop index + 1, again because of the offset for the array indices.

Desk Checking/Test Plan

Because there are no inputs to this program, a single run will be enough to determine whether the program works as intended. If the program output all the prime numbers from 1 to 10,000, then it can be determined that the program works correctly. However, an additional test was added that shows the amount of time the program takes to run (using function from the time.h include statement in C). This was important for comparing this program to the other 2 in question 2 of this assignment, and a table comparing the runtimes of the three programs can be seen below all programs in question 2.

Implementation

```c
/*
* Programmer: Kollin Labowski
* Date Completed: September 28, 2020
* Course: CS 350
* Description: This program generates all of the prime numbers from 1 to 10000 using a method called "The Sieve of Eratosthenes". This method uses an array
* of 1s and 0s which are used to indicate whether a number at the specified index is prime or not. It should be noted that each index (0-9999) is actually
* used to represent a number that is one greater than it (ie: index 0 corresponds to the number 1), thus giving the range 1-10000.
*/

//Include statements
#include <stdio.h>
#include <time.h>

//Main method
void main()
{
    //Used for timing the program
    clock_t start = clock();

    //Initialize the array at size set by the user, here is 10000
    const int ARRSIZE = 10000;
    int array[ARRSIZE];

    //Set each element of the array to 1 to begin
    for(int i = 0; i < ARRSIZE; i++)
    {
        array[i] = 1;
    }

    //Loops through all values in the array
    //When a 1 is found, print the value connected to that index and set the indexes of all multiples of that value to 0 so they are not checked again
    int format = 0;
    for(int i = 1; i < ARRSIZE; i++)
    {
        if(array[i] == 1)
        {
            printf("%d ", i + 1);
            for(int j = i; j < ARRSIZE; j += (i + 1))
            {
                array[j] = 0;
            }
            if(format % 10 == 0)
                printf("\n");
            format++;
        }
    }

    //Methods used for measuring the time the program spends running
    clock_t end = clock();
    double time = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\n\nTime elapsed: %lf", time);
}
```

