C Program for Prime Numbers

Prime numbers are an interesting and fundamental concept in number theory. They're the building blocks of natural numbers, having a significant role in various fields like cryptography and internet security. In this article, we'll explore different ways to write a prime number program in C, providing detailed explanations and examples for each method.

What is a Prime Number?

The prime number refers to any natural number that is greater than 1, with no positive divisors besides 1 or itself. For example, the first six prime numbers are 2, 3, 5, 7, 11, and 13.

Here are some of the more interesting facts about prime numbers to understand them better!

- Prime numbers are infinite. There's no limit to how many prime numbers exist.
- Every natural number greater than 1 is either a prime number or can be factored into prime numbers.
- '2' is the smallest and only even prime number.
- All prime numbers larger than 3 can be written as 6k+1 or 6k−1 for some natural number k.

C Program for Prime Numbers Using For Loop

A basic way to find if a number is prime is to loop from 2 to n-1 and check if the number is divisible by any number in this range. If it is, then it's not a prime number; otherwise, it is.

Here's a simple prime number program in C using for loop:

```
#include <stdio.h>

int main()
  int                0
  printf "Enter a positive integer: "
  scanf "%d"

  for     2          2
    if           0
                 1
       break
```

```
  if          1
    printf "1 is neither prime nor composite."

  else
    if              0
      printf "%d is a prime number."
    else
      printf "%d is not a prime number."

  return 0
```

Time Complexity: O(n)

The time complexity for this program is O(n) because, in the worst-case scenario, we're iterating from 2 to n/2.

Space Complexity: O(1)

The space complexity for this program is O(1), as we're not using any additional space that scales with input size.

Programs to Check for Prime Numbers in C

1. Naive Approach to check Prime number in C

The naive approach to check for a prime number is by checking divisibility from 2 to n-1. If n is divisible by any number in this range, it's not prime.

Example:

```
#include<stdio.h>
int main()
  int                0
  printf "Enter a positive integer: "
```

```
  scanf "%d"
  for     2
    if              0
                  1
        break
```

```
  if        1
              1

  if              0
     printf "%d is a prime number.\n"
  else
     printf "%d is not a prime number.\n"

  return 0
```

2. Program to Check Prime Number using sqrt(N)

A more optimised approach is to check divisibility up to the square root of n. This is because a larger factor of the number must be multiple of a smaller factor that has already been checked.

```
#include<stdio.h>
#include<math.h>

int main()
  int                 0
  printf "Enter a positive integer: "
  scanf "%d"

  for     2        sqrt
    if              0
                  1
        break
```

```
  if        1
              1
```

```
  if              0
     printf "%d is a prime number.\n"
  else
     printf "%d is not a prime number.\n"

  return 0
```

3. Program to Check Prime Numbers using Wilson’s Theorem

Wilson's theorem states that a number n is a prime number program in C only if (n - 1)! + 1 is a multiple of n.

Note: this method may not be efficient for large numbers as factorial values grow rapidly.

```
#include<stdio.h>

long long factorial(int n)
  long long          1
  for int      2
  return
```

```
int main()
  int
  printf "Enter a positive integer: "
  scanf "%d"

  if         1
     printf "%d is not a prime number.\n"
     return 0
  if                    1                   1
     printf "%d is a prime number.\n"
  else
     printf "%d is not a prime number.\n"

  return 0
```

Reason for Iterating the Loop Till n/2

The reason we only need to iterate the loop till n/2 in a prime checking program is because of the nature of factors. Any n factor greater than n/2 would need to be multiplied by a number less than 2 to produce n, and since all factors must be integers, the only possible factor in this scenario is n itself.

C Program for Prime Numbers Using While Loop

In C programming, the while loop is used to repeatedly execute a block of code as long as a given condition is true. We can use a while loop to check whether a number is prime.

Example:

```
#include<stdio.h>

int main()
  int          2              0
  printf "Enter a positive integer: "
  scanf "%d"

  while            2
    if             0
                 1
       break
  if        1
              1

  if             0
    printf "%d is a prime number.\n"
  else
    printf "%d is not a prime number.\n"

  return 0
```

The check_prime function first handles the special cases of n being less than or equal to 1, equal to 2, or an even number (which can be determined without looping).

The while loop then starts with i equal to 3, and continues as long as i squared is less than or equal to n (since we only need to check divisors up to the square root of n). For each i, the function checks if n is divisible by i, and if so, it immediately returns 0 (indicating that n is not prime). After each iteration, i is incremented by 2 (since even divisors other than 2 can be ignored). If the while loop completes without finding any divisors, n is prime, so the function returns 1.

In the main function, the program asks the user to input a number n, then calls check_prime(n). If check_prime(n) returns 1 (indicating that n is prime), the program prints that n is a prime number. Otherwise, it prints that n is not a prime number.

