

**Lab Assignment 6**

Fall 2024

**Course Title: Structured Programming Lab**

**Course Code: CSE 1202 (Fall 2024)**

**Submitted by:** **Student Name and ID**

Md. Tazminur Rahman Tanim (242014124)

**Department of CSE**

# University of Liberal Arts Bangladesh (ULAB)

1. Write a function in c program to find the perfect numbers within a given number of ranges.

Pass the range as a parameter.

Input: Lower Range: 1 Upper Range: 50

Output: 6 28

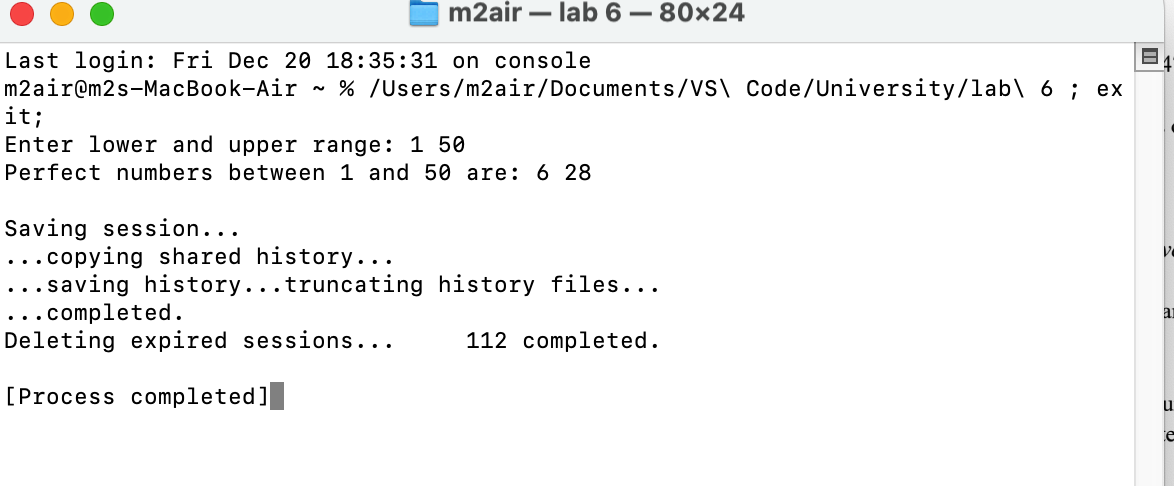
Answer :

**Algorithm:**

1. Input lower and upper range.
2. For each number in range, sum its divisors.
3. If the sum equals the number, it’s perfect.
4. Print perfect numbers.

|  |
| --- |
| Code |
| #include <stdio.h>  int is\_perfect(int num) {  int sum = 0;  for (int i = 1; i < num; i++) {  if (num % i == 0) sum += i;  }  return sum == num;  }  void find\_perfect\_numbers(int lower, int upper) {  printf("Perfect numbers between %d and %d are: ", lower, upper);  for (int i = lower; i <= upper; i++) {  if (is\_perfect(i)) printf("%d ", i);  }  printf("\n");  }  int main() {  int lower, upper;  printf("Enter lower and upper range: ");  scanf("%d %d", &lower, &upper);  find\_perfect\_numbers(lower, upper);  return 0;  } |

Output Result :



2. Write a function in C to find the prime numbers within a range of numbers.

Pass the range as a parameter.

Input: Lower Range: 1 Upper Range: 50

Output : 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

Answer :