Testing

Test 1:

```
2
3 5 7 11 13 17 19 23 29 31
37 41 43 47 53 59 61 67 71 73
79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179
181 191 193 197 199 211 223 227 229 233
239 241 251 257 263 269 271 277 281 283
293 307 311 313 317 331 337 347 349 353
359 367 373 379 383 389 397 401 409 419
421 431 433 439 443 449 457 461 463 467
479 487 491 499 503 509 521 523 541 547
557 563 569 571 577 587 593 599 601 607
613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739
743 751 757 761 769 773 787 797 809 811
821 823 827 829 839 853 857 859 863 877
881 883 887 907 911 919 929 937 941 947
953 967 971 977 983 991 997 1009 1013 1019
1021 1031 1033 1039 1049 1051 1061 1063 1069 1087
1091 1093 1097 1103 1109 1117 1123 1129 1151 1153
1163 1171 1181 1187 1193 1201 1213 1217 1223 1229
1231 1237 1249 1259 1277 1279 1283 1289 1291 1297
1301 1303 1307 1319 1321 1327 1361 1367 1373 1381
1399 1409 1423 1427 1429 1433 1439 1447 1451 1453
1459 1471 1481 1483 1487 1489 1493 1499 1511 1523
1531 1543 1549 1553 1559 1567 1571 1579 1583 1597
1601 1607 1609 1613 1619 1621 1627 1637 1657 1663
1667 1669 1693 1697 1699 1709 1721 1723 1733 1741
1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901
1907 1913 1931 1933 1949 1951 1973 1979 1987 1993
1997 1999 2003 2011 2017 2027 2029 2039 2053 2063
2069 2081 2083 2087 2089 2099 2111 2113 2129 2131
2137 2141 2143 2153 2161 2179 2203 2207 2213 2221
2237 2239 2243 2251 2267 2269 2273 2281 2287 2293
2297 2309 2311 2333 2339 2341 2347 2351 2357 2371
2377 2381 2383 2389 2393 2399 2411 2417 2423 2437
2441 2447 2459 2467 2473 2477 2503 2521 2531 2539
2543 2549 2551 2557 2579 2591 2593 2609 2617 2621
2633 2647 2657 2659 2663 2671 2677 2683 2687 2689
2693 2699 2707 2711 2713 2719 2729 2731 2741 2749
2753 2767 2777 2789 2791 2797 2801 2803 2819 2833
2837 2843 2851 2857 2861 2879 2887 2897 2903 2909
2917 2927 2939 2953 2957 2963 2969 2971 2999 3001
3011 3019 3023 3037 3041 3049 3061 3067 3079 3083
3089 3109 3119 3121 3137 3163 3167 3169 3181 3187
```

```
3191 3203 3209 3217 3221 3229 3251 3253 3257 3259
3271 3299 3301 3307 3313 3319 3323 3329 3331 3343
3347 3359 3361 3371 3373 3389 3391 3407 3413 3433
3449 3457 3461 3463 3467 3469 3491 3499 3511 3517
3527 3529 3533 3539 3541 3547 3557 3559 3571 3581
3583 3593 3607 3613 3617 3623 3631 3637 3643 3659
3671 3673 3677 3691 3697 3701 3709 3719 3727 3733
3739 3761 3767 3769 3779 3793 3797 3803 3821 3823
3833 3847 3851 3853 3863 3877 3881 3889 3907 3911
3917 3919 3923 3929 3931 3943 3947 3967 3989 4001
4003 4007 4013 4019 4021 4027 4049 4051 4057 4073
4079 4091 4093 4099 4111 4127 4129 4133 4139 4153
4157 4159 4177 4201 4211 4217 4219 4229 4231 4241
4243 4253 4259 4261 4271 4273 4283 4289 4297 4327
4337 4339 4349 4357 4363 4373 4391 4397 4409 4421
4423 4441 4447 4451 4457 4463 4481 4483 4493 4507
4513 4517 4519 4523 4547 4549 4561 4567 4583 4591
4597 4603 4621 4637 4639 4643 4649 4651 4657 4663
4673 4679 4691 4703 4721 4723 4729 4733 4751 4759
4783 4787 4789 4793 4799 4801 4813 4817 4831 4861
4871 4877 4889 4903 4909 4919 4931 4933 4937 4943
4951 4957 4967 4969 4973 4987 4993 4999 5003 5009
5011 5021 5023 5039 5051 5059 5077 5081 5087 5099
5101 5107 5113 5119 5147 5153 5167 5171 5179 5189
5197 5209 5227 5231 5233 5237 5261 5273 5279 5281
5297 5303 5309 5323 5333 5347 5351 5381 5387 5393
5399 5407 5413 5417 5419 5431 5437 5441 5443 5449
5471 5477 5479 5483 5501 5503 5507 5519 5521 5527
5531 5557 5563 5569 5573 5581 5591 5623 5639 5641
5647 5651 5653 5657 5659 5669 5683 5689 5693 5701
5711 5717 5737 5741 5743 5749 5779 5783 5791 5801
5807 5813 5821 5827 5839 5843 5849 5851 5857 5861
5867 5869 5879 5881 5897 5903 5923 5927 5939 5953
5981 5987 6007 6011 6029 6037 6043 6047 6053 6067
6073 6079 6089 6091 6101 6113 6121 6131 6133 6143
6151 6163 6173 6197 6199 6203 6211 6217 6221 6229
6247 6257 6263 6269 6271 6277 6287 6299 6301 6311
6317 6323 6329 6337 6343 6353 6359 6361 6367 6373
6379 6389 6397 6421 6427 6449 6451 6469 6473 6481
6491 6521 6529 6547 6551 6553 6563 6569 6571 6577
6581 6599 6607 6619 6637 6653 6659 6661 6673 6679
6689 6691 6701 6703 6709 6719 6733 6737 6761 6763
6779 6781 6791 6793 6803 6823 6827 6829 6833 6841
6857 6863 6869 6871 6883 6899 6907 6911 6917 6947
6949 6959 6961 6967 6971 6977 6983 6991 6997 7001
7013 7019 7027 7039 7043 7057 7069 7079 7103 7109
```

```
7121 7127 7129 7151 7159 7177 7187 7193 7207 7211
7213 7219 7229 7237 7243 7247 7253 7283 7297 7307
7309 7321 7331 7333 7349 7351 7369 7393 7411 7417
7433 7451 7457 7459 7477 7481 7487 7489 7499 7507
7517 7523 7529 7537 7541 7547 7549 7559 7561 7573
7577 7583 7589 7591 7603 7607 7621 7639 7643 7649
7669 7673 7681 7687 7691 7699 7703 7717 7723 7727
7741 7753 7757 7759 7789 7793 7817 7823 7829 7841
7853 7867 7873 7877 7879 7883 7901 7907 7919 7927
7933 7937 7949 7951 7963 7993 8009 8011 8017 8039
8053 8059 8069 8081 8087 8089 8093 8101 8111 8117
8123 8147 8161 8167 8171 8179 8191 8209 8219 8221
8231 8233 8237 8243 8263 8269 8273 8287 8291 8293
8297 8311 8317 8329 8353 8363 8369 8377 8387 8389
8419 8423 8429 8431 8443 8447 8461 8467 8501 8513
8521 8527 8537 8539 8543 8563 8573 8581 8597 8599
8609 8623 8627 8629 8641 8647 8663 8669 8677 8681
8689 8693 8699 8707 8713 8719 8731 8737 8741 8747
8753 8761 8779 8783 8803 8807 8819 8821 8831 8837
8839 8849 8861 8863 8867 8887 8893 8923 8929 8933
8941 8951 8963 8969 8971 8999 9001 9007 9011 9013
9029 9041 9043 9049 9059 9067 9091 9103 9109 9127
9133 9137 9151 9157 9161 9173 9181 9187 9199 9203
9209 9221 9227 9239 9241 9257 9277 9281 9283 9293
9311 9319 9323 9337 9341 9343 9349 9371 9377 9391
9397 9403 9413 9419 9421 9431 9433 9437 9439 9461
9463 9467 9473 9479 9491 9497 9511 9521 9533 9539
9547 9551 9587 9601 9613 9619 9623 9629 9631 9643
9649 9661 9677 9679 9689 9697 9719 9721 9733 9739
9743 9749 9767 9769 9781 9787 9791 9803 9811 9817
9829 9833 9839 9851 9857 9859 9871 9883 9887 9901
9907 9923 9929 9931 9941 9949 9967 9973

Time elapsed: 0.167000
```