C Program for Prime Numbers Using Functions

We can also create a function to check for prime number program in C and then call that function whenever we need it:

```
#include<stdio.h>

int check_prime(int n)
  int
  for     2          2
    if             0
       return 0

  return 1


int main()
  int
  printf "Enter a positive integer: "
  scanf "%d"

  if       1
    printf "%d is not a prime number.\n"
  else if                          1
    printf "%d is a prime number.\n"
  else
    printf "%d is not a prime number.\n"
```

```
  return 0
}
```

C Program for Prime Numbers Within a Range

The C program to print prime numbers from 1 to n, i.e., within a range, follows a simple procedure: it loops through each number within the given range, checks if the number is prime and if the number is prime, it prints the number.

Example:

```
#include<stdio.h>

int check_prime(int n)
{
  int
  for     2          2
    if              0
      return 0
  }
  return 1
}

int main()
{
  int
  printf "Enter two numbers (intervals): "
  scanf "%d %d"

  printf "Prime numbers between %d and %d are: "

  for int
    if     1
      if                    1
        printf "%d "
    }
  }

  return 0
}
```

Here's what the program does:

- It starts by defining a function check_prime(int n). This function checks whether a given number n is prime by looping from 2 to n/2 and checking if n is divisible by any of these numbers. If n is divisible by any number in this range, the function returns 0 (indicating that n is not prime). If n is not divisible by any number in this range, the function returns 1 (indicating that n is prime).
- In the main function, the program first asks the user to input two numbers, low and high, representing the range within which the program will find prime numbers.
- The program then enters a loop that runs from low to high. For each number i in this range, the program checks if i is greater than 1 (since 1 is not a prime number) and if check_prime(i) returns 1 (indicating that i is prime). If both conditions are met, the program prints i.

So, in this way, the program prints all prime numbers within the range [low, high].

Please note that the prime-checking algorithm used in this program has a time complexity of O(n), so it may run slowly for large input ranges. You can optimise it by checking divisibility up to the square root of n or using more efficient algorithms like the Sieve of Eratosthenes.

C Program for Prime Numbers Using Sieve of Eratosthenes

The Sieve of Eratosthenes algorithm efficiently finds all prime numbers up till a given limit. This algorithm is named after the ancient Greek mathematician Eratosthenes, who first described it.

What does "sieve" mean?

The word "sieve" refers to a tool used to separate fine particles from coarse ones. In the context of this algorithm, it's a metaphorical sieve used to filter out composite numbers (non-primes), leaving behind only prime numbers.

The algorithm works by iteratively marking the multiples of each number starting from 2. The steps are as follows:

1. For a list of successive integers from 2 across n.
2. Let the smallest prime, p, equal 2,
3. Enumerate multiples of p by counting in increments of p from 2p to n, while marking them in the list (these are 2p, 3p, 4p, etc.; the p itself must be unmarked).

4. Look for the smallest number listed, which is greater than p and unmarked. If no number as such is found, then stop. Otherwise, enable p to equal this new number (the next prime) and repeat the process from step 3.

Here's the C program for it:

```
#include<stdio.h>
#define MAX_SIZE 100000

void sieve(int n, int primes[])
  int            0
  int

  for     2
                1
    for int    2
      if          0
                    0
          break


    if






int main()
  int
  printf "Enter the number till which prime numbers are to be found: "
  scanf "%d"

  int


  for int    0                                  0
    printf "%d "

  return 0

```

This program starts by taking an integer n as input from the user. Then it calls the sieve function, which populates an array of primes with all prime numbers up to n. In the end, it prints the prime numbers.

Conclusion

We delved into an array of methods to identify and program for prime number in C in this comprehensive tutorial. We examined diverse techniques, each with their respective time complexities and code implementations. While each method has its use cases, it's critical to choose the most efficient approach that meets your specific requirements.

Programming is a domain where there is always more to learn and master. As we've seen in this tutorial, even a seemingly simple problem like prime number identification can have multiple solutions, each with its own trade-offs.

If you are excited about mastering the art of programming and want to dig deeper into languages like C, Python, Java, and many more, you might want to consider upGrad's Full Stack Software Development Bootcamp, designed to equip you with the skills needed to excel in the technology industry.

Remember, the journey of mastering programming is a marathon, not a sprint, and every line of code you write gets you one step closer to your goal. Happy Coding!

FAQs

1. What is the time complexity of the Sieve of Eratosthenes?
The time complexity of the Sieve of Eratosthenes is O(n log log n), making it an efficient method to find all primes inferior to n when n is smaller than 10 million or so.

2. Why do we check up to the square root of a number to determine if it's prime?
We check up to the square root of a number because a larger factor of the number would be multiple of a smaller factor that has already been checked.

3. Can the number 1 be considered a prime number?
No, 1 is not considered a prime number. By definition, a prime number has exactly two distinct positive divisors: 1 and itself. Since 1 only has one divisor (1), it does not meet the criteria.