**Algorithm:**

1. Input lower and upper range.

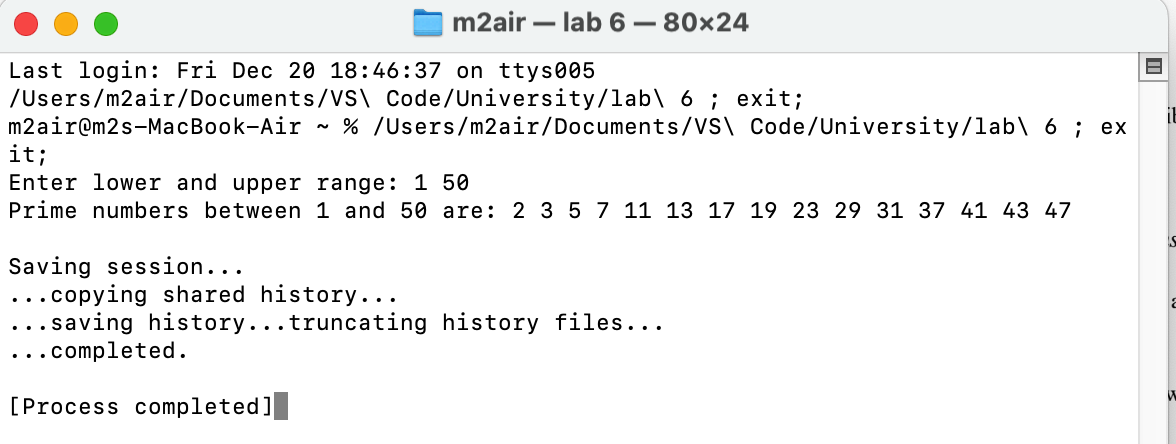
2. For each number, check divisibility from 2 to √number.

3. If not divisible, it’s prime.

4. Print prime numbers.

|  |
| --- |
| Code |
| #include <stdio.h>  int is\_prime(int num) {  if (num < 2) return 0;  for (int i = 2; i \* i <= num; i++) {  if (num % i == 0) return 0;  }  return 1;  }  void find\_primes(int lower, int upper) {  printf("Prime numbers between %d and %d are: ", lower, upper);  for (int i = lower; i <= upper; i++) {  if (is\_prime(i)) printf("%d ", i);  }  printf("\n");  }  int main() {  int lower, upper;  printf("Enter lower and upper range: ");  scanf("%d %d", &lower, &upper);  find\_primes(lower, upper);  return 0;  } |

Output Result :



3. Write a function to obtain the first 10 numbers of a Fibonacci sequence.

Pass a value as a parameter.

Input: 10

Output: 0 1 1 2 3 5 8 13 21 34

Answer :

**Algorithm:**

1 Input number of terms (n).

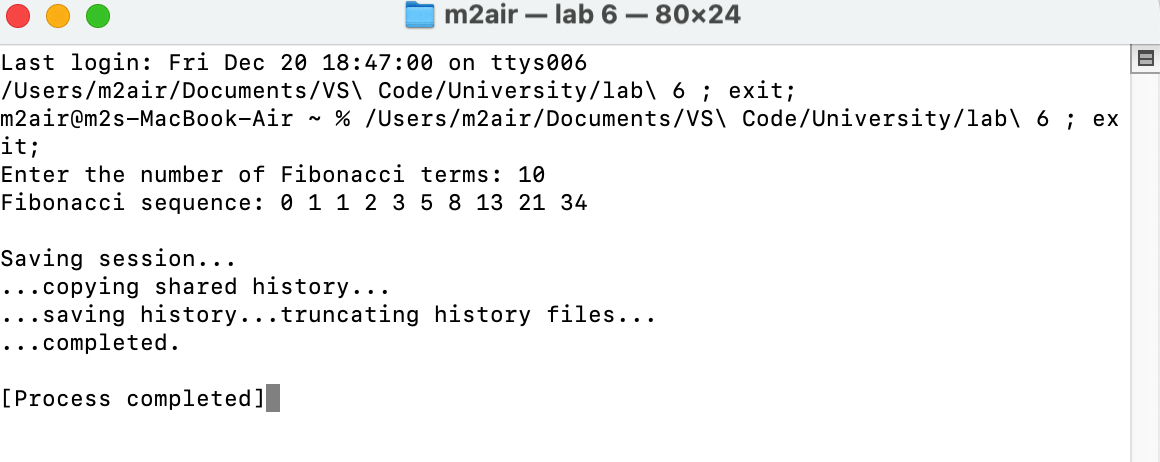
2 Start with 0 and 1.