Test with size 100,000:

```
99559 99563 99571 99577 99581 99607 99611 99623 99643 99661
99667 99679 99689 99707 99709 99713 99719 99721 99733 99761
99767 99787 99793 99809 99817 99823 99829 99833 99839 99859
99871 99877 99881 99901 99907 99923 99929 99961 99971 99989
99991

Time elapsed: 0.698000
```

Results of Testing Programs in Question 2

Algorithm	n	10,000	100,000
n/2		0.178	1.169
sqrt(n)		0.170	0.744
Eratosthenes		0.167	0.698

Note: The numbers above are the times measured in seconds. A second trial of 100,000 was performed because the times for size 10,000 were very quick and it was difficult to differentiate them significantly.

Observations: The slowest of the algorithms was the 1st one, which tested values from 1 to n/2 for each tested in the program. This was expected because the portion of values that were checked from sqrt(n) to n/2 were unnecessary. Because these unnecessary values were not checked in the second program where the loop terminated at sqrt(n), the second program ran faster. The difference in runtime is more pronounced in the 100,000-size test. The Eratosthenes algorithm was consistently faster than the sqrt(n) program, which was expected, but the difference in runtime between these programs was very slight, even in the test size of 100,000. This is likely because the sqrt(n) had certain optimizations implemented into its design, such as only testing odd numbers, and these implementations were not coded into the Eratosthenes design to follow the instructions on the programming assignment. It is likely that the Eratosthenes program would have a more noticeable difference in runtime to the sqrt(n) program if these optimizations were implemented into it.

Question 3

Problem Definition/Specification

A program is needed which will allow the user to control a “turtle” which can draw lines and shapes. The details are outlined in the project assignment page as well as Exercise 6.23.

Problem Analysis

The program is to allow the user to control a “turtle”, which moves through a grid, by inputting numbers to the program which correspond to functions. These functions include the ability to place down a pen, lift the pen up, turn left, turn right, move forward an indicated number of spaces, print out the full grid, and exit the program. The numbers corresponding to each function are shown in the programming assignment handout and are detailed in the Relations section below. Additionally, some additional functions may be implemented to allow for a greater ease of use for the programmer. Two additional functions that will be implemented include a help menu to help keep track of the different possible commands, and the ability to see the current position and direction of the cursor. With all the commands correctly implemented, the user of the program should be able to control the “turtle” on the screen and draw different shapes depending on whether the pen is currently down.

Program Inputs

- An integer command which tells the program what function to perform
- A number of spaces to move forward (only necessary if a move forward command is input)

Program Outputs

- A 20 x 20 grid with portions of the grid that have been “drawn” on labeled with an asterisk (*)
- Statements to indicate whether commands were successfully read and performed

Formula or Relations

- Command 1: Lift pen up
- Command 2: Place pen down
- Command 3: Turn to the right
- Command 4: Turn to the left
- Command 5, #: Move forward a number (#) of spaces
- Command 6: Print the grid
- Command 7: (Added by programmer) Print out a help menu
- Command 8: (Added by programmer) Print the current position and direction
- Command 9: Exit the program

Algorithm Design

Initial Algorithm

1. Initialize a 20x20 array of integers and set each position to 0
2. Read in a command from the user
3. Determine which function to be completed based on the input value
4. Perform the function based on the determination made
5. Allow the user to input a new command and repeat until the exit command is entered

