Assignment 1 Report: sqrtUser()

MECHTRON 2MP3: Programming for Mechatronics

Furqaan Khurram Qamar
400514719

Method Used:

The method used to find the square root of a specified number was the binary search method.

The function calculatePrecision() is used to calculate the precision needed to the decimal place we need. For example if I need ***n*** values of accuracy, entering int ***n*** into the function calculatePrecision() will return a float value called precision which is equal to $1 \cdot 10^{-n}$.

In the sqrtUser() function itself:

- An initial float ***'number'*** is the value we want to find the square root to
- A float ***'precision'*** is set by using the calculatePrecision() function.
- Floats ***'left'*** and ***'right'*** are set as 0 and ***number*** respectively.
- A float **'middle'** is set halfway between ***left*** and ***right.***

While the difference between ***right*** and ***left*** is more than the ***precision*** value, the function checks to see if the ***middle*** value is the square root of ***number.*** If $middle^2$ is greater than ***number*** then ***right*** is set to ***middle,*** otherwise ***left*** is equal to ***middle.*** This sets a new ***middle*** and either a new ***left*** or ***right.*** This loop continues until the difference between ***left*** and ***right*** is less than the precision value. When this loop ends, a new variable ***answer*** is equal to ***right*** since ***right*** is the value we need rather than ***middle*** as found through debugging.

Exceptions:

- If ***number*** is negative, there is an error and the function returns an error message that says we cannot calculate the square root of a negative number.

- If ***number*** is “0” or “1”, the answer is equal to ***number*** itself.
- If ***number*** is less than 1, we have to set the **right** value to 1 since the square root at that point is greater than ***number*** itself and it will be between 0 and 1. The same logic as 1 < ***number*** < infinity applies after.

Compiling and Running:

The C code provided below should compile and run on any operating system that has a C compiler, such as GCC or Clang. Firstly on an IDE or text editor, replace ***number*** and ***n*** in main with the values needed and save the file. To compile and run the code in the terminal using GCC we can use:

```
gcc -o sqrtUser sqrtUser.c
./sqrtUser
```

This will then successfully run the code.

Time Complexity and Comparison:

According to ChatGPT, the time complexity of the sqrt() function in the <math.h> library is considered to be O(1), which means it takes a constant time to run the sqrt() function. Algorithmic complexity and hardware optimization causes the function to work as a direct computation.

Also according to ChatGPT, the time complexity of the sqrtUser() function implemented by me should be O(n+log(m)).

This is because the time complexity of the precision calculation function is O(n), as the precision calculation runs as many times as the number of decimal places, ***n***, required for precision. As for the binary search algorithm, the time complexity is O(log(m)) where m is the variable ***number***. Since these two time complexities contribute together to the total performance, the time complexity of sqrtUser() is O(n+log(m)).

Since it is highly optimized, avoids the extra calculations needed and the time complexity is much faster, the sqrt() function is much more likely to be faster compared to my sqrtUser() function.

Appendix:

References:

OpenAI. (2024). *ChatGPT* (Mar 14 version) [Large language model]. https://chat.openai.com/chat

Source Code:

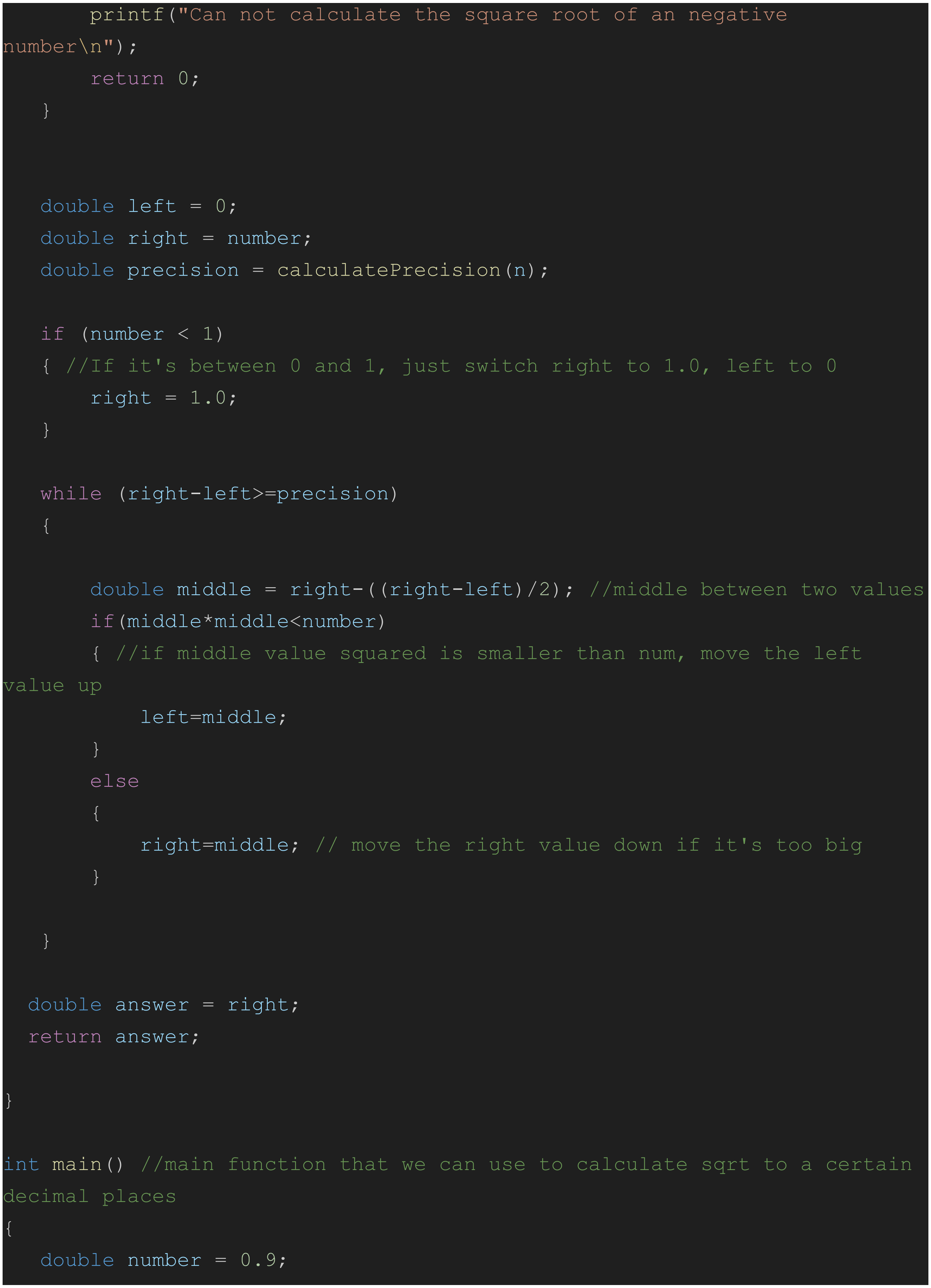
```c
#include <stdio.h>

double calculatePrecision(int n)
{ // This function uses n to calculate the precision to the decimal place
    double precision = 1.0;
    for (int i = 0; i < (n+1); i++)
    {
        precision /= 10.0; // Divide by 10 for each decimal place plus an extra decimal place
    }
    return precision;
}



double sqrtUser (double number , int n )
{
    if (number==1 || number == 0)
    { //sqrt of these is itself
        printf("The square root of this number is %f\n", number);
        return 0;
    }

    if (number<0)
    { //can't calculate imaginary number sqrt
```



```
       printf("Can not calculate the square root of an negative
number\n");
       return 0;
   }


   double left = 0;
   double right = number;
   double precision = calculatePrecision(n);

   if (number < 1)
   { //If it's between 0 and 1, just switch right to 1.0, left to 0
       right = 1.0;
   }

   while (right-left>=precision)
   {

       double middle = right-((right-left)/2); //middle between two values
       if(middle*middle<number)
       { //if middle value squared is smaller than num, move the left
value up
           left=middle;
       }
       else
       {
           right=middle; // move the right value down if it's too big
       }

   }

  double answer = right;
  return answer;

}

int main() //main function that we can use to calculate sqrt to a certain
decimal places
{
   double number = 0.9;
```

```
    int n = 10;
    double result = sqrtUser(number,n);
    if (result!=0){
        printf("The square root of %f, accurate to %d decimal places is %.*lf\n",number,n,n,result);
    }
}
```