3 Add the last two terms to get the next term.

4 Repeat for n terms and print.

|  |
| --- |
| Code |
| #include <stdio.h>  void fibonacci(int n) {  int a = 0, b = 1, next;  printf("Fibonacci sequence: %d %d ", a, b);  for (int i = 3; i <= n; i++) {  next = a + b;  printf("%d ", next);  a = b;  b = next;  }  printf("\n");  }  int main() {  int n;  printf("Enter the number of Fibonacci terms: ");  scanf("%d", &n);  fibonacci(n);  return 0;  } |

Output Result :



4. Write 4 different functions for calculating the area of a triangle, rectangle,

square and circle.

Answer :

**Algorithm:**

a) Input dimensions ( base, height, radius).

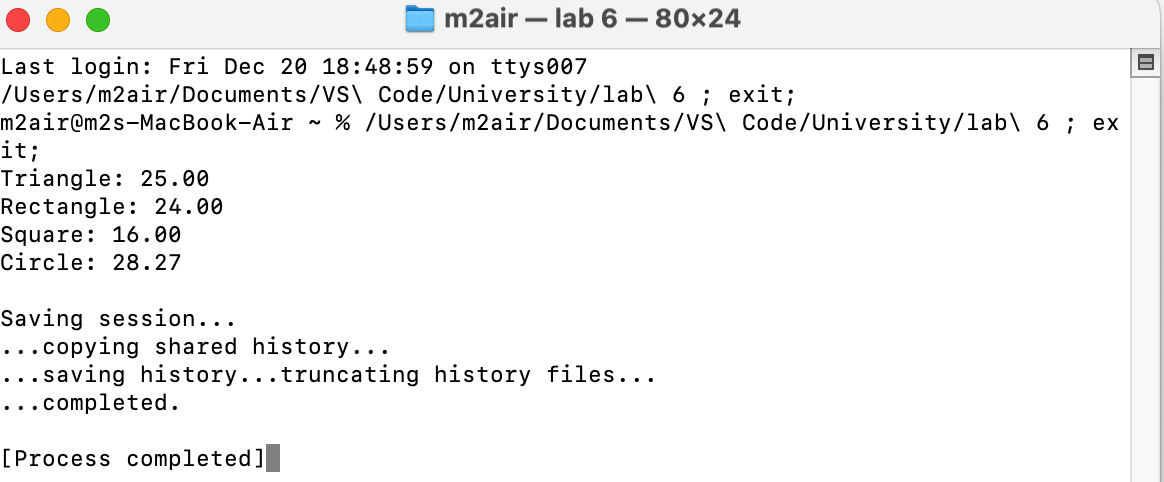
b) Use formulas:

1. Triangle: 0.5×base×height0.5 \times \text{base} \times \text{height}0.5×base×height.
2. Rectangle: length×width\text{length} \times \text{width}length×width.
3. Square: side2\text{side}^2side2.
4. Circle: π×radius2\pi \times \text{radius}^2π×radius2.

c) Print the calculated area..

|  |
| --- |
| Code |
| #include <stdio.h>  #include <math.h>  float area\_triangle(float base, float height) {  return 0.5 \* base \* height;  }  float area\_rectangle(float length, float width) {  return length \* width;  }  float area\_square(float side) {  return side \* side;  }  float area\_circle(float radius) {  return M\_PI \* radius \* radius;  }  int main() {  printf("Triangle: %.2f\n", area\_triangle(5.0, 10.0));  printf("Rectangle: %.2f\n", area\_rectangle(4.0, 6.0));  printf("Square: %.2f\n", area\_square(4.0));  printf("Circle: %.2f\n", area\_circle(3.0));  return 0;  } |

Output Result :



5. Give declaration for a function called DigitCount( ), which takes a integer and

returns the number of digit found in the given integer.