Refinement

1. Create a 20x20 integer array and loop through its rows and column with a for loop to set each position to 0. Additionally, initialize the position to be (0,0) and set the initial direction to east.
2. Use a scanf() line to read in up to 2 integer inputs from the user (the command and the number of spaces to move if the command to move forward is given)
3. Use a series of if-else statements to determine what function should be performed based on the command input. See the previous Relations section to see how the input integers will map to each command
4. Within the if statement, perform the appropriate function, using methods if necessary
   a. If the pen up command is given, set an integer value penIsDown to 0 (false) and print out a success message
   b. If the pen down command is given, set an integer value penIsDown to 1 (true) and print out a success message
   c. If turn right command is given, call the void method rightTurn(), which will update the direction the cursor is facing based on the current direction using a series of if-else statements. Directions will be referred to as north, south, east, and west
   d. If turn left command is given, call the void method leftTurn(), which will update the direction the cursor is facing based on the current direction using a series of if-else statements. Directions will be referred to as north, south, east, and west
   e. If the move forward command is given, first use a series of if-else statements to determine the current direction of the cursor. Then, based on this direction, increment the x or y value in the appropriate direction in a loop that last a number of times taken from the user's input. If during this loop a bound of the 20x20 array is reached, immediately exit the function. Regardless of how many spaces were input, output how many spaces were actually moved. All of this takes place in the integer method moveForward(int, int[][COL_SIZE]), which returns the number of spaces successfully moved
   f. If the print grid command is given, loop through the entire 20x20 array and print an asterisk (*) wherever a 1 is found. Additionally, for ease of use, print out a plus sign (+) wherever the cursor is currently located by using an if statement that checks if the position in the loop is the current position of the cursor
   g. If the help command is input, output a list of all the commands and their corresponding numbers

    h. If the print position command is given, print the current position and direction of the cursor
    i. If the exit command is given, print out an end message and break out of the loop
    j. If none of the above commands are given, display an error message and prompt the user to input a valid command
5. The program will continue to allow the user to input commands in a loop until the exit command is entered

Desk Checking/Test Plan

Ensuring this program works correctly requires more comprehensive testing than the previous programs in this assignment. First, the program will be used to create a shape in the console, including both right and left turns and alternations of the pen being down and up. In order to ensure the bounds have been correctly input, some values that would normally allow the cursor to go beyond the bounds of the grid will be tested. Each of the 9 commands will be tested at least once, with the more complex ones such as the move forward command being tested more often. Additionally, an invalid command will be tested in the program to ensure that it does not cause any unwanted side effects. If the program crashes or fails to print out the expected output, it can be determined that the program was not created correctly.

Implementation

```
/*
* Programmer: Kollin Labowski
* Date Completed: October 1, 2020
* Course: CS 350
* Description: This program allows the user to control a "turtle" by inputting a variety of commands that allow the
* turtle to draw shapes and lines when indicated by the user. Commands include the ability to raise and lower a pen
* that is used to draw the lines and shapes, the ability to turn either right or left, the ability to move forward
* a number of spaces, the ability to print out the grid, and the ability to exit the program. Additionally, 2 extra
* functions were added that were not required in the assignment document, these functions being the ability to display
* a help menu and the ability to display the current position and direction of the "turtle".
*/

//Include Statements
#include <stdio.h>

//Constants used to indicate direction
const int EAST = 0;
const int SOUTH = 1;
const int WEST = 2;
const int NORTH = 3;

//Constants used to indicate size of grid
const int ROW_SIZE = 20;
const int COL_SIZE = 20;

//Function prototypes
void rightTurn();
void leftTurn();
void draw(int[][COL_SIZE]);
int moveForward(int numSpaces, int[][COL_SIZE]);
void drawGrid(int[][COL_SIZE]);

//Variables that are modified
int penIsDown = 0;
int xPos = 0;
int yPos = 0;
int direction = EAST;

//Main method
void main()
{
    //The grid to be modified throughout the program
    int grid[ROW_SIZE][COL_SIZE];

    //Initialize the array to have all zeroes
    for(int i = 0; i < ROW_SIZE; i++)
        for(int j = 0; j < COL_SIZE; j++)
            grid[i][j] = 0;