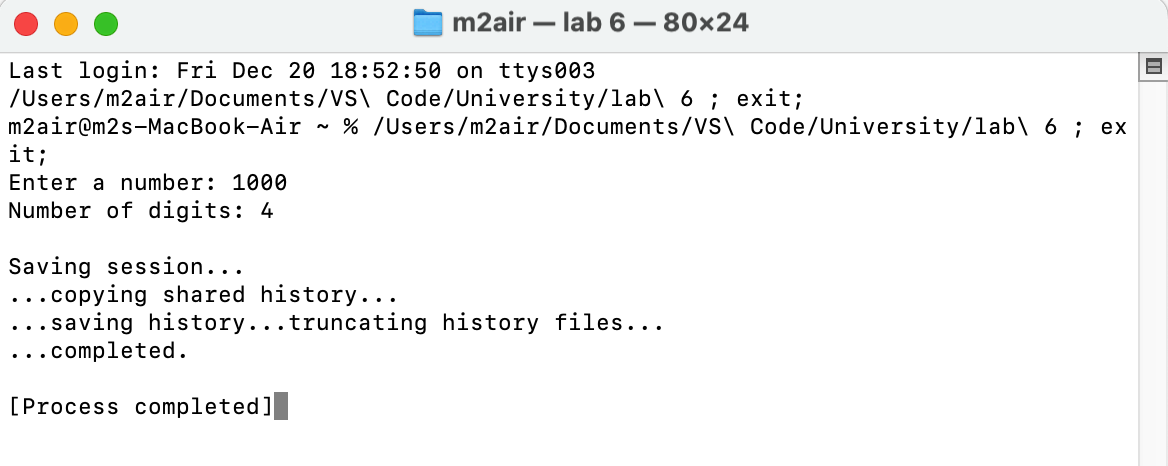
Answer :

**Algorithm:**

1. Input a number.
2. Initialize counter to 0.
3. Divide number by 10 in a loop until it’s 0, incrementing the counter.
4. Print the counter.

|  |
| --- |
| Code |
| #include <stdio.h>  int DigitCount(int num) {  int count = 0;  while (num != 0) {  num /= 10;  count++;  }  return count;  }  int main() {  int num;  printf("Enter a number: ");  scanf("%d", &num);  printf("Number of digits: %d\n", DigitCount(num));  return 0;  } |

**Output Result :**



6. Write a program to find the sum of a series of positive odd numbers using recursion.

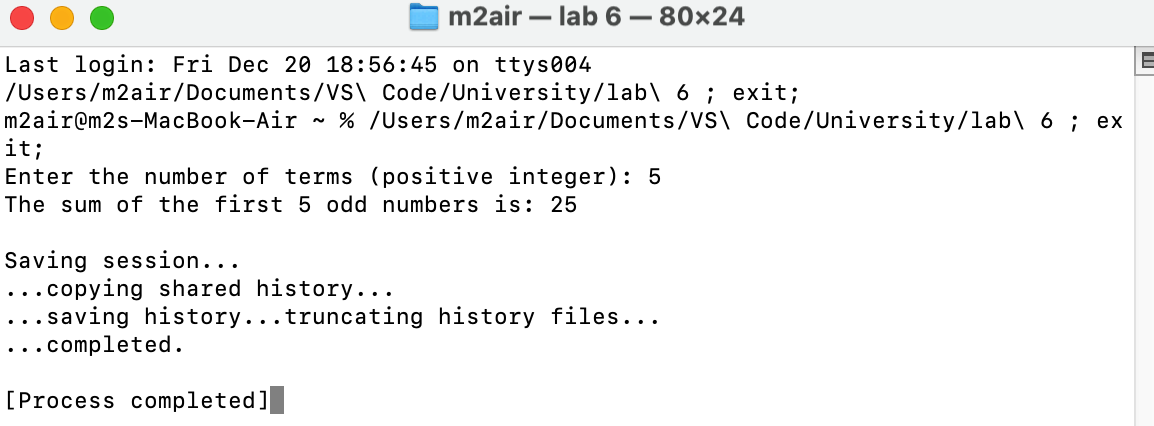
Answer :