```

```
50      //Loops until a sentinel values is reached, in which case the loop is broken out of
51      while(1)
52      {
53          //Variables to track the input command and number of spaces if command 5 is input
54          int command;
55          int spaces;
56          printf("Please input a command: ");
57          scanf("%d, %d", &command, &spaces);
58
59          //If statements used to ensure the correct command is performed based on the input
60          if(command == 1)
61          {
62              //Brings pen up
63              penIsDown = 0;
64              printf("\nThe pen is now up...\n\n");
65          }
66          else if(command == 2)
67          {
68              //Puts pen down
69              penIsDown = 1;
70              draw(grid);
71              printf("\nThe pen is now down...\n\n");
72          }
73          else if(command == 3)
74          {
75              //Performs a right turn
76              rightTurn();
77              printf("\nTurned to the right...\n\n");
78          }
79          else if(command == 4)
80          {
81              //Performs a left turn
82              leftTurn();
83              printf("\nTurned to the left...\n\n");
84          }
85          else if(command == 5)
86          {
87              //Moves forward a number of spaces if possible
88              if((spaces >= ROW_SIZE && spaces >= COL_SIZE) || spaces < 0)
89                  printf("\nInvalid input for number of spaces...\n\n");
90              else
91              {
92                  int actualMovement = moveForward(spaces, grid);
93                  printf("\nMoved forward %d spaces...\n\n", actualMovement);
94              }
95          }
```

```
96          else if(command == 6)
97          {
98              //Prints out the full grid
99              printf("\nPrinting the grid below...\n\n");
100             drawGrid(grid);
101         }
102         else if(command == 7)
103         {
104             //Displays a help menu
105             printf("\n - Input 1 to move pen up");
106             printf("\n - Input 2 to move pen down");
107             printf("\n - Input 3 to turn right");
108             printf("\n - Input 4 to turn left");
109             printf("\n - Input 5 followed be a comma and a # to move a # of spaces forward");
110             printf("\n - Input 6 to print out the grid");
111             printf("\n - Input 7 to print this help menu");
112             printf("\n - Input 8 to print the current location and direction of \"turtle\"");
113             printf("\n - Input 9 to exit the program\n\n");
114         }
115         else if(command == 8)
116         {
117             //Displays the position and direction of the "turtle"
118             printf("\nCurrent position is (%d,%d)", xPos, yPos);
119             if(direction == EAST)
120                 printf("\nCurrent direction is EAST\n\n");
121             else if(direction == SOUTH)
122                 printf("\nCurrent direction is SOUTH\n\n");
123             else if(direction == WEST)
124                 printf("\nCurrent direction is WEST\n\n");
125             else if(direction == NORTH)
126                 printf("\nCurrent direction is NORTH\n\n");
127             else
128                 printf("\nCurrent direction is UNKNOWN\n\n");
129         }
130         else if(command == 9)
131         {
132             //Exits the program
133             printf("\nExiting the program...\n");
134             break;
135         }
136         else
137         {
138             //Invalid input was received
139             printf("\nPlease enter a valid input (input 7 for help)...\n\n");
140         }
141     }
142 }
143
```

```
144  /*
145  * This method allows the "turtle" to turn right by resetting the direction based on the current direction
146  */
147  void rightTurn()
148  {
149      if(direction == EAST)
150      {
151          direction = SOUTH;
152      }
153      else if(direction == SOUTH)
154      {
155          direction = WEST;
156      }
157      else if(direction == WEST)
158      {
159          direction = NORTH;
160      }
161      else if(direction == NORTH)
162      {
163          direction = EAST;
164      }
165  }
166
167  /*
168  * This method allows the "turtle" to turn left by resetting the direction based on the current direction
169  */
170  void leftTurn()
171  {
172      if(direction == EAST)
173      {
174          direction = NORTH;
175      }
176      else if(direction == SOUTH)
177      {
178          direction = EAST;
179      }
180      else if(direction == WEST)
181      {
182          direction = SOUTH;
183      }
184      else if(direction == NORTH)
185      {
186          direction = WEST;
187      }
188  }
189
```

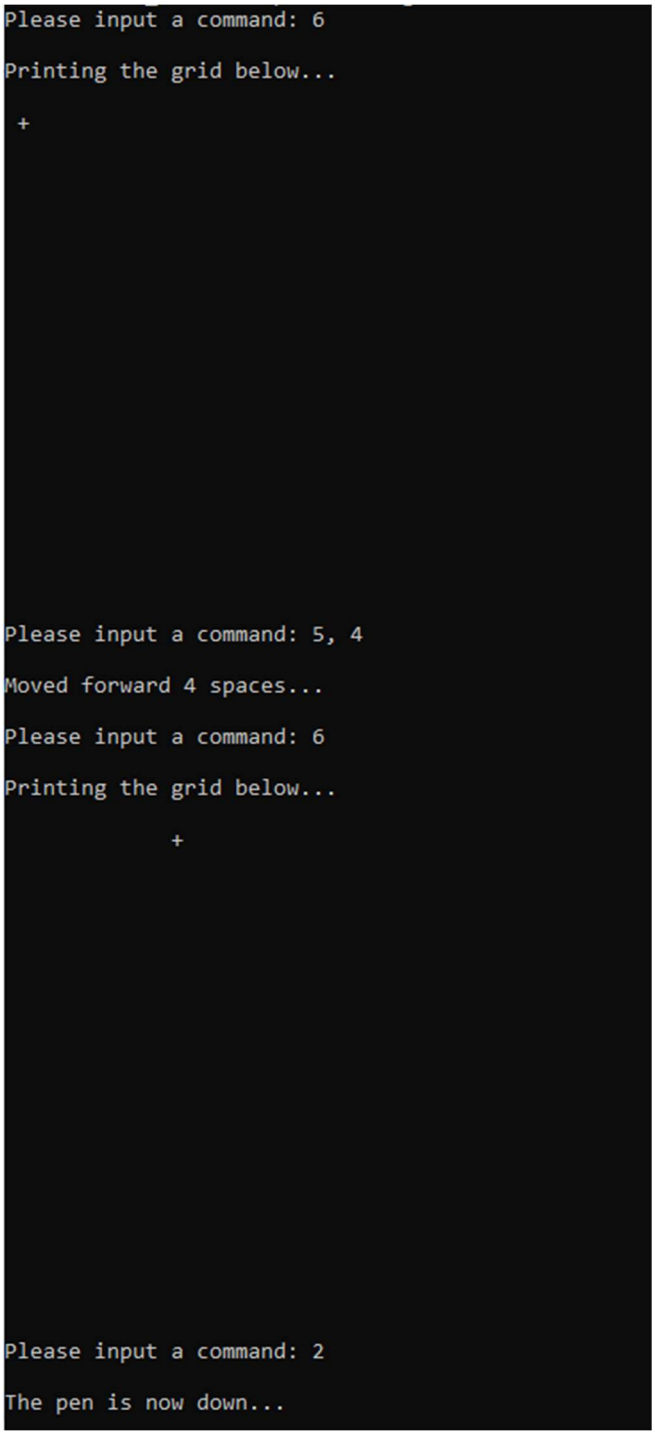
```
/*
* This is a simple method which sets a position to 1 based on the current x and y position of the "turtle"
*/
void draw(int grid[ROW_SIZE][COL_SIZE])
{
    grid[yPos][xPos] = 1;
}

/*
* Allows the "turtle" to move forward a number of spaces as input by the user. If the pen is down, also draw along this line
*/
int moveForward(int numSpaces, int grid[ROW_SIZE][COL_SIZE])
{
    //Used to keep track of the amount of spaces ACTUALLY moved, which may differ from numSpaces if a bound is reached
    int movementCount = 0;

    //If statements used to output based on the current direction
    if(direction == EAST)
    {
        for(int i = 0; i < numSpaces; i++)
        {
            if(xPos + 1 >= ROW_SIZE)
                return movementCount;

            xPos += 1;
            if(penIsDown)
                draw(grid);
            movementCount++;
        }
    }
    else if(direction == SOUTH)
    {
        for(int i = 0; i < numSpaces; i++)
        {
            if(yPos + 1 >= COL_SIZE)
                return movementCount;

            yPos += 1;
            if(penIsDown)
                draw(grid);
            movementCount++;
        }
    }
```

```
233         else if(direction == WEST)
234         {
235             for(int i = 0; i < numSpaces; i++)
236             {
237                 if(xPos - 1 < 0)
238                     return movementCount;
239 
240                 xPos -= 1;
241                 if(penIsDown)
242                     draw(grid);
243                 movementCount++;
244             }
245         }
246         else if(direction == NORTH)
247         {
248             for(int i = 0; i < numSpaces; i++)
249             {
250                 if(yPos - 1 < 0)
251                     return movementCount;
252 
253                 yPos -= 1;
254                 if(penIsDown)
255                     draw(grid);
256                 movementCount++;
257             }
258         }
259         return movementCount;
260     }
261 
```

```
262 /*
263 * Prints out the entire grid, using * to indicate positions of 1 and + to indicate the current location of the "turtle"
264 */
265 void drawGrid(int grid[ROW_SIZE][COL_SIZE])
266 {
267     for(int i = 0; i < ROW_SIZE; i++)
268     {
269         for(int j = 0; j < COL_SIZE; j++)
270         {
271             if(i == yPos && j == xPos)
272                 printf(" + ");
273             else
274             {
275                 if(grid[i][j] == 1)
276                     printf(" * ");
277                 else
278                     printf("   ");
279             }
280         }
281         printf("\n");
282     }
283 }
284 //End of program
```

Testing

Test 1:

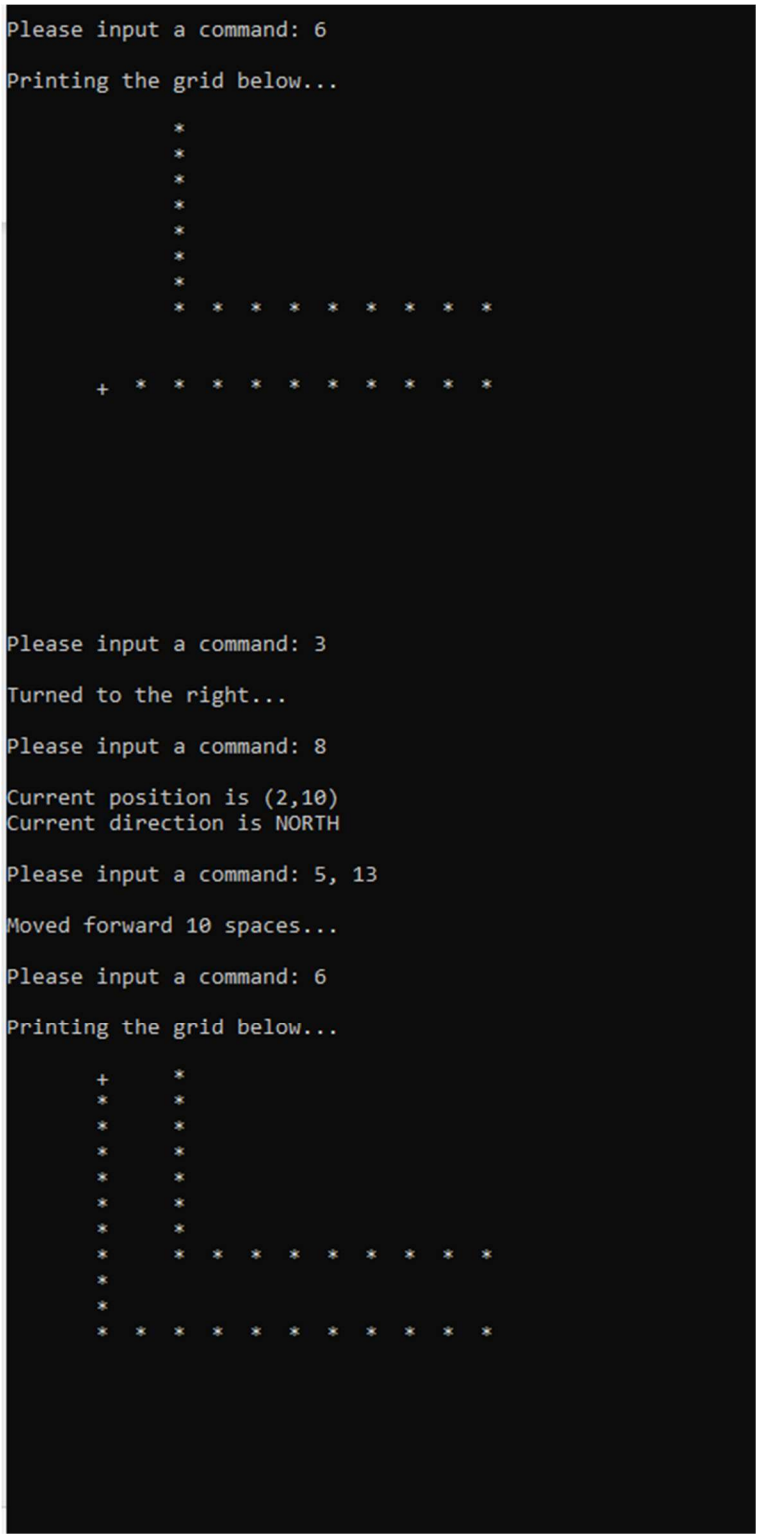


Note: The + sign is used to show the position of the cursor, and is not actually part of the array

```
Please input a command: 3

Turned to the right...

Please input a command: 5, 7

Moved forward 7 spaces...

Please input a command: 6

Printing the grid below...

             *
             *
             *
             *
             *
             *
             *
             *
             +









Please input a command: 4

Turned to the left...

Please input a command: 5, 8

Moved forward 8 spaces...
```

```
Please input a command: 6

Printing the grid below...

            *
            *
            *
            *
            *
            *
            *
            *  *  *  *  *  *  *  *  +









Please input a command: 1

The pen is now up...

Please input a command: 3

Turned to the right...

Please input a command: 5,6

Moved forward 6 spaces...

Please input a command: 6

Printing the grid below...

            *
            *
            *
            *
            *
            *
            *
            *  *  *  *  *  *  *  *  *




                                  +
```

```
Please input a command: 4

Turned to the left...

Please input a command: 4

Turned to the left...

Please input a command: 5, 3

Moved forward 3 spaces...

Please input a command: 6

Printing the grid below...

            *
            *
            *
            *
            *
            *
            *
            *  *  *  *  *  *  *  *  *


                                  +









Please input a command: 2

The pen is now down...

Please input a command: 8

Current position is (12,10)
Current direction is NORTH

Please input a command: 4

Turned to the left...

Please input a command: 5, 10

Moved forward 10 spaces...
```