**Algorithm:**

1. Input n.
2. If n is 0, return 0.
3. If odd, add n to the sum of the previous number.
4. Print the sum.

|  |
| --- |
| Code |
| #include <stdio.h>  int sum\_of\_odds(int n) {  if (n == 0) {  return 0; // Base case: the sum of 0 numbers is 0  }  return (2 \* n - 1) + sum\_of\_odds(n - 1);  }  int main() {  int n;  printf("Enter the number of terms (positive integer): ");  scanf("%d", &n);  if (n <= 0) {  printf("Please enter a positive integer.\n");  return 1; // Exit with an error code  }  int result = sum\_of\_odds(n);  printf("The sum of the first %d odd numbers is: %d\n", n, result);  return 0;  } |

Output Result:



7. Write a recursive function to generate Fibonacci series.

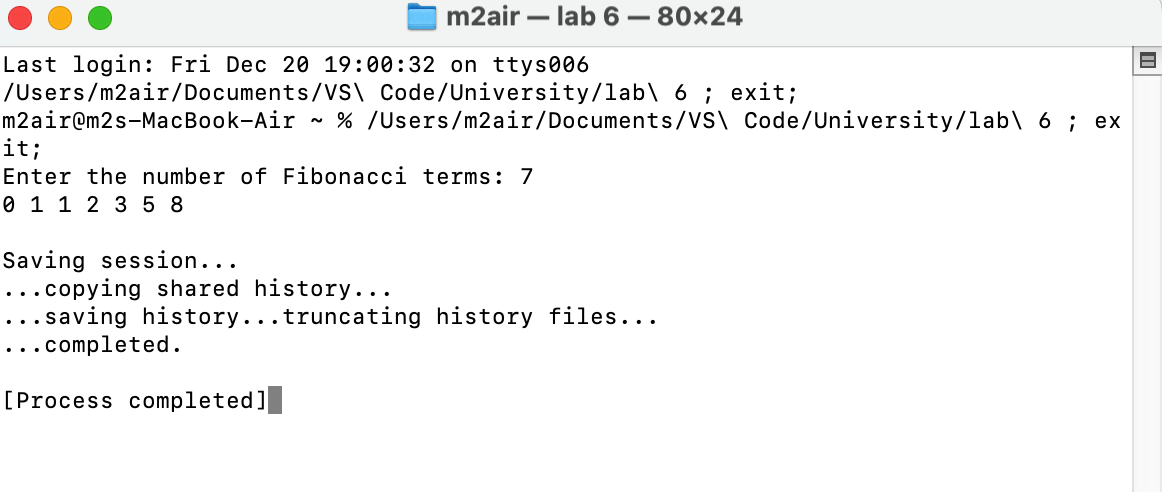
Answer :

**Algorithm:**

1. Input n.
2. If n <= 1, return n.
3. Else return the sum of the two previous Fibonacci numbers.
4. Print Fibonacci series for n terms.

|  |
| --- |
| Code |
| #include <stdio.h>  int fibonacci\_recursive(int n) {  if (n <= 1) return n;  return fibonacci\_recursive(n - 1) + fibonacci\_recursive(n - 2);  }  void print\_fibonacci(int n) {  for (int i = 0; i < n; i++) {  printf("%d ", fibonacci\_recursive(i));  }  printf("\n");  }  int main() {  int n;  printf("Enter the number of Fibonacci terms: ");  scanf("%d", &n);  print\_fibonacci(n);  return 0;  } |

Output Result :



**Discussion :** This lab assignment explores fundamental programming concepts in C, focusing on recursion, loops, and efficient algorithms. It demonstrates practical problem-solving through tasks like finding perfect and prime numbers, generating Fibonacci sequences, and calculating geometric areas. The programs emphasize modular design, efficient use of recursion, and logical breakdown of problems. Tasks such as summing odd numbers and counting digits highlight the versatility of loops and recursion, while area calculations showcase the practical application of mathematical formulas. These exercises collectively reinforce structured programming principles and enhance algorithmic thinking.