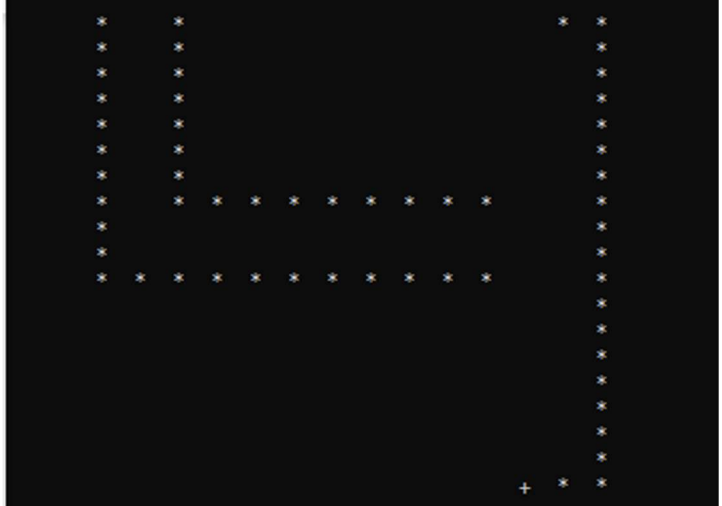


Note: A distance of 13 was entered, but it only moved a distance of 10. This is expected behavior when the cursor approaches the bounds of the array

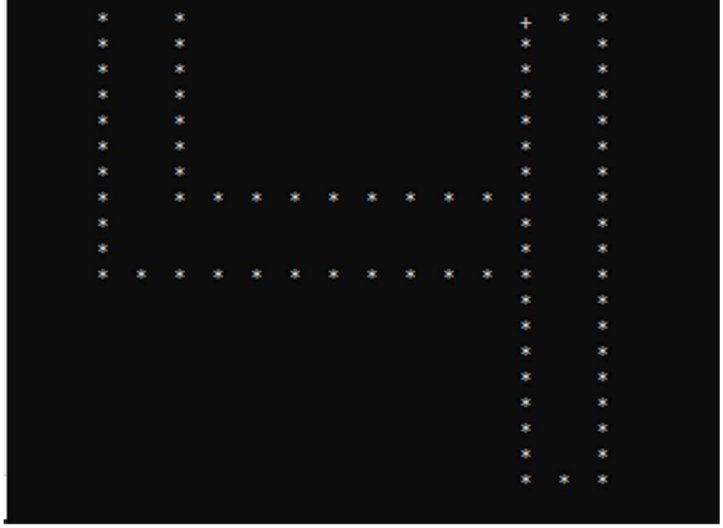
```
Please input a command: 1

The pen is now up...

Please input a command: 3

Turned to the right...

Please input a command: 5, 12

Moved forward 12 spaces...

Please input a command: 6

Printing the grid below...

       *     *                           +
       *     *
       *     *
       *     *
       *     *
       *     *
       *     *
       *     *  *  *  *  *  *  *  *  *
       *
       *
       *  *  *  *  *  *  *  *  *  *  *









Please input a command: 2

The pen is now down...

Please input a command: 5, 1

Moved forward 1 spaces...
```


Please input a command: 6
Printing the grid below...
Please input a command: 3
Turned to the right...
Please input a command: 5, 18
Moved forward 18 spaces...
Please input a command: 6
Printing the grid below...
Please input a command: 3
Turned to the right...

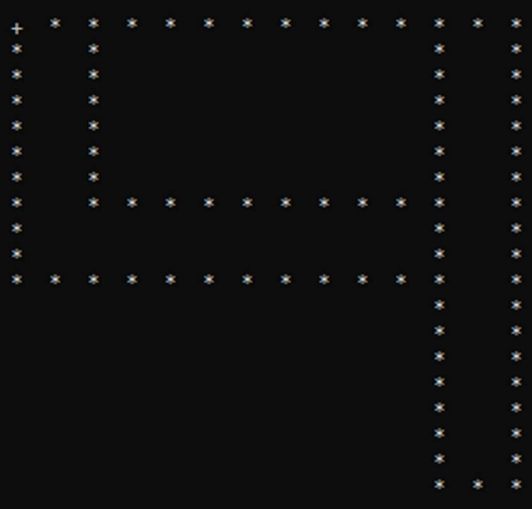
```
Please input a command: 5, 2

Moved forward 2 spaces...

Please input a command: 6

Printing the grid below...
```



```
Please input a command: 3

Turned to the right...

Please input a command: 5, 18

Moved forward 18 spaces...

Please input a command: 6

Printing the grid below...
```



```
Please input a command: 4

Turned to the left...

Please input a command: 5, 11

Moved forward 11 spaces...

Please input a command: 6

Printing the grid below...
```



```
Please input a command: 8

Current position is (2,0)
Current direction is WEST

Please input a command: 7

 - Input 1 to move pen up
 - Input 2 to move pen down
 - Input 3 to turn right
 - Input 4 to turn left
 - Input 5 followed be a comma and a # to move a # of spaces forward
 - Input 6 to print out the grid
 - Input 7 to print this help menu
 - Input 8 to print the current location and direction of "turtle"
 - Input 9 to exit the program

Please input a command: 9

Exiting the program...